

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Пользовательское web-приложение управления метаданными
в составе информационно-управляющей платформы**

Студент _____ Савинова Кристина Дмитриевна
(подпись, дата) (ФИО полностью)

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

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

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

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, 2025

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент _____ Савинова Кристина Дмитриевна _____ гр. 21ВП1 факультета _____ ВТ _____
<Фамилия, имя, отчество полностью>

2. Тема работы _____ Пользовательское web-приложение управления метаданными в
составе информационно-управляющей платформы _____

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

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: Пользовательское web-приложение управления метаданными в
информационно-управляющей платформы _____

2) Среда выполнения: Windows 10 _____

Языковые средства разработки: TypeScript, JavaScript, UML _____

Инструментальные средства: Node.js, Visual Studio Code, Visual
Paradigm16.3. _____

Технологии разработки: ООП, сервисно-ориентированный подход. _____

3) Основные функции разрабатываемого приложения: _____

а) регистрация и авторизация пользователей; _____

б) просмотр метаданных систем, подсистем, устройств, событий; _____

в) редактирование и удаление метаданных систем, подсистем, устройств,
событий; _____

г) просмотр и редактирование активных правил в списочном, языковом и
графическом представлении; _____

е) удаление активных правил. _____

5. Перечень подлежащих разработке вопросов:

1) *Анализ предметной области и планирование процесса разработки;*

2) *Анализ требований и постановка задачи на разработку;*

3) *Разработка;*

4) *Контроль качества программного обеспечения;*

5) *Оформление пояснительной записки.*

6. Календарный график выполнения работы

п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и планирование процесса разработки	10%	08.04.2025 – 14.04.2025	
2	Анализ требований и постановка задачи на разработку	20%	15.04.2025 – 25.04.2025	
3	Разработка программного обеспечения	30%	26.04.2025 – 12.05.2025	
4	Контроль качества программного обеспечения	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 139 листов, 35 рисунков, 22 таблицы, 21 использованный источник и 7 приложений.

WEB-ПРИЛОЖЕНИЕ, УПРАВЛЕНИЕ МЕТАДААННЫМИ, ИНФОРМАЦИОННО-УПРАВЛЯЮЩАЯ ПЛАТФОРМА, FRONTEND, АКТИВНЫЕ ПРАВИЛА, СОБЫТИЯ, REACT JS, UML.

Объект исследования – пользовательское web-приложение управления метаданными в составе информационно-управляющей платформы.

Цель выпускной квалификационной работы – разработать пользовательское web-приложение управления метаданными в составе информационно-управляющей платформы.

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

Среда исполнения – Windows 10, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы было создано пользовательское web-приложение управления метаданными в составе информационно-управляющей платформы.

Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08BP211.20 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Пользовательское web-приложение управления метаданными в составе информационно-управляющей платформы Пояснительная записка	Лит.	Лист	Листов
Разраб.		Савинова К.Д.						
Пров.		Шибанов С.В.					4	139
Н. Контр.		Такташкин Д.В				Группа 21ВП1		
Утв.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и планирование процесса разработки пользовательского web-приложения управления метаданными.....	9
1.1 Low-code платформа для развертывания, эксплуатации и сопровождения информационно-управляющих систем	9
1.1.1 Назначение и основные возможности платформы	9
1.1.2 Обобщенная программная архитектура платформы	11
1.1.3 Концептуальная модель информационно-управляющей системы на основе платформы.....	13
1.2 Управление метаданными в информационно-управляющей платформе	14
1.2.1 Объектная модель метаданных платформы	14
1.2.2 Жизненный цикл метаданных при развертывании, эксплуатации и сопровождении информационно-управляющих систем	17
1.2.3 Бизнес-процессы управления при развертывании, эксплуатации и сопровождении информационно-управляющих систем	18
1.3 Сравнительный анализ пользовательских приложений управления метаданными в составе аналогичных платформ.....	20
1.3.1 Обоснование сравнительного анализа	20
1.3.2 Обзор пользовательских приложений для анализа	23
1.3.2.1 Интеграционная no-code платформа «Некстби».....	23
1.3.2.2 Low-code платформа «Losant Enterprise IoT».....	25
1.3.2.3 Облачная платформа «Azure IoT Central»	26
1.3.3 Результаты сравнительного анализа и выводы.....	28
1.4 Планирование процесса разработки приложения.....	29
1.4.1 Цели разработки и назначение приложения	29
1.4.2 Классы пользователей приложения	30
1.4.3 План-график разработки и планирование её бюджета.....	31

2 Анализ требований и постановка задачи на разработку пользовательского web-приложения управления метаданными.....	38
2.1 Анализ функциональных требований	38
2.2 Анализ нефункциональных требований	41
2.3 Анализ требований к пользовательскому интерфейсу	41
2.4 Анализ требований к аппаратно-программному обеспечению.....	43
2.4.1 Требования к аппаратному обеспечению	43
2.4.2 Требования к системному программному обеспечению	43
2.4.3 Требования к средствам разработки	44
2.5 Постановка задачи на разработку приложения.....	46
3 Разработка пользовательского web-приложения управления метаданными...	47
3.1 Разработка архитектуры развертывания приложения.....	47
3.2 Разработка пользовательского интерфейса приложения	51
3.2.1 Разработка сценариев взаимодействия пользователя с приложением	51
3.2.2 Разработка визуальных прототипов пользовательского интерфейса	54
3.3 Разработка программного обеспечения приложения	60
3.3.1 Разработка программных модулей уровня модели	60
3.3.2 Разработка программных модулей уровня контроллера	61
3.3.3 Разработка программных модулей уровня представления.....	63
4 Контроль качества программного обеспечения пользовательского web-приложения управления метаданными.....	66
4.1 Планирование процесса контроля качества приложения	66
4.1.1 Мероприятия по контролю качества.....	66
4.1.2 Обоснование выбора модели тестирования	67
4.1.3 Обоснование выбора метрик программного кода	69
4.1.4 Обоснование выбора методик оценки качества пользовательского интерфейса	70
4.2 Тестирование приложения	71

4.2.1 Функциональное тестирование.....	71
4.2.2 Конфигурационное тестирование	74
4.3 Оценка качества программного кода	76
4.3.1 Метрики устойчивости и абстрактности	76
4.3.2 Метрики Холстеда	80
4.4 Оценка качества пользовательского интерфейса	83
Заключение	86
Список использованных источников	88
Приложение А (справочное) Глоссарий	90
Приложение Б (обязательное) Реализация моделей.....	92
Приложение В (обязательное) Реализация контроллера компонента системы.....	95
Приложение Г (обязательное) Реализация представления компонента системы.....	100
Приложение Д (рекомендуемое) Визуальные прототипы пользовательского интерфейса	119
Приложение Е (рекомендуемое) Результаты функционального тестирования.....	124
Приложение Ж (рекомендуемое) Результаты конфигурационного тестирования.....	133

Введение

В работе с большими объемами информации информационно-управляющие системы играют важную роль. Такие системы позволяют не только собирать и обрабатывать данные с различных устройств, но и анализировать их, обеспечивая мониторинг и управление процессами.

В рамках выпускной квалификационной работы информационно-управляющая система рассматривается как сервис потоковой обработки событий, который в реальном времени анализирует и обрабатывает поступающие данные. Способность таких сервисов своевременно реагировать на события, отправлять уведомления и передавать события другим системам, делает их незаменимыми в условиях быстро меняющейся информационной среды.

Для эффективной работы с растущими объемами данных необходимы средства управления метаданными. В свою очередь, метаданные представляют собой важные сведения об объектах, которые помогают структурировать данные, а также упрощают поиск и анализ информации.

Для обеспечения эффективного управления метаданными возникает необходимость в создании web-приложения, которое будет предоставлять пользователю широкий функционал при работе с информационно-управляющей платформой. Приложение должно обеспечивать интуитивно понятный интерфейс, который позволит пользователям легко взаимодействовать с событиями, устройствами и активными правилами.

Целью выпускной квалификационной работы является разработка пользовательского web-приложения управления метаданными в составе информационно-управляющей платформы. Работа включает в себя анализ предметной области, анализ и разработку требований, разработку прототипов интерфейса и web-приложения, тестирование и оценку качества программного кода и пользовательского интерфейса.

1 Анализ предметной области и планирование процесса разработки пользовательского web-приложения управления метаданными

1.1 Low-code платформа для развертывания, эксплуатации и сопровождения информационно-управляющих систем

1.1.1 Назначение и основные возможности платформы

Информационно-управляющая система – цифровая система контроля или управления некоторым реальным объектом, называемым «объектом управления» [1]. Для развертывания, эксплуатации и сопровождения информационно-управляющих систем была разработана low-code платформа, предоставляющая готовые компоненты и шаблоны для создания определенного решения. В контексте данной разработки информационно-управляющая система представляет собой сервис потоковой обработки событий и исполнения активных правил. Сервис предназначен для сбора, обработки, анализа и своевременного реагирования на поступающие потоки данных в реальном времени.

Работа платформы основана на модели Event-Condition-Action (ECA). Правила этого типа содержат три части:

- событие, по которому активируется правило;
- проверяемое условие;
- выполняемое действие [2].

Модель ECA позволяет реагировать на события при возникновении условия, которое заранее определяется пользователем. Событие – изменение состояния объекта в определенный момент времени. Условие – логическое выражение, которое проверяется по данным поступающего события. Действие – команда, выполняющаяся при истинности условия, связанного с событием.

Платформа может применяться в различных отраслях, где необходимо отслеживать состояние устройств и своевременно реагировать на изменения, так как представляет собой инструмент для эффективной обработки

непрерывных потоков данных, поступающих в режиме реального времени. Информационно-управляющая система может быть полезна в следующих сферах:

- промышленность: наблюдение за состоянием оборудования, условиями в производственных помещениях;
- логистика: мониторинг условий перевозок в грузовых отсеках товаров, требующих особых условий хранения;
- здравоохранение: отслеживание показателей здоровья пациентов, мониторинг состояния медицинского оборудования, контроль условий хранения медикаментов;
- система «Умный дом»: управление устройствами в доме, наблюдение за измерениями датчиков и устройств, предупреждение опасных ситуаций.

Для эффективного управления метаданными в информационно-управляющей системе используются три группы репозиториев метаданных: глобальные, оперативные, репозитории журнализации. Глобальные репозитории содержат данные обо всех системах, устройствах, событиях, активных правилах и пользователях. Оперативные репозитории служат отдельными хранилищами данных для функционирования каждого компонента. Репозитории журнализации содержат общую информацию о работе системы: сведения о выполненных действиях, поступивших событиях.

Основные функциональные возможности платформы следующие:

- создание, редактирование и исполнение активных правил;
- управление метаданными событий и устройств;
- отслеживание поступающих событий и их обработка;
- журнализация обработанных событий и процессов исполнения активных правил;
- создание и управление учётными записями пользователей и их правами доступа;

- синхронизация метаданных во всех компонентах платформы.

Таким образом, платформа значительно упрощает процесс развертывания, эксплуатации и сопровождения информационно-управляющих систем.

1.1.2 Обобщенная программная архитектура платформы

При проектировании платформы использовались принципы микросервисной и событийно-ориентированной архитектуры.

Микросервисная архитектура представляет собой подход к разработке единого приложения в виде набора небольших сервисов, работающих в отдельных процессах и взаимодействующих с применением упрощенных механизмов [3]. Событийно-ориентированная архитектура состоит из независимых и специализированных компонентов обработки событий, которые асинхронно получают и обрабатывают события [4]. Использование микросервисной и событийно-ориентированной архитектуры предполагает независимое масштабирование сервисов, высокую отказоустойчивость, ускоренную разработку.

Архитектура платформы для развертывания, эксплуатации и сопровождения информационно-управляющих систем представлена на рисунке 1.

Платформа состоит из двух основных частей: портала управления и сервиса потоковой обработки событий и исполнения активных правил.

Портал управления представляет собой пользовательское приложение для управления метаданными. Портал управления позволяет пользователям, в зависимости от их роли в системе, управлять компонентами самой системы, создавать и изменять объекты метаданных.

Сервис потоковой обработки событий и исполнения активных правил включает в себя набор микросервисов и их локальных репозиторий, центральный репозиторий метаданных, REST-сервер и брокер сообщений.

Каждый микросервис выполняет свою конкретную функцию: обрабатывает входные события, обнаруживает события разных видов, обеспечивает журнализацию событий. Для каждого микросервиса реализован локальный репозиторий, который повышает производительность платформы, за счёт уменьшения времени получения метаданных событий микросервисом.

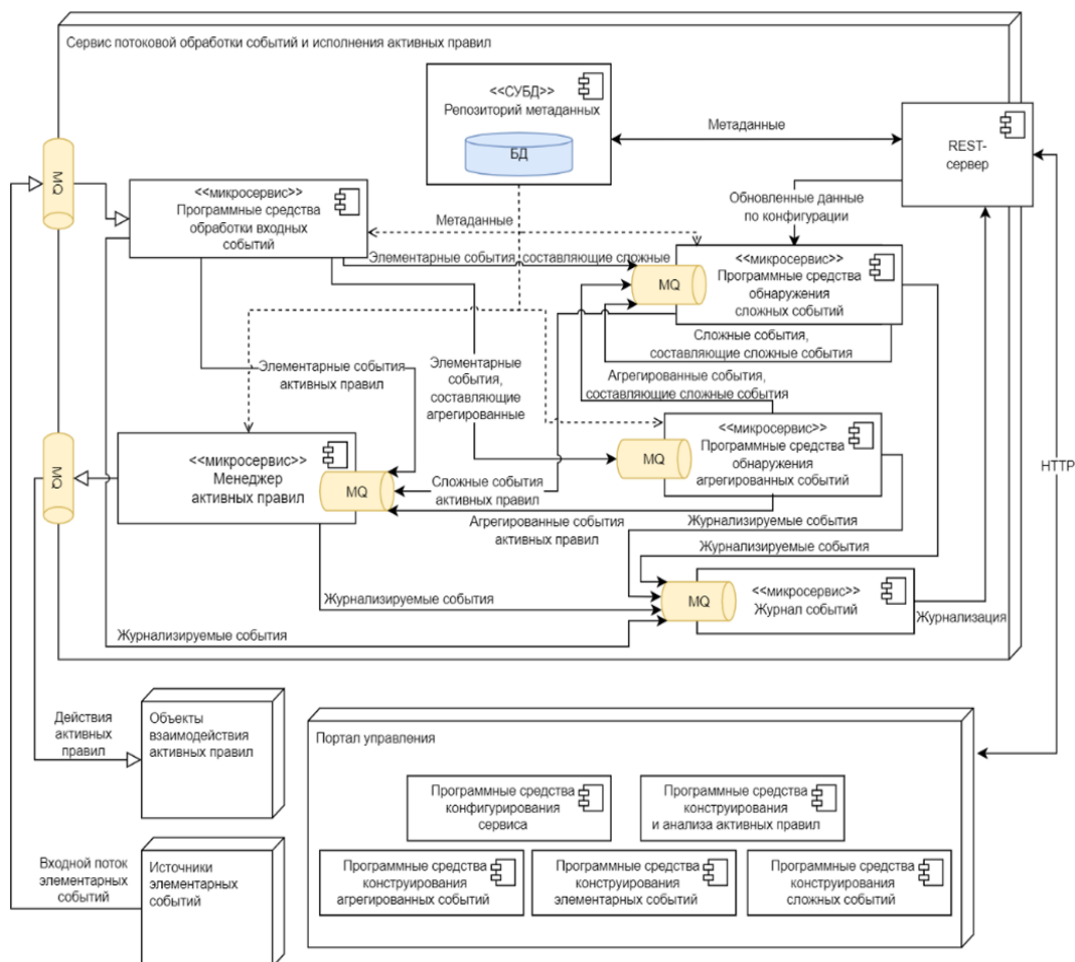


Рисунок 1 – Архитектура платформы

Для получения данных платформа использует брокер сообщений, который управляет очередью сообщений, собирает и перенаправляет события в нужный сервис. Основные преимущества использования брокера сообщений заключаются в повышении производительности и отказоустойчивости, гарантированной доставки и маршрутизации сообщений.

Центральный репозиторий представляет собой глобальное хранилище метаданных объектов. Метаданные могут представлять собой информацию о событиях, активных правилах, системах и их компонентах.

REST-сервер отвечает за конструирование метаданных со стороны сервиса и синхронизацию центрального и локального репозитория. Для каждого класса метаданных разработан соответствующий сервис с набором сценариев.

1.1.3 Концептуальная модель информационно-управляющей системы на основе платформы

Практически все данные создаются как непрерывные потоки событий. Любое приложение, которое обрабатывает поток событий, должно учитывать состояния с возможностью хранения промежуточных данных и доступа к ним. Когда приложение получает событие, оно может выполнять произвольные вычисления, которые включают чтение или запись состояния.

Приложения потоковой обработки событий получают входящие события из журнала событий. Журнал событий хранит и распределяет потоки событий. События записываются в долговременный журнал, доступный только для добавления [5].

Диаграмма деятельности информационно-управляющей системы на основе платформы представлена на рисунке 2.

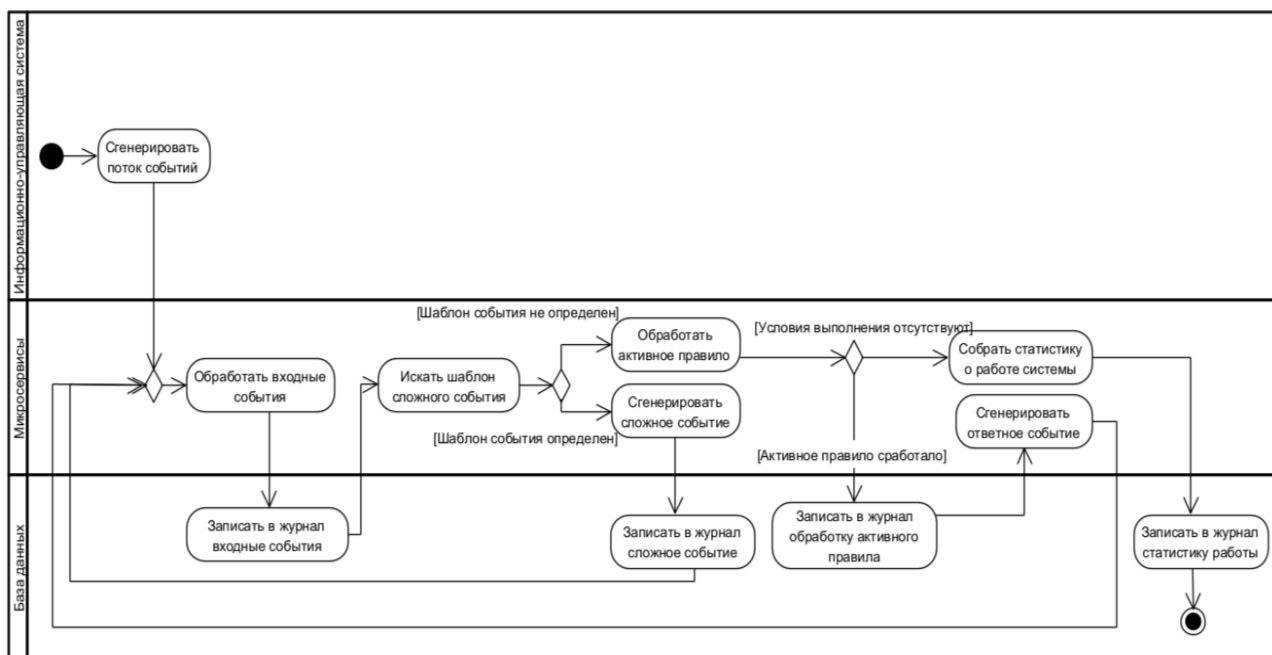


Рисунок 2 – Поток управления в информационно-управляющей системе

Сгенерированный поток событий поступает из наблюдаемой системы, после чего передается микросервисам для дальнейшей обработки. Микросервисы проверяют корректность поступивших данных и ищут совпадения с заранее созданными шаблонами событий. В случае, если шаблон определен, то генерируется сложное событие, которое направляется на повторную обработку. Все события передаются менеджеру активных правил, который при выполнении заданных условий выполняет соответствующее правило. В журнал записывается статистика о работе системы и входные события для дальнейшего анализа бизнес-аналитиками.

Для обработки поступающих событий используется технология Event Stream Processing (ESP). Технология позволяет обрабатывать потоки данных в режиме реального времени, обнаруживать шаблоны событий и реагировать на них с минимальной задержкой.

1.2 Управление метаданными в информационно-управляющей платформе

1.2.1 Объектная модель метаданных платформы

Метаданные – это данные, описывающие содержание и структуру объекта [6].

На платформе используются метаданных трёх видов:

- метаданные внешних систем (содержат наблюдаемые системы и системы управления);
- метаданные событий (содержат элементарные и сложные события);
- метаданные активных правил (содержат классические, расширенные, многоблочные и многофункциональные правила).

Структура метамодели пользовательского web-приложения управления метаданными в составе информационно-управляющей платформы представлена на рисунке 3.

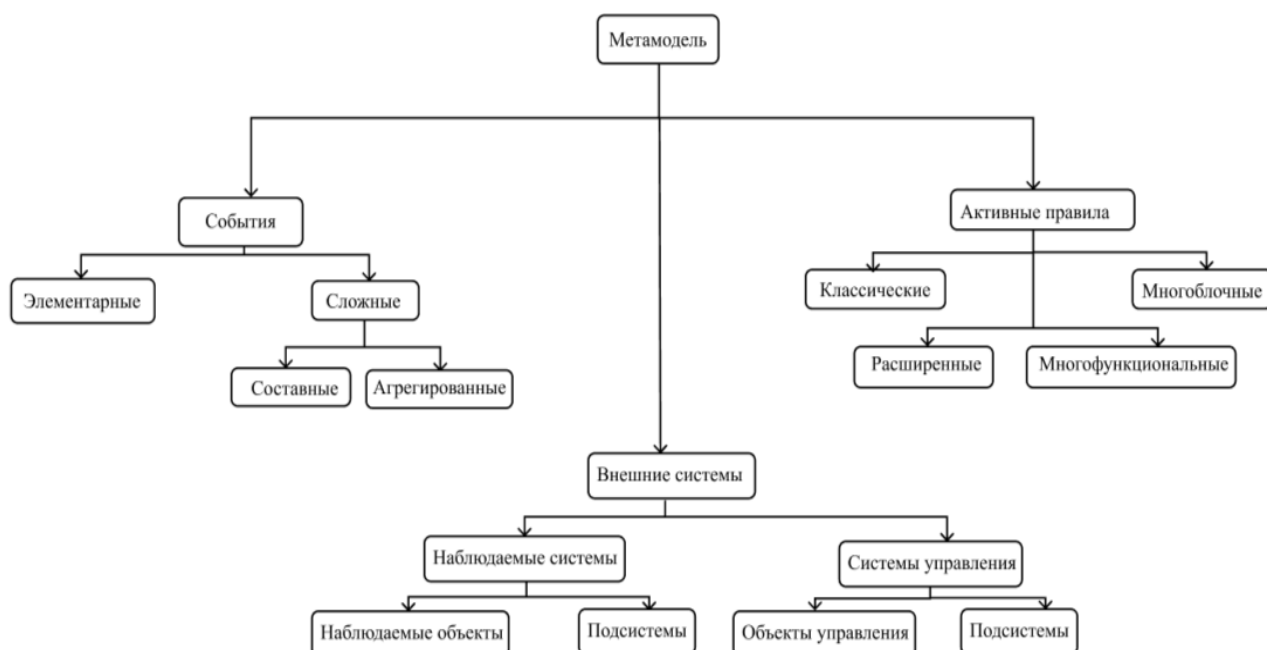


Рисунок 3 – Структура метамодели

Внешние системы внутри платформы делятся на два вида: наблюдаемые системы и системы управления. Наблюдаемыми объектами могут быть устройства и датчики, они являются первоисточниками данных и событий. Объектами управления во внешних системах также могут быть устройства, к которым могут быть применены заранее описанные действия при возникновении определенного события. Данные от наблюдаемых объектов формируют входящий поток событий; данные от объектов управления – выходной поток действий.

События делятся на два вида: элементарные и сложные. В свою очередь, сложные события делятся на агрегированные и составные.

Элементарное событие – простейшее событие, представляющее собой мгновенное появление в системе какого-либо изменения (например, вход пользователя в аккаунт). Кортеж элементарного события состоит из уникального идентификатора события, набора контекстных параметров, объекта наблюдения, типа события и времени возникновения события.

Агрегированное событие – сложное событие, состоящее из нескольких элементарных событий и операторов. Кортеж агрегированного события состоит

из уникального идентификатора события, набора контекстных параметров, запроса, типа события, времени начала и конца события.

Составное событие – сложное событие, состоящее из нескольких событий с использованием комбинаций последовательности событий. Кортеж составного события состоит из уникального идентификатора события, набора контекстных параметров, комбинации событий, типа события, времени начала и конца события.

Кортеж классического активного правила состоит из типа события, условия и действия. Кортеж расширенного активного правила состоит из типа события, условия, действия при истинности условия и альтернативного действия при ложности условия.

Кортеж многоблочного активного правила состоит из множества результирующих событий, множества условий и множества действий. Многофункциональные активные правила используются при возникновении события, вызывающего выполнение множества действий при истинности условия.

На рисунке 4 представлена диаграмма классов, показывающая объектную модель метаданных.

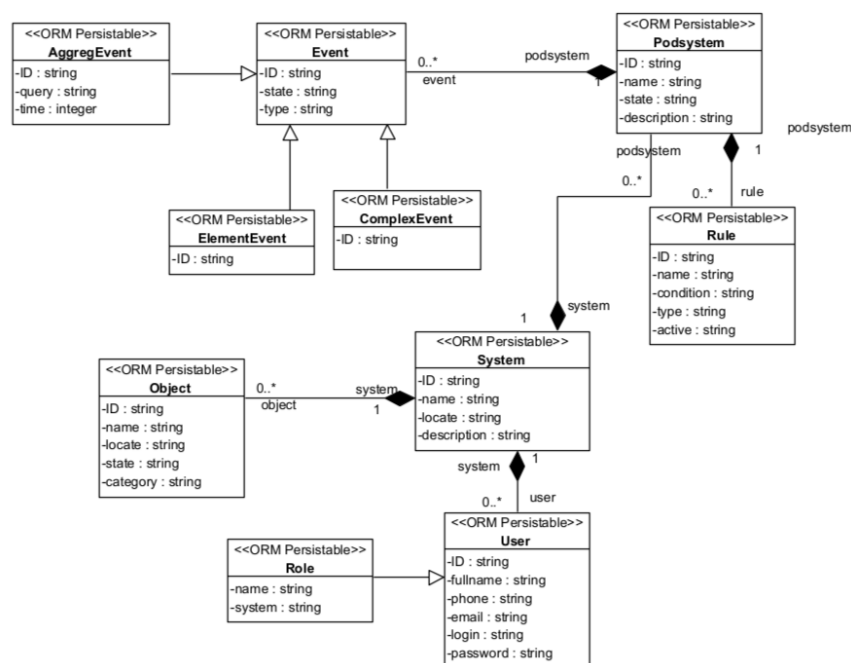


Рисунок 4 – Объектная модель метаданных

В таблице 1 представлены спецификации объектов метамодели.

Таблица 1 – Спецификации объектов модели

Объект	Описание
User	Содержит метаданные пользователей системы
Role	Содержит метаданные ролей, которые определяют права доступа
Event	Содержит информацию о событиях
ElementEvent	Содержит метаданные элементарных событий
ComplexEvent	Содержит метаданные составных событий
AggregEvent	Содержит метаданные агрегированных событий
Rule	Содержит метаданные активных правил и условий их выполнения
System	Содержит объекты наблюдаемых систем и систем управления
Podsystem	Содержит метаданные подсистем
Object	Содержит метаданные устройств наблюдения и управления

Для хранения метаданных всех событий, активных правил и внешних систем используется центральный репозиторий.

1.2.2 Жизненный цикл метаданных при развертывании, эксплуатации и сопровождении информационно-управляющих систем

Жизненный цикл метаданных охватывает несколько этапов, начиная с их создания и заканчивая их удалением.

Этапы жизненного цикла метаданных следующие: инициализация метаданных, анализ изменений, тиражирование и синхронизация, журнализация и деактуализация метаданных (рисунок 5).



Рисунок 5 – Жизненный цикл метаданных

Этап инициализации метаданных: определение и описание наблюдаемых объектов и объектов управления, подсистем, событий, активных правил и условий их активации. На данном этапе происходит сохранение метаданных в центральный репозиторий.

Этап анализа изменений метаданных: проверка согласованности с другими объектами метаданных, выявление и устранение коллизий в метаданных.

Этап тиражирования и синхронизации является ключевым этапом в жизненном цикле метаданных. Тиражирование метаданных используется для обеспечения единства данных между центральным репозиторием и локальными хранилищами. Синхронизация метаданных необходима для поддержания актуальности метаданных во всех компонентах.

Этап журнализации: запись метаданных для дальнейшего анализа бизнес-аналитиками.

Этап деактуализации: удаление данных, информация о которых устарела и больше не актуальна. Метаданные удаляются из центрального репозитория и локальных хранилищ.

1.2.3 Бизнес-процессы управления при развертывании, эксплуатации и сопровождении информационно-управляющих систем

Пользовательское web-приложение используется для управления метаданными информационно-управляющих систем.

Для разграничения прав доступа к управлению метаданными были созданы следующие роли пользователей: разработчик, главный администратор, системный аналитик, администратор и бизнес-аналитик.

В обязанности разработчика входит создание и изменение метаданных событий и активных правил. Главный администратор имеет функционал всех ролей, также может управлять метаданными систем. Системный аналитик управляет системами, создавая и редактируя подсистемы и объекты управления и наблюдения. Администратор отвечает за работу с метаданными

пользователей, имеет возможность изменять роль пользователя в системе. Бизнес-аналитик анализирует произошедшие события и выполненные правила.

Процесс управления метаданными в виде диаграммы деятельности, представлен на рисунке 6.

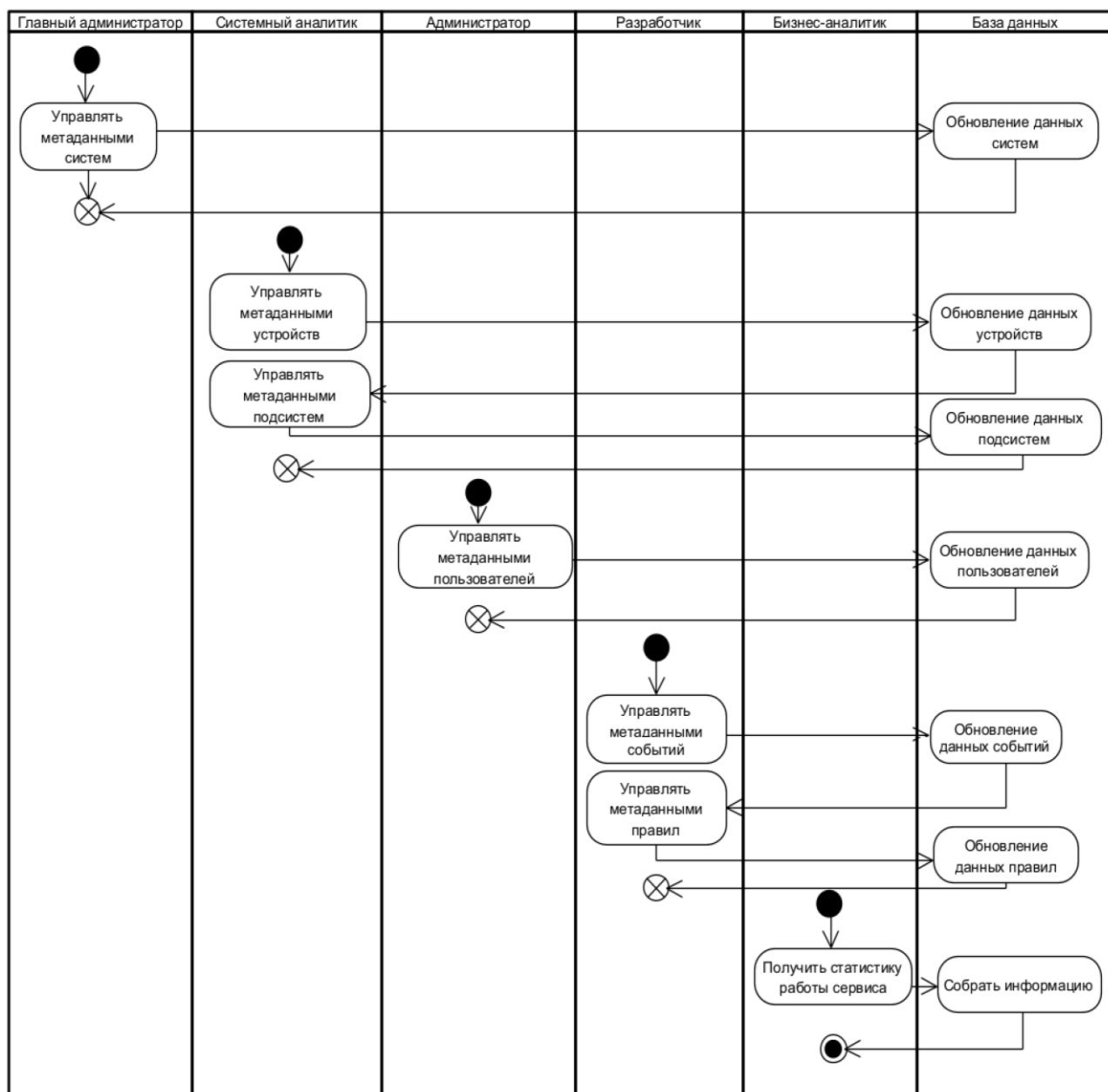


Рисунок 6 – Процесс управления метаданными

Процесс управления метаданными основан на принципе CRUD (создание, чтение, обновление и удаление), то есть включает в себя следующие действия: создание, редактирование и мониторинг данных. При создании или изменении метаданных система проверяет сохраняемые данные на корректность.

1.3 Сравнительный анализ пользовательских приложений управления метаданными в составе аналогичных платформ

1.3.1 Обоснование сравнительного анализа

Сравнительный анализ – процесс сопоставления объектов, выполняющих схожие функции, для определения их характеристик, выявления сильных и слабых сторон.

Основные цели сравнительного анализа:

- определить преимущества и недостатки существующих приложений;
- повысить конкурентоспособность продукта;
- оценить потенциальные риски;
- улучшить процесс разработки на основе опыта конкурентов.

Для проведения сравнительного анализа необходимо выполнить следующие задачи: выбрать приложения для сравнения, определить критерии сравнения, изучить выбранные приложения, сравнить приложения по выбранным критериям и проанализировать результаты сравнения.

Сравнительный анализ пользовательских приложений управления метаданными был проведен по следующим критериям: удобство использования, производительность, функциональные возможности и управление правами доступа.

Критерий «Удобство использования» показывает, насколько интуитивно понятен и удобен в использовании интерфейс. Критерий оценивается по балльной шкале от 0 до 5. Описание оценок критерия представлено в таблице 2.

Таблица 2 – Описание оценок критерия «Удобство использования»

Оценка	Описание оценки
0	Интерфейс имеет сложную навигацию, для выполнения задачи пользователю необходимо сделать большое количество шагов, интерфейс не предоставляет полную информацию об объектах.
1	Интерфейс имеет сложную навигацию, для выполнения задачи пользователю необходимо сделать большое количество шагов, интерфейс предоставляет полную информацию об объектах.

Продолжение таблицы 2

Оценка	Описание оценки
2	Интерфейс имеет понятную навигацию, для выполнения задачи пользователю необходимо сделать большое количество шагов, интерфейс не предоставляет полную информацию об объектах.
3	Интерфейс имеет понятную навигацию, для выполнения задачи пользователю необходимо сделать большое количество шагов, интерфейс предоставляет полную информацию об объектах.
4	Интерфейс имеет понятную навигацию, для выполнения задачи пользователю достаточно сделать минимальное количество шагов, интерфейс не предоставляет полную информацию об объектах.
5	Интерфейс имеет понятную навигацию, для выполнения задачи пользователю достаточно сделать минимальное количество шагов, интерфейс предоставляет полную информацию об объектах.

Критерий «Производительность» показывает, насколько быстро работает приложение. Критерий оценивается по балльной шкале от 0 до 3. Описание оценок критерия представлено в таблице 3.

Таблица 3 – Описание оценок критерия «Производительность»

Оценка	Описание оценки
0	Страницы загружаются долго, интерфейс имеет медленную реакцию на действия пользователя, приложение работает медленно с большими объемами данных.
1	Страницы загружаются быстро, интерфейс имеет медленную реакцию на действия пользователя, приложение работает медленно с большими объемами данных.
2	Страницы загружаются быстро, интерфейс имеет быструю реакцию на действия пользователя, приложение работает медленно с большими объемами данных.
3	Страницы загружаются быстро, интерфейс имеет быструю реакцию на действия пользователя, приложение работает эффективно с большими объемами данных.

Критерий «Функциональные возможности» определяет наличие функций, соответствующих потребностям пользователей. Критерий оценивается по балльной шкале от 0 до 6. Описание оценок критерия представлено в таблице 4.

Таблица 4 – Описание оценок критерия «Функциональные возможности»

Оценка	Описание оценки
0	Отсутствует инструмент работы с активными правилами, инструмент управления устройствами и источниками данных.
1	Имеется инструмент работы с правилами только в одном виде представления, инструмент управления устройствами и источниками данных отсутствует.
2	Имеется инструмент работы с активными правилами только в одном виде представления; инструмент управления устройствами и источниками данных.
3	Имеется инструмент работы с правилами в двух видах представлений, инструмент управления устройствами и источниками данных отсутствует.
4	Имеется инструмент работы с активными правилами в двух видах представлений, инструмент управления устройствами и источниками данных.

Продолжение таблицы 4

Оценка	Описание оценки
5	Имеется инструмент работы с активными правилами в списковом, языковом и графическом представлении, инструмент управления устройствами и источниками данных отсутствует.
6	Имеется инструмент работы с активными правилами в списковом, языковом и графическом представлении, инструмент управления устройствами и источниками данных.

Критерий «Управление правами доступа» определяет наличие ограничения доступа к метаданным и функциям в соответствии с ролями пользователей. Критерий оценивается по балльной шкале от 0 до 2. Описание оценок критерия представлено в таблице 5.

Таблица 5 – Описание оценок критерия «Управление правами доступа»

Оценка	Описание оценки
0	Приложение не позволяет назначать пользователям различные права доступа в зависимости от их роли. Администратор не имеет возможности редактировать права доступа пользователей.
1	Приложение позволяет назначать пользователям различные права доступа в зависимости от их роли. Администратор не имеет возможности редактировать права доступа пользователей.
2	Приложение позволяет назначать пользователям различные права доступа в зависимости от их роли. Администратор имеет возможность редактировать права доступа пользователей.

Общая информация о платформах и приложениях аналогичного назначения представлена в таблице 6.

Таблица 6 – Информация о платформах аналогичного назначения

Название платформы	Общая информация
Некстби	Интеграционная no-code платформа для цифровой трансформации бизнеса. Платформа предоставляет полный цикл работы с данными, возможность принимать решения в режиме реального времени на основе поступающих данных и событий.
Losant Enterprise IoT	Корпоративная low-code платформа Интернета вещей, обеспечивающая возможность потоковой передачи данных в реальном времени. Имеет редактор рабочего интерфейса с функцией перетаскивания компонентов.
Azure IoT Central	Облачная платформа для разработки и управления приложениями Интернета вещей. Платформа позволяет работать с устройствами и их аналитикой, используя готовые шаблоны.
Rest Api OpenHAB	Платформа с открытым исходным кодом, работающая с домашними устройствами. Передает устройствам запрос на выполнение действий, заранее определенных пользователем.

Продолжение таблицы 6

Название платформы	Общая информация
Cisco IoT	Система, обеспечивающая внедрение и развитие Интернета вещей. Предлагает модульный подход с различными программными и аппаратными компонентами, которые могут быть объединены для создания пользовательских решений.

При проведении сравнительного анализа были рассмотрены следующие платформы: «Некстби», «Losant Enterprise IoT», «Azure IoT Central».

1.3.2 Обзор пользовательских приложений для анализа

1.3.2.1 Интеграционная no-code платформа «Некстби»

«Некстби» – платформа для автоматизации операционного управления компаний. «Некстби» использует облачные технологии и микросервисную архитектуру, обладает отказоустойчивой конфигурацией. Платформа содержит множество эффективных инструментов для автоматизации и улучшения бизнес-процессов:

- инструмент обработки и анализа данных, событий с возможностью настройки реакций на события;
- инструмент для хранения данных в виде визуальных объектов, графов, таблиц;
- инструмент создания активных правил;
- инструмент для аутентификации пользователей и разграничению прав доступа к данным платформы;
- инструмент формирования отчётов;
- инструмент планирования и администрирования.

«Некстби» является гибкой no-code платформой, которая предоставляет пользователю готовые блоки и шаблоны для выполнения задач. Платформа обладает удобным и понятным пользовательским интерфейсом, поддерживает вертикальное и горизонтальное масштабирование. Интерфейсы страниц редактирования активных правил и настройки прав доступа пользователей приведены на рисунках 7, 8.

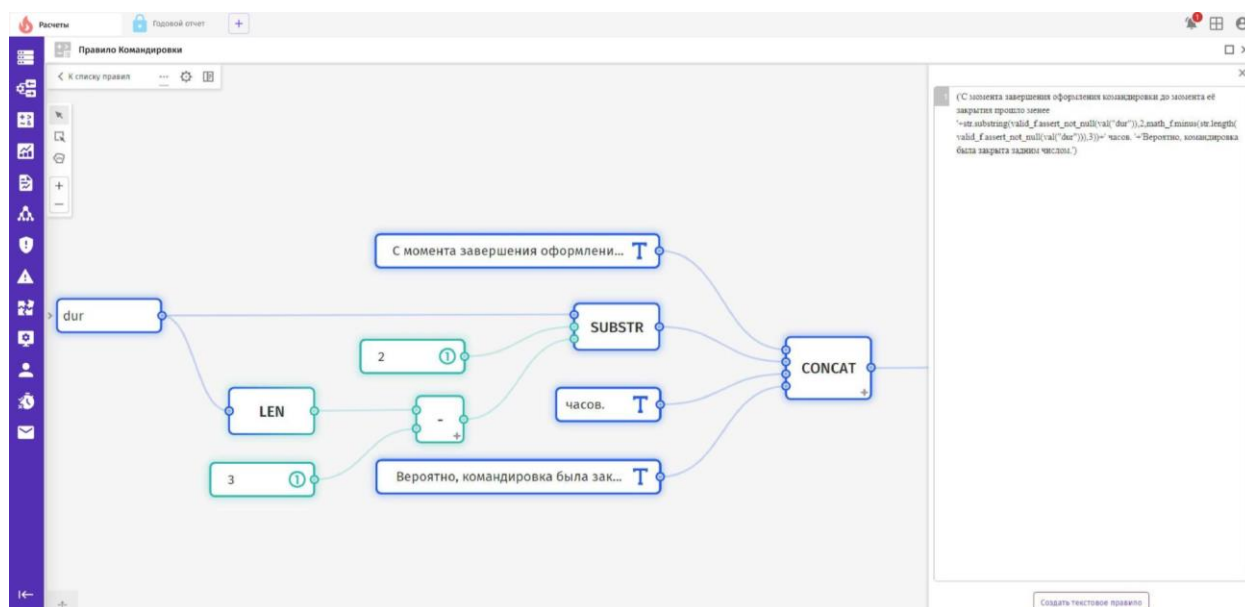


Рисунок 7 – Интерфейс страницы редактирования правила

Пользователь может создавать активные правила с помощью конструктора, состоящего из графических блоков и связей. Также платформа позволяет создавать правила с помощью текста.

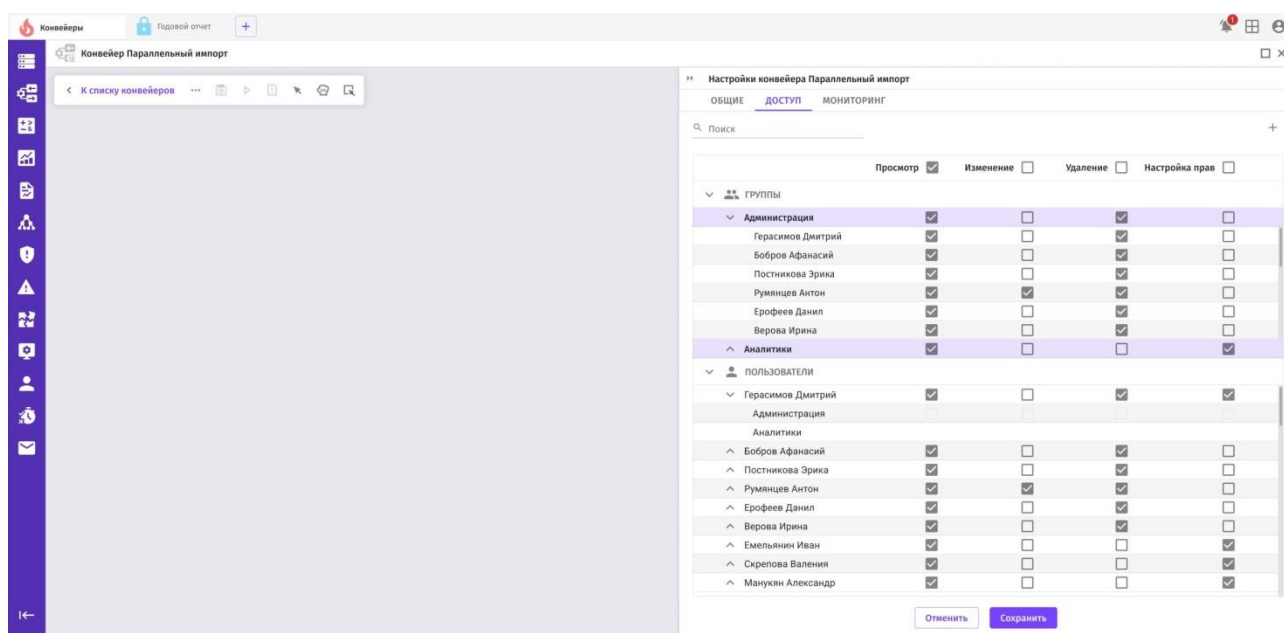


Рисунок 8 – Интерфейс страницы настройки прав доступа

Платформа предоставляет возможность просмотра пользователей и изменения их прав. Пользовательский интерфейс платформы состоит из трех основных частей: в верхней части отображены открытые страницы, кнопка

аккаунта и уведомлений; в левой части содержится панель инструментов; основная часть – область для работы.

1.3.2.2 Low-code платформа «Losant Enterprise IoT»

«Losant Enterprise IoT» – платформа, предоставляющая набор инструментов для подключения, визуализации устройств и автоматизации работы с ними. Платформа предоставляет пользователям следующий функционал:

- low-code разработка;
- информационные панели и аналитика;
- управление устройствами и источниками данных;
- многофакторная аутентификация;
- ведение журнала платформы в реальном времени, в котором отображаются основные события;
- организация рабочих процессов, которые позволяют отправлять уведомления, создавать события, отправлять команды устройствам.

Интерфейс главной страницы платформы приведен на рисунке 9. В левой части страницы располагается панель инструментов, правая часть страницы является областью для работы с платформой.

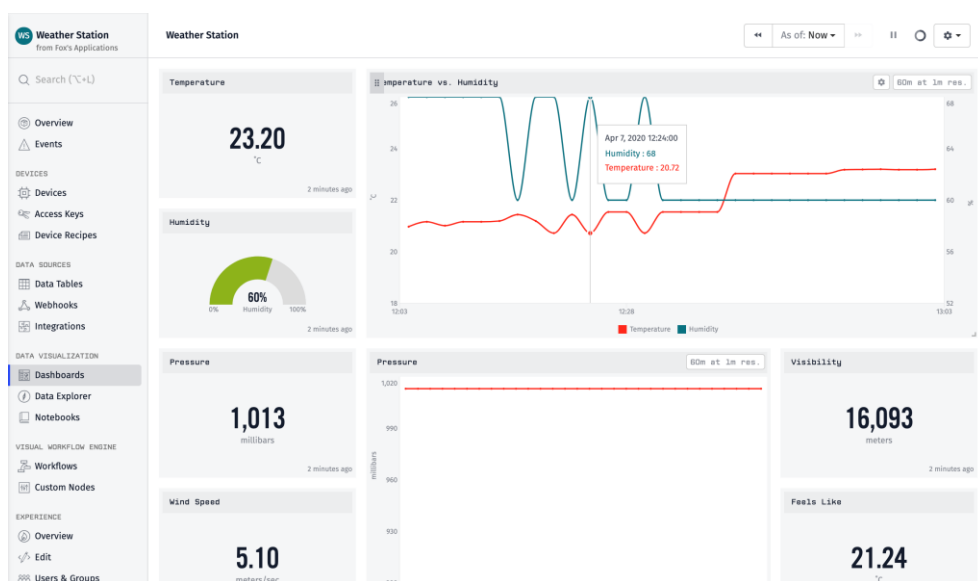


Рисунок 9 – Интерфейс главной страницы

Интерфейс страницы создания правила представлен на рисунке 10.

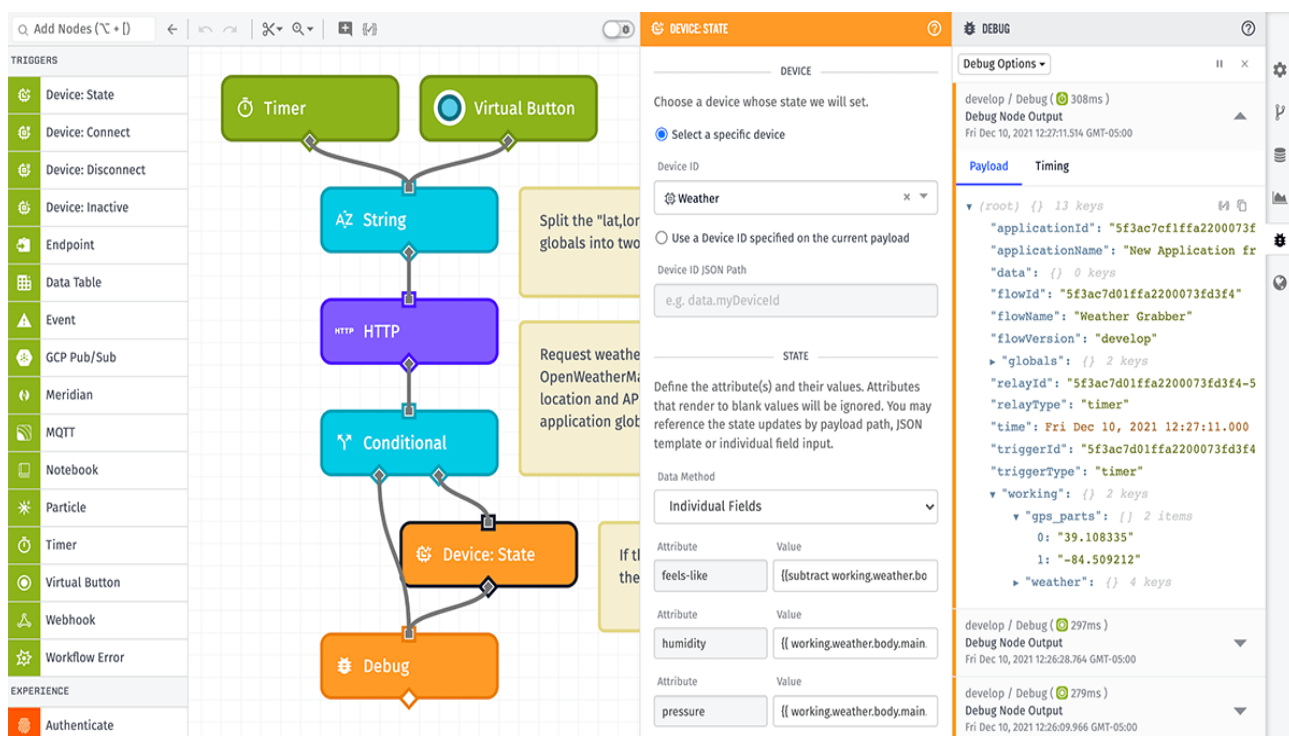


Рисунок 10 – Интерфейс страницы создания правила

Платформа позволяет создавать правила с помощью визуального конструктора. Также возможно редактирование правила с помощью изменения значений параметров: пользователю не приходится писать код для создания правила, достаточно ввести значения в заранее определенные параметры.

1.3.2.3 Облачная платформа «Azure IoT Central»

«Microsoft Azure IoT Central» – облачная платформа, позволяющая создавать и разворачивать решения для Интернета вещей.

Основные характеристики платформы:

- быстрая разработка (платформа предоставляет готовые шаблоны приложений);
- простая интеграция устройств (поддержка широко спектра устройств и датчиков);
- визуализация данных (панели мониторинга и отчёты);
- управление устройствами, правилами и действиями.

Интерфейс страницы редактирования устройства приведен на рисунке 11. Пользователь может внести изменения в параметры устройства, наблюдать за изменением значений измерений устройства в реальном времени.

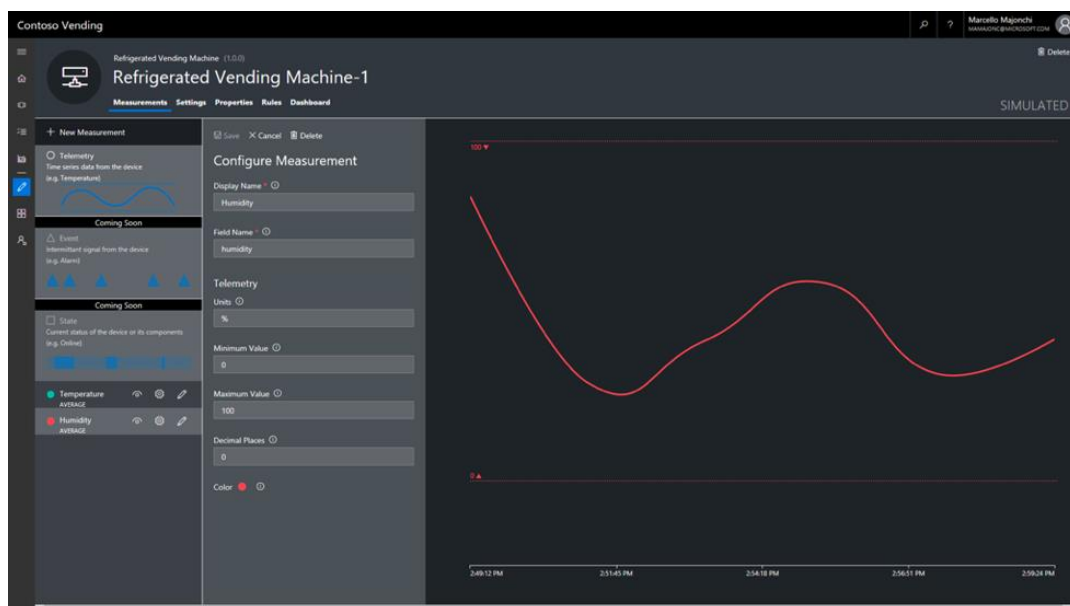


Рисунок 11 – Интерфейс страницы редактирования устройства

На рисунке 12 приведен интерфейс страницы создания правила. Процесс создания происходит с помощью работы с текстовым конструктором.

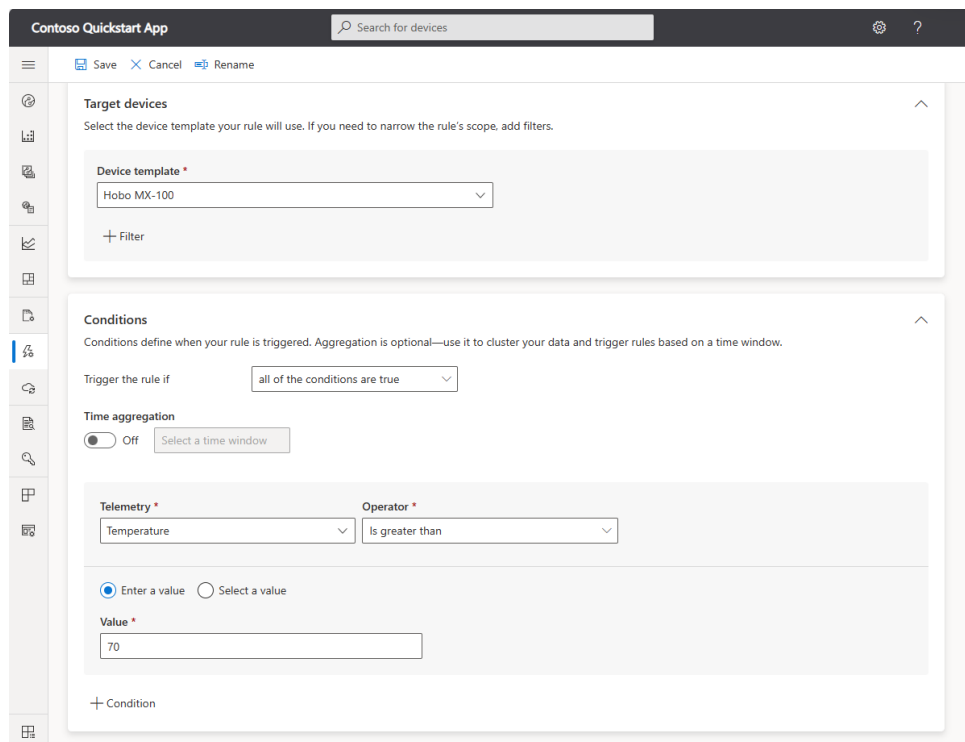


Рисунок 12 – Интерфейс страницы создания правила

«Azure IoT Central» не подходит для сложных сценариев обработки событий, требующих агрегирования данных, фильтрации, применения сложных алгоритмов.

1.3.3 Результаты сравнительного анализа и выводы

Сравнение платформ аналогичного назначения по выбранным ранее критериям представлено в таблице 7.

Таблица 7 – Сравнение платформ аналогичного назначения

	Некстби	Losant Enterprise IoT	Azure IoT Central
Удобство использования	4	3	3
Производительность	3	3	2
Функциональные возможности	4	4	4
Управление правами доступа	2	1	1

Более удобный интерфейс имеет платформа «Некстби», так как для выполнения задачи пользователю достаточно сделать минимальное количество шагов. В производительности уступает платформа «Azure IoT Central», так как приложение медленно работает с большими объёмами данных. Функциональные возможности платформ имеют одинаковую оценку: платформы обеспечивают возможность работы с устройствами и источниками данных, но не позволяют пользователю работать с активными правилами в графическом, списковом и языковом представлении. Выбранные платформы позволяют назначать пользователям различные права доступа в зависимости от их роли, но только платформа «Некстби» предоставляет возможность администратору редактировать права доступа.

В результате сравнительного анализа можно сделать вывод, что ни одно из рассмотренных приложений не имеет лучшую оценку по каждому критерию.

В результате сравнительного анализа также были определены требования к пользовательскому интерфейсу приложения управления метаданными:

- интерфейс должен быть удобным и интуитивно понятным для пользователя;

- приложение должно быстро реагировать на действия пользователя и без задержки загружать страницы;
- приложение должно иметь инструмент управления устройствами;
- пользователь должен иметь возможность работы с активными правилами в графическом, языковом и списочном представлении;
- приложение должно иметь возможность назначать и редактировать роли.

Разработка программного продукта проводилась с учётом выявленных требований.

1.4 Планирование процесса разработки приложения

1.4.1 Цели разработки и назначение приложения

Пользовательское web-приложение управления метаданными в составе информационно-управляющей платформы должно обеспечить пользователя удобным и интуитивно понятным интерфейсом. Приложение должно иметь все необходимые инструменты для работы с метаданными информационно-управляющей системы.

Цели разработки приложения следующие:

- создание удобного интерфейса пользовательского web-приложения;
- разграничения доступа к интерфейсам в зависимости от роли пользователя в системе;
- обеспечение авторизации и аутентификации пользователей;
- обеспечение взаимодействия пользователя с системами, их подсистемами и объектами;
- обеспечение взаимодействия пользователя с активными правилами в графическом, языковом и списочном представлении;
- обеспечение управления метаданными пользователем.

Пользовательское web-приложение предназначено для комплексного управления метаданными, что требует доступного и понятного представления

информации пользователям. Разделение пользователей по ролям предотвращает ошибочные изменения за счёт ограничения доступа к интерфейсам и данным. Обеспечение взаимодействия пользователя с активными правилами в разных представлениях значительно улучшает восприятие информации и облегчает процесс создания и редактирования правил.

1.4.2 Классы пользователей приложения

Приложение включает в себя следующие роли: системный аналитик, разработчик, главный администратор, бизнес-аналитик и администратор. Каждая роль имеет собственный функционал и ограничения.

На рисунке 13 представлена диаграмма вариантов использования, показывающая права доступа пользователя в зависимости от его роли в системе.

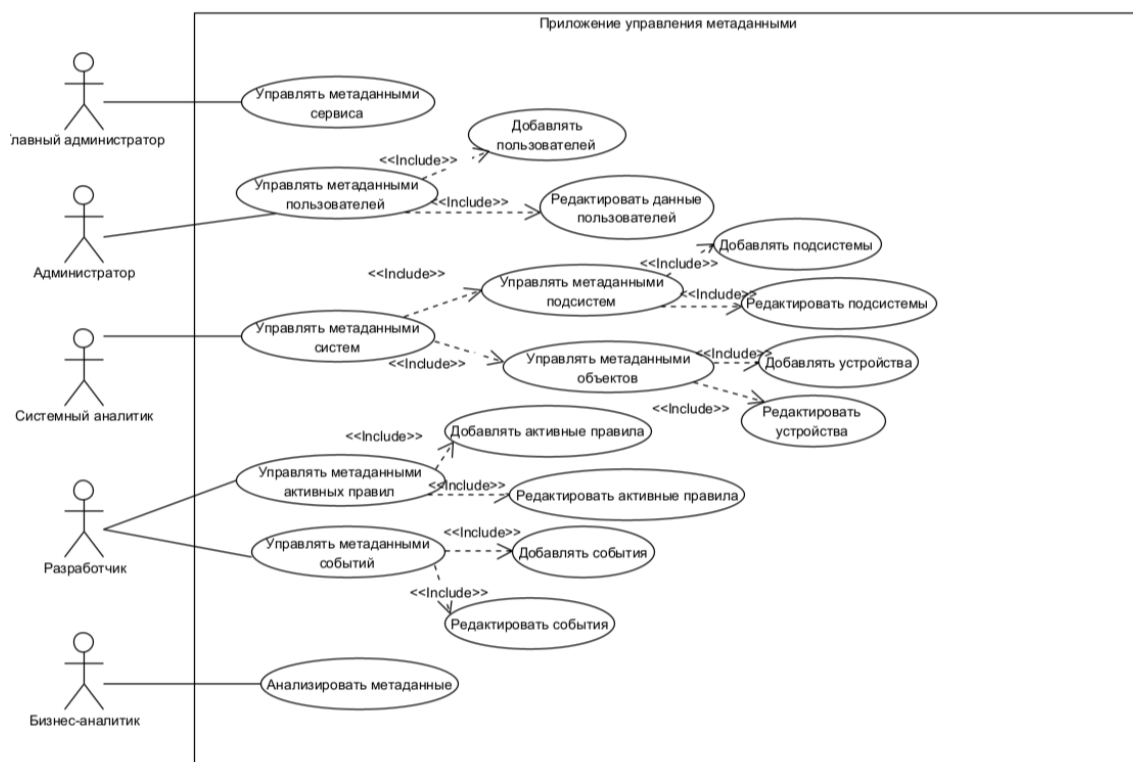


Рисунок 13 – Права доступа ролей

Разработчик управляет метаданными событий и активных правил, то есть создает и редактирует их метаданные.

В задачи системного аналитика входит управление метаданными систем. Пользователи, имеющие данную роль, создают и описывают системы и подсистемы, определяют объекты наблюдения и управления.

Бизнес-аналитик получает и анализирует информацию о произошедших событиях и активированных активных правилах. На основе полученной информации выявляет полученные ошибки.

Администратор управляет параметрами приложения, добавляет новых пользователей. У администратора также есть возможность изменять функционал роли в системе.

Главный администратор имеет функционал всех ролей. Пользователь, обладающий такой ролью, может управлять метаданными всего приложения.

1.4.3 План-график разработки и планирование её бюджета

План – это документ, описывающий подробно все этапы и действия, необходимые для завершения разработки проекта.

План разработки включает в себя следующие этапы: определение задач, анализ требований, назначение ресурсов, составление бюджета, определение сроков выполнения. Грамотное планирование обеспечивает не только мониторинг процесса разработки, но и удобное взаимодействие участников проекта, структурированный подход к разработке, минимизацию рисков.

Для успешной реализации проекта было выполнено планирование разработки и её бюджета. В таблице 8 представлены задачи проекта, определена последовательность и продолжительность их выполнения. Была выбрана пятидневная рабочая неделя.

Таблица 8 – Список задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ предметной области и планирование процесса разработки	-	-
3	Определение назначения и основных возможностей платформы	1 день	1
4	Анализ архитектуры платформы	1 день	3

Продолжение таблицы 8

№	Название задачи	Длительность	Предшественники
5	Анализ концептуальной модели	1 день	4
6	Анализ объектной модели метаданных	2 дня	5
7	Анализ жизненного цикла метаданных	1 день	6
8	Анализ бизнес-процессов управления	1 день	7
9	Сравнительный анализ приложения аналогичного назначения	2 дня	8
10	Планирование процесса разработки приложения	2 дня	9
11	Анализ предметной области и планирование процесса разработки завершены	-	10
12	Анализ требований и постановка задачи на разработку приложения	-	2
13	Определение функциональных требований	2 дня	11
14	Определение нефункциональных требований	1 день	13
15	Определение требований к пользовательскому интерфейсу	2 дня	14
16	Анализ требований к аппаратно-программному обеспечению	2 дня	15
17	Постановка задачи на разработку	1 день	16
18	Анализ требований и постановка задачи на разработку приложения завершены	-	17
19	Разработка приложения	-	12
20	Разработка архитектуры развертывания приложения	2 дня	18
21	Разработка пользовательского интерфейса	3 дня	20
22	Разработка программного обеспечения приложения	4 дня	21
23	Разработка приложения завершена	-	22
24	Оценка качества приложения	-	19
25	Планирование процесса контроля качества	1 день	23
26	Тестирование	2 дня	25
27	Оценка качества программного кода	1 день	25
28	Оценка качества пользовательского интерфейса	1 день	25
29	Оценка качества приложения завершена	-	26;27;28
30	Завершение работы над проектом	-	24
31	Оформление документации	3 дня	29
32	Сдача проекта	2 дня	31
33	Работа над проектом завершена	-	32
34	Конец проекта	-	33

По таблице 8 видно, что необходимо выполнить 34 задачи для выполнения проекта. В таблице 9 представлены ресурсы проекта, определены стандартные и сверхурочные ставки.

Таблица 9 – Список ресурсов проекта

№	Название ресурса	Тип ресурса	Стандартная ставка	Сверхурочная ставка
1	Аналитик	Трудовой	300 руб/час	350 руб/час

Продолжение таблицы 9

№	Название ресурса	Тип ресурса	Стандартная ставка	Сверхурочная ставка
2	Программист	Трудовой	350 руб/час	400 руб/час
3	Дизайнер	Трудовой	250 руб/час	300 руб/час
4	Тестировщик	Трудовой	250 руб/час	300 руб/час
5	Компьютер	Материальный	30 000 руб	-
6	Электричество	Затраты	-	-
7	Интернет	Затраты	-	-

По таблице 9 можно заметить, что в реализации проекта задействовано 7 ресурсов. В таблице 10 описано назначение каждого ресурса для проекта.

Таблица 10 – Назначение ресурсов

№	Название задачи	Название ресурса	Единицы
1	Начало проекта	-	-
2	Анализ предметной области и планирование процесса разработки	-	-
3	Определение назначения и основных возможностей платформы	Аналитик	100%
		Электричество	20 рублей
		Интернет	30 рублей
4	Анализ архитектуры платформы	Аналитик	100%
		Электричество	20 рублей
		Интернет	30 рублей
5	Анализ концептуальной модели	Аналитик	100%
		Электричество	20 рублей
6	Анализ объектной модели метаданных	Аналитик	100%
		Электричество	40 рублей
7	Анализ жизненного цикла метаданных	Аналитик	100%
		Электричество	20 рублей
8	Анализ бизнес-процессов управления	Аналитик	100%
		Электричество	20 рублей
9	Сравнительный анализ приложения аналогичного назначения	Аналитик	100%
		Электричество	40 рублей
		Интернет	60 рублей
10	Планирование процесса разработки приложения	Аналитик	100%
		Электричество	40 рублей
11	Анализ предметной области и планирование процесса разработки завершены	-	-
12	Анализ требований и постановка задачи на разработку приложения	-	-
13	Определение функциональных требований	Программист	100%
		Электричество	40 рублей
14	Определение нефункциональных требований	Программист	100%
		Электричество	20 рублей
15	Определение требований к пользовательскому интерфейсу	Аналитик	100%
		Электричество	40 рублей
16	Анализ требований к аппаратно-программному обеспечению	Программист	100%
		Электричество	40 рублей

Продолжение таблицы 10

№	Название задачи	Название ресурса	Единицы
17	Постановка задачи на разработку	Аналитик	100%
		Электричество	20 рублей
18	Анализ требований и постановка задачи на разработку приложения завершены	-	-
19	Разработка приложения	-	-
20	Разработка архитектуры развертывания приложения	Программист	100%
		Интернет	60 рублей
		Электричество	40 рублей
21	Разработка пользовательского интерфейса	Дизайнер	100%
		Интернет	90 рублей
		Электричество	60 рублей
22	Разработка программного обеспечения приложения	Программист	100%
		Интернет	120 рублей
		Электричество	80 рублей
23	Разработка приложения завершена	-	-
24	Оценка качества приложения	-	-
25	Планирование процесса контроля качества	Тестировщик	100%
		Электричество	20 рублей
26	Тестирование	Тестировщик	100%
		Интернет	60 рублей
		Электричество	40 рублей
27	Оценка качества программного кода	Тестировщик	100%
		Интернет	30 рублей
		Электричество	20 рублей
28	Оценка качества пользовательского интерфейса	Тестировщик	100%
		Дизайнер	100%
		Интернет	30 рублей
		Электричество	20 рублей
29	Оценка качества приложения завершена	-	-
30	Завершение работы над проектом	-	-
31	Оформление документации	Аналитик	100%
		Электричество	60 рублей
32	Сдача проекта	Программист	100%
		Аналитик	100%
		Интернет	60 рублей
		Электричество	40 рублей
33	Работа над проектом завершена	-	-
34	Конец проекта	-	-

Чтобы спланировать, проконтролировать и завершить проект в установленный срок, использовался инструмент Microsoft Project. Программное обеспечение обладает следующими возможностями:

- планирование проектов (определение задач, сроков и ресурсов);
- управление ресурсами (людьми, материалами, оборудованием);

- создание диаграммы Ганта (визуальное отображение плана проекта);
- создание отчётов (анализ общих затрат и использования ресурсов).

Для демонстрации загрузки каждого ресурса был создан отчет об использовании ресурсов, представленный на рисунке 14.

Использование ресурсов от Пн 12.05.25
ms_project

	31.03.25	07.04.25	14.04.25	21.04.25	28.04.25	05.05.25	12.05.25	19.05.25	26.05.25	02.06.25	Итого
Аналитик	24 ч	40 ч	24 ч	16 ч	8 ч			32 ч	8 ч		152 ч
Определение назначения и основных возможностей платформы	8 ч										8 ч
Анализ архитектуры платформы	8 ч										8 ч
Анализ концептуальной модели	8 ч										8 ч
Анализ объектной модели метаданных		16 ч									16 ч
Анализ жизненного цикла метаданных		8 ч									8 ч
Анализ бизнес-процессов управления		8 ч									8 ч
Сравнительный анализ приложения аналогичного назначения		8 ч									16 ч
Планирование процесса разработки приложения			8 ч								16 ч
Определение требований к пользовательскому интерфейсу			16 ч								16 ч
Постановка задачи на разработку				16 ч	8 ч						8 ч
Оформление документации								24 ч			24 ч
Сдача проекта								8 ч	8 ч		16 ч
Программист			16 ч	24 ч	16 ч	8 ч	24 ч	8 ч	8 ч		104 ч
Определение функциональных требований			16 ч								16 ч
Определение нефункциональных требований				8 ч							8 ч
Анализ требований к аппаратно-программному обеспечению				16 ч							16 ч
Разработка архитектуры развертывания приложения					16 ч						16 ч
Разработка программного обеспечения приложения						8 ч	24 ч				32 ч
Сдача проекта								8 ч	8 ч		16 ч
Дизайнер					16 ч	32 ч	8 ч				56 ч
Разработка пользовательского интерфейса					16 ч	32 ч					48 ч
Оценка качества пользовательского интерфейса							8 ч				8 ч
Тестировщик							32 ч	8 ч			40 ч
Планирование процесса контроля качества							8 ч				8 ч
Тестирование							8 ч	8 ч			16 ч
Оценка качества программного кода							8 ч				8 ч
Оценка качества пользовательского интерфейса							8 ч				8 ч
Компьютер											
Электричество											
Определение назначения и основных возможностей платформы											
Анализ архитектуры платформы											
Анализ концептуальной модели											
Анализ объектной модели метаданных											
Анализ жизненного цикла метаданных											
Анализ бизнес-процессов управления											
Сравнительный анализ приложения аналогичного назначения											
Планирование процесса разработки приложения											
Определение функциональных требований											
Определение нефункциональных требований											
Определение требований к пользовательскому интерфейсу											
Анализ требований к аппаратно-программному обеспечению											
Постановка задачи на разработку											
Разработка архитектуры развертывания приложения											
Разработка пользовательского интерфейса											
Разработка программного обеспечения приложения											
Планирование процесса контроля качества											
Тестирование											
Оценка качества программного кода											
Оценка качества пользовательского интерфейса											
Оформление документации											
Сдача проекта											
Интернет											
Определение назначения и основных возможностей платформы											
Анализ архитектуры платформы											
Сравнительный анализ приложения аналогичного назначения											
Разработка архитектуры развертывания приложения											
Разработка пользовательского интерфейса											
Разработка программного обеспечения приложения											
Тестирование											
Оценка качества программного кода											
Оценка качества пользовательского интерфейса											
Сдача проекта											
Итого	24 ч	40 ч	40 ч	40 ч	40 ч	40 ч	64 ч	48 ч	16 ч		352 ч

Рисунок 14 – Отчёт об использовании ресурсов

С помощью Microsoft Project была создана диаграмма Ганта, представленная на рисунке 15.

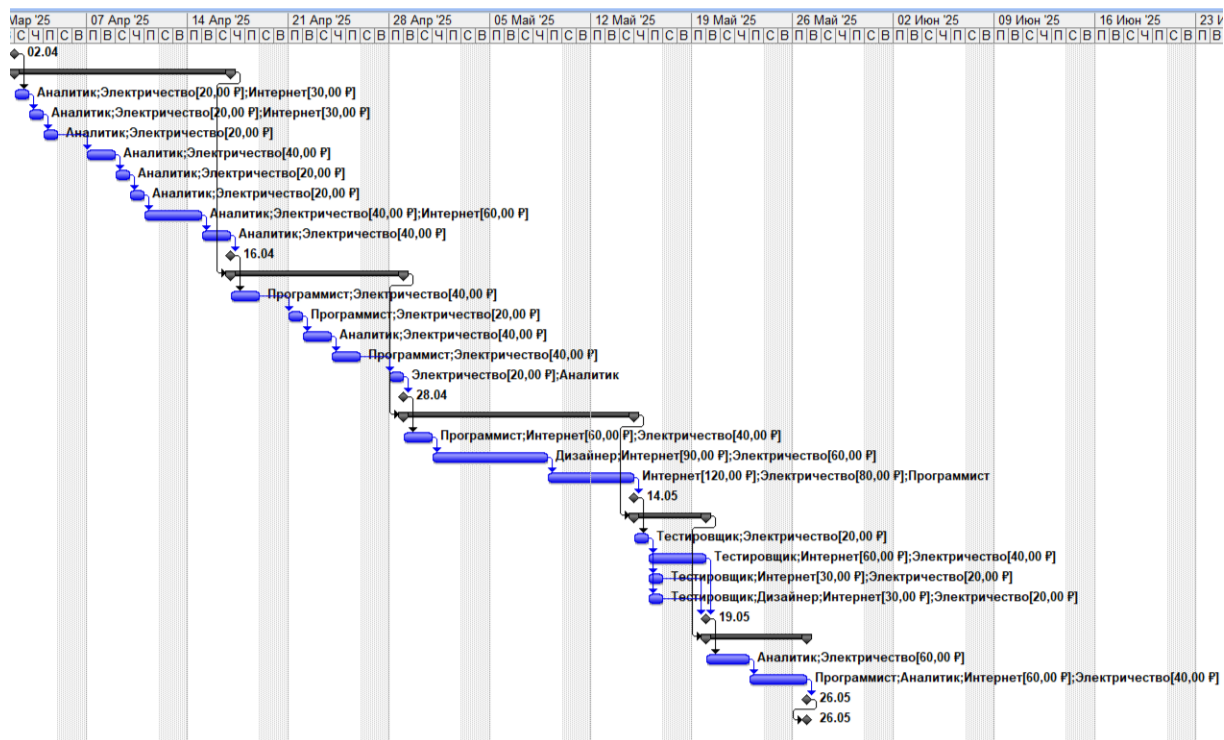


Рисунок 15 – План проекта

На диаграмме Ганта можно увидеть задачи проекта и их продолжительность, даты начала и окончания проекта, необходимые ресурсы.

Для демонстрации общих затрат был создан отчёт о бюджете, представленный на рисунке 16.

Отчет о бюджете от Пн 12.05.25
ms_project

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
21	Разработка пользовательского интв:	0,00 P	Пропорциональное	12 150,00 P	0,00 P	12 150,00 P
22	Разработка программного обеспече:	0,00 P	Пропорциональное	11 400,00 P	0,00 P	11 400,00 P
32	Сдача проекта	0,00 P	Пропорциональное	10 500,00 P	0,00 P	10 500,00 P
31	Оформление документации	0,00 P	Пропорциональное	7 260,00 P	0,00 P	7 260,00 P
20	Разработка архитектуры развертыва:	0,00 P	Пропорциональное	5 700,00 P	0,00 P	5 700,00 P
13	Определение функциональных треб:	0,00 P	Пропорциональное	5 640,00 P	0,00 P	5 640,00 P
16	Анализ требований к аппаратно-про:	0,00 P	Пропорциональное	5 640,00 P	0,00 P	5 640,00 P
9	Сравнительный анализ приложения	0,00 P	Пропорциональное	4 900,00 P	0,00 P	4 900,00 P
6	Анализ объектной модели метаданн:	0,00 P	Пропорциональное	4 840,00 P	0,00 P	4 840,00 P
10	Планирование процесса разработки	0,00 P	Пропорциональное	4 840,00 P	0,00 P	4 840,00 P
15	Определение требований к пользова:	0,00 P	Пропорциональное	4 840,00 P	0,00 P	4 840,00 P
26	Тестирование	0,00 P	Пропорциональное	4 100,00 P	0,00 P	4 100,00 P
28	Оценка качества пользовательского	0,00 P	Пропорциональное	4 050,00 P	0,00 P	4 050,00 P
14	Определение нефункциональных тр	0,00 P	Пропорциональное	2 820,00 P	0,00 P	2 820,00 P
3	Определение назначения и основны	0,00 P	Пропорциональное	2 450,00 P	0,00 P	2 450,00 P
4	Анализ архитектуры платформы	0,00 P	Пропорциональное	2 450,00 P	0,00 P	2 450,00 P
5	Анализ концептуальной модели	0,00 P	Пропорциональное	2 420,00 P	0,00 P	2 420,00 P
7	Анализ жизненного цикла метаданн:	0,00 P	Пропорциональное	2 420,00 P	0,00 P	2 420,00 P
8	Анализ бизнес-процессов управлен:	0,00 P	Пропорциональное	2 420,00 P	0,00 P	2 420,00 P
17	Постановка задачи на разработку	0,00 P	Пропорциональное	2 420,00 P	0,00 P	2 420,00 P
27	Оценка качества программного кода	0,00 P	Пропорциональное	2 050,00 P	0,00 P	2 050,00 P
25	Планирование процесса контроля ка	0,00 P	Пропорциональное	2 020,00 P	0,00 P	2 020,00 P
1	Начало проекта	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
11	Анализ предметной области и плани	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
18	Анализ требований и постановка за:	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
23	Разработка приложения завершена	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
29	Оценка качества приложения завер:	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
33	Работа над проектом завершена	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
34	Конец проекта	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
		0,00 P		107 330,00 P	0,00 P	107 330,00 P

Рисунок 16 – Отчёт о бюджете

Анализируя отчеты, можно заметить, что для реализации проекта потребуется 107330 рублей и 352 часа работы. Дата начала проекта: 2 апреля. Дата окончания проекта: 26 мая.

2 Анализ требований и постановка задачи на разработку пользовательского web-приложения управления метаданными

2.1 Анализ функциональных требований

Пользовательское приложение управления метаданными предполагает разделение прав доступа пользователей в зависимости от их роли. В рамках приложения действуют следующие роли:

- главный администратор – отвечает за управление системами, обладает функционалом всех ролей;
- администратор системы – отвечает за управление пользователями и их ролями;
- разработчик – управляет активными правилами и событиями;
- системный аналитик – управляет подсистемами, объектами наблюдения и управления;
- бизнес-аналитик – анализирует произошедшие события и активированные активные правила.

Для обеспечения управления метаданными пользователями разных ролей пользовательское приложение должно обладать следующими функциями:

- регистрация пользователя;
- авторизация зарегистрированного пользователя;
- отображение метаданных систем, подсистем и объектов наблюдения и управления, событий;
- редактирование метаданных систем, подсистем и объектов наблюдения и управления, событий;
- просмотр активных правил в списочном, графическом и языковом представлении;
- редактирование активных правил в списочном, графическом и языковом представлении;

– удаление систем, подсистем и объектов наблюдения и управления, событий, активных правил.

В результате описания основных требований к пользовательскому приложению была составлена диаграмма вариантов использования, представленная на рисунке 17.

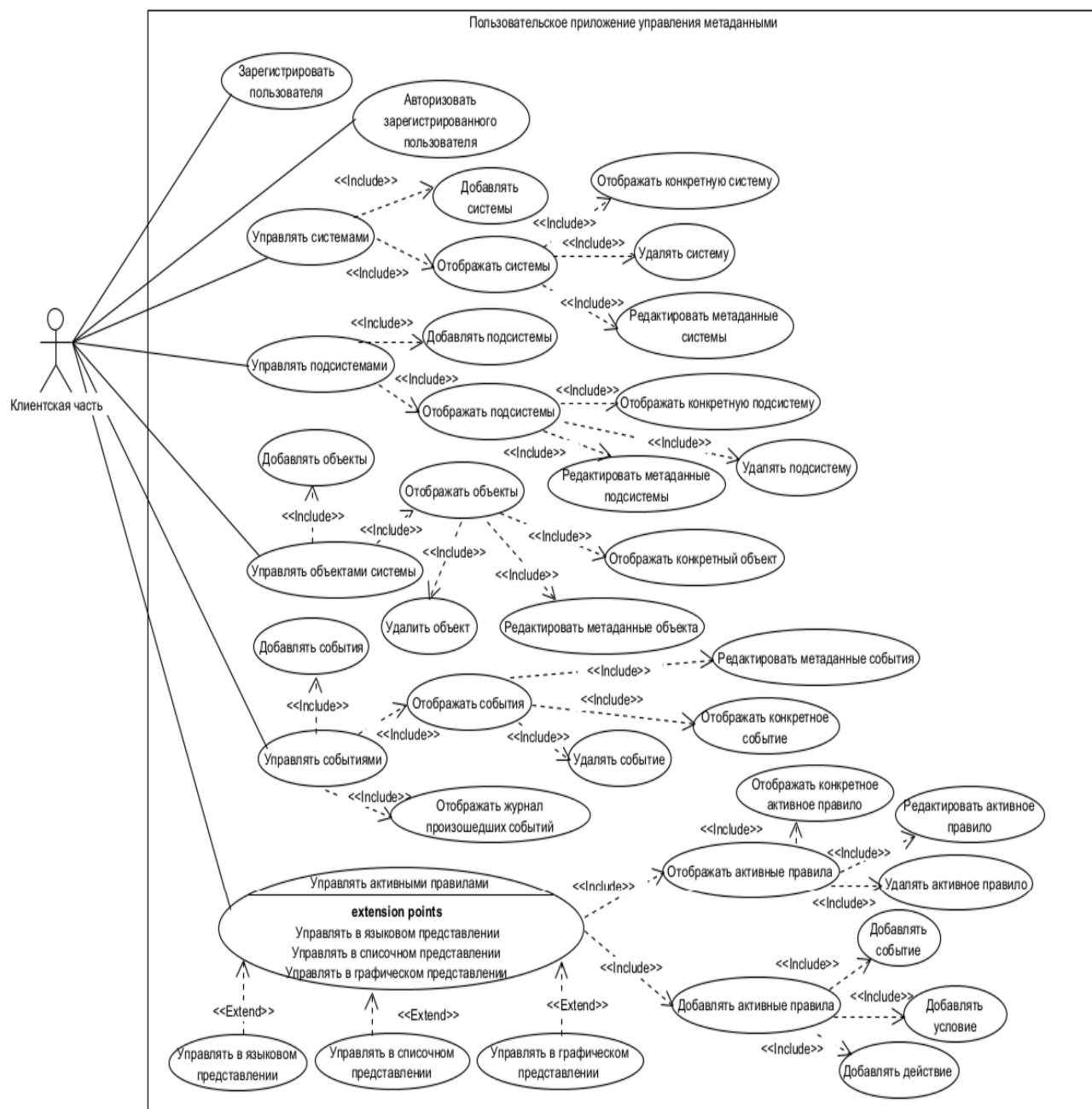


Рисунок 17 – Функции пользовательского приложения

На рисунках 18 – 20 представлены сценарии некоторых вариантов использования.

Добавлять активные правила
Id: 1
Описание: Приложение позволяет пользователю добавлять активное правило
Действующие лица: Пользователь, приложение
Предусловия: Пользователь авторизовался в приложении, в системе имеет роль разработчика
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает вид представления 2. Приложение отображает конструктор добавления активного правила в выбранном представлении 3. Пользователь добавляет событие 4. Пользователь добавляет условие 5. Пользователь добавляет действие 6. Пользователь нажимает кнопку «Добавить» 7. Приложение отображает добавленное активное правило
Постусловие: Пользователь добавил активное правило

Рисунок 18 – Сценарий варианта использования «Добавлять активные правила»

Редактировать метаданные системы
Id: 2
Описание: Приложение позволяет пользователю редактировать метаданные системы
Действующие лица: Пользователь, приложение
Предусловия: Пользователь авторизовался в приложении, имеет роль главного администратора
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает конкретную систему 2. Приложение отображает метаданные системы 3. Пользователь изменяет метаданные системы: название, адрес, администратора системы, описание. 4. Пользователь нажимает кнопку «Сохранить изменения» 5. Приложение отображает измененную систему
Постусловие: Пользователь отредактировал метаданные системы

Рисунок 19 – Сценарий варианта использования «Редактировать метаданные системы»

Авторизовать зарегистрированного пользователя
Id: 3
Описание: Приложение позволяет пользователю авторизоваться
Действующие лица: Пользователь, приложение
Предусловия: Пользователь зарегистрирован в приложении
Основной поток: <ol style="list-style-type: none"> 1. Пользователь указывает логин 2. Пользователь указывает пароль 3. Пользователь нажимает кнопку «Войти» 4. Приложение выполняет вход пользователя
Постусловие: Пользователь авторизовался в приложении

Рисунок 20 – Сценарий варианта использования «Авторизовать зарегистрированного пользователя»

Пользовательское web-приложение управления метаданными было реализовано, учитывая вышеупомянутые требования.

2.2 Анализ нефункциональных требований

Нефункциональные требования – описание свойств или особенностей, которым должно обладать приложение, или ограничение, которое должно соблюдать приложение. Такие требования описывают, не что система делает, а как хорошо она это делает [7].

При анализе нефункциональных требований были рассмотрены требования к доступности, надежности, производительности, масштабируемости, безопасности и удобству использования (таблица 11).

Таблица 11 – Нефункциональные требования

Характеристика	Требования
Доступность	Приложение должно быть доступно 98% времени, за исключением времени планового обслуживания.
Надежность	Приложение должно обеспечивать время восстановления после сбоя не более 2 часов.
Производительность	Страницы должны загружаться не более чем за 3 секунды, обработка запросов должна продолжаться не более 1 секунды, задержка отклика должна быть не более 5 миллисекунд.
Масштабируемость	Приложение должно обеспечивать масштабируемость до 500 пользователей с сохранением текущего уровня производительности.
Безопасность	Допускается использование только учётных записей, доступ к функционалу зависит от роли пользователя.
Удобство использования	Интерфейс приложения должен быть интуитивно понятен пользователю и должен быть выполнен в спокойных цветовых тонах для концентрации внимания.

Пользовательское web-приложение управления метаданными должно обеспечивать выполнение вышеупомянутых нефункциональных требований.

2.3 Анализ требований к пользовательскому интерфейсу

Пользовательское web-приложение должно иметь удобный и интуитивно понятный интерфейс, который будет обеспечивать весь необходимый функционал для управления метаданными.

Для удобства работы с информационно-управляющей системой интерфейс приложения должен быть разделен на следующие страницы:

- страница регистрации;
- страница авторизации;
- страница профиля пользователя;
- страница систем;
- страница подсистем;
- страница объектов наблюдения и управления;
- страница активных правил в графическом, языковом и списочном представлении;
- страница событий.

На странице регистрации должна быть форма с полями ввода данных пользователя. На странице авторизации должны быть поля ввода идентификационных данных пользователя. На странице профиля пользователя должна быть возможность редактирования своих данных, просмотра систем, в которых присутствует пользователь, и его роли в них.

На странице систем должен располагаться список систем с краткой информацией о каждой из них. Страница должна предполагать, что пользователь может выбрать конкретную систему и посмотреть более подробную информацию о ней. Аналогично со страницей подсистем, объектов наблюдения и управления и событий.

На странице активных правил должна быть возможность изменять вид представления. При выборе графического представления должна быть возможность управления событиями, условиями и действиями с помощью графических элементов. При выборе языкового представления страница должна иметь текстовое поле для написания языковой конструкции. При выборе списочного представления параметры активного правила должны быть представлены в виде последовательного списка.

Пользовательский интерфейс должен учитывать роль пользователя в системе и в зависимости от этого показывать или скрывать компоненты интерфейса.

2.4 Анализ требований к аппаратно-программному обеспечению

2.4.1 Требования к аппаратному обеспечению

Требования к аппаратному обеспечению описывают, какое оборудование должно быть у пользователя для запуска и корректной работы программы.

В таблице 12 представлены основные требования к аппаратному обеспечению для запуска пользовательского web-приложения управления метаданными.

Таблица 12 – Требования к аппаратному обеспечению

Оборудование	Минимальные требования
Процессор	6 ядер с частотой 3 ГГц
Оперативная память	От 4 ГБ
Объем свободного места на жестком диске	От 10 ГБ

В результате анализа требования к аппаратному обеспечению можно сделать вывод, что приложение работает со стандартными конфигурациями и не требует больших ресурсов.

2.4.2 Требования к системному программному обеспечению

Требования к системному программному обеспечению определяют операционные системы, на которых возможен запуск приложения и его корректная работа.

Для развертывания приложения рекомендуемыми операционными системами являются следующие:

- Microsoft Windows (64-bit) 10 и выше;
- Apple OS X 10.7 и выше;
- Ubuntu 16.04 и выше.

Для использования приложения рекомендуемыми браузерами являются:

- Яндекс Браузер 18.2.1 и выше;

- Google Chrome 72.0.3626 и выше;
- Microsoft Edge 136.0.3240.50 и выше;
- Opera 36.0 и выше;
- Apple Safari 8.0 и выше;
- Mozilla Firefox 47.0 и выше.

Использование выше указанных браузеров и их версий обеспечивает точное отображение и корректную работу пользовательского web-приложения.

2.4.3 Требования к средствам разработки

В результате анализа функциональных и нефункциональных требований, требований к пользовательскому интерфейсу и аппаратно-программному обеспечению, были определены технологии и средства разработки, необходимые для реализации пользовательского web-приложения управления метаданными.

Для разработки пользовательского интерфейса использовалась JavaScript-библиотека React. React использует компонентную архитектуру с чистым JavaScript, декларативный стиль программирования и мощные, удобные для разработчика абстракции DOM. React обеспечивает высокую производительность благодаря своей виртуальной модели DOM и алгоритму интеллектуального согласования [8].

В качестве языков программирования использовались JavaScript и TypeScript. JavaScript – интерпретируемый язык программирования без строгой типизации. TypeScript – доработанная версия JavaScript со статической типизацией данных. Для выполнения JavaScript и TypeScript использовалась кроссплатформенная среда выполнения Node.js и менеджер пакетов для платформы npm.

При разработке приложения использовалась система верстки HTML, которая определяет, как и какие элементы должны располагаться на веб-странице, и язык стилей для оформления веб-страниц CSS [9]. Совместное

использование HTML и CSS позволяет разделить структуру и содержание страницы от визуального представления элементов.

Для визуализации структуры и наглядного представления процессов приложения использовался стандартизированный язык моделирования UML. Язык UML – средство визуализации моделей программных систем. UML использует графическую нотацию, то есть позволяет изображать устройство и работу приложения в виде рисунков-диаграмм, что сильно упрощает процесс создания и восприятия моделей приложения [10]. В качестве конструктора диаграмм на языке UML использовалась программа Visual Paradigm.

Для построения визуальных прототипов пользовательского интерфейса использовался графический редактор Figma. Основные преимущества Figma заключаются в возможности создания векторных иллюстраций, в поддержке компонентов и стилей, в создании адаптивных макетов.

Для развертывания и управления приложением использовалось программное обеспечение Docker, которое позволяет упаковать приложение со всеми необходимыми зависимостями в единый контейнер.

Помимо выше перечисленных средств разработки использовались следующие библиотеки:

- react-router-dom – библиотека для управления маршрутизацией;
- axios – библиотека для выполнения HTTP-запросов;
- ts-node – библиотека для выполнения TypeScript в Node.js без предварительной компиляции;
- chart.js – библиотека для создания графиков и диаграмм;
- express – веб-фреймворк, предоставляющий множество функций для создания веб-приложений и API.

Разработка пользовательского web-приложения управления метаданными в составе информационно-управляющей платформы проводилась в операционной системе Windows 10 с использованием Яндекс Браузера версии 25.2.4.1024 и редактора кода Visual Studio Code.

2.5 Постановка задачи на разработку приложения

Разрабатываемое пользовательское web-приложение предназначено для управления метаданными информационно-управляющей платформы. Основная цель разрабатываемого приложения – обеспечить взаимодействие пользователя с системами и подсистемами, объектами управления и наблюдения, событиями и активными правилами.

Основные требования к пользовательскому приложению:

- приложение должно иметь удобный и понятный пользовательский интерфейс;
- приложение должно обеспечивать возможность просмотра, создания, редактирования систем;
- приложение должно обеспечивать возможность просмотра, создания, редактирования подсистем и объектов наблюдения и управления;
- приложение должно обеспечивать возможность просмотра, создания, редактирования событий;
- приложение должно обеспечивать возможность просмотра, создания, редактирования активных правил в графическом, языковом, списочном представлении;
- приложение должно иметь разный функционал и пользовательский интерфейс для каждой роли;
- приложение должно обеспечивать регистрацию и авторизацию пользователей.

Пользовательское web-приложение должно обладать высокими значениями таких характеристик, как производительность, доступность, надежность, производительность, масштабируемость и безопасность.

3 Разработка пользовательского web-приложения управления метаданными

3.1 Разработка архитектуры развертывания приложения

В результате анализа требований и постановки задачи к разработке программного обеспечения была определена архитектура пользовательского web-приложения управления метаданными. Диаграмма компонентов приложения представлена на рисунке 21.

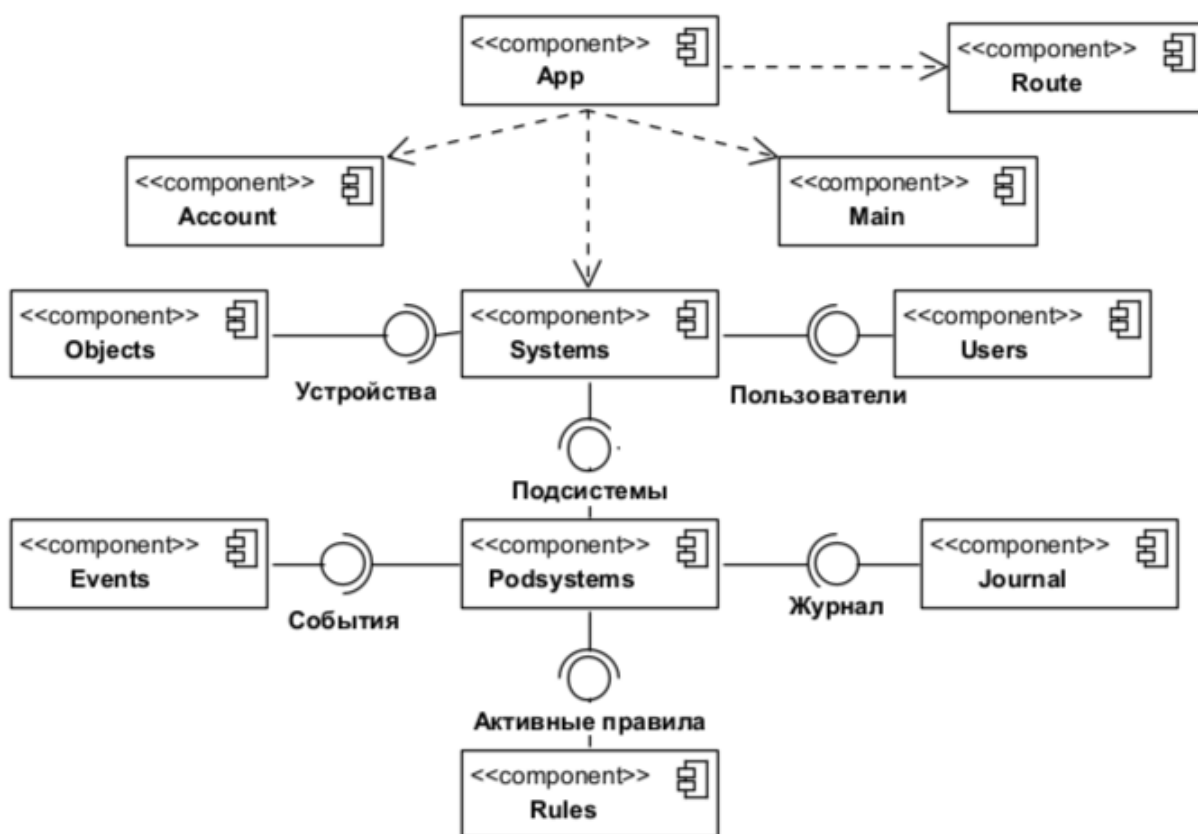


Рисунок 21 – Компоненты приложения

Диаграмма компонентов состоит из компонентов, зависимостей между компонентами и интерфейсов.

Компонент – логическая замещаемая часть системы, которая соответствует некоторому набору интерфейсов и обеспечивает их реализацию.

Интерфейс – набор операций, который специфицируют сервис, предоставляемый либо требуемый компонентом [11].

Описание назначений компонентов приложения представлено в таблице 12.

Таблица 12 – Описание назначений компонентов

Название компонента	Назначение компонента
App	Является точкой входа приложения и содержит зависимости всех остальных компонентов.
Main	Отвечает за взаимодействие пользователя с главной страницей приложения.
Account	Предоставляет отображение и редактирование данных вошедшего пользователя.
Systems	Отвечает за взаимодействие пользователя с метаданными доступных систем.
Objects	Отвечает за взаимодействие пользователя с метаданными устройств выбранной системы.
Users	Отвечает за взаимодействие пользователя с метаданными пользователей выбранной системы.
Podsystems	Отвечает за взаимодействие пользователя с метаданными подсистем выбранной системы.
Events	Отвечает за взаимодействие пользователя с метаданными событий выбранной подсистемы.
Rules	Отвечает за взаимодействие пользователя с метаданными активных правил выбранной подсистемы.
Journal	Отвечает за взаимодействие пользователя с метаданными произошедших событий выбранной подсистемы.
Route	Обеспечивает маршрутизацию запросов пользователя серверу.

Взаимосвязанные компоненты были объединены в модули для визуализации общей архитектуры приложения с помощью построения диаграммы пакетов (рисунок 22).

Диаграмма пакетов состоит из пакетов и зависимостей между ними. Пакет является элементом модели, который используется для группировки других элементов модели. Элементы модели, которые входят в состав некоторого пакета, называются его подпакетами [12].

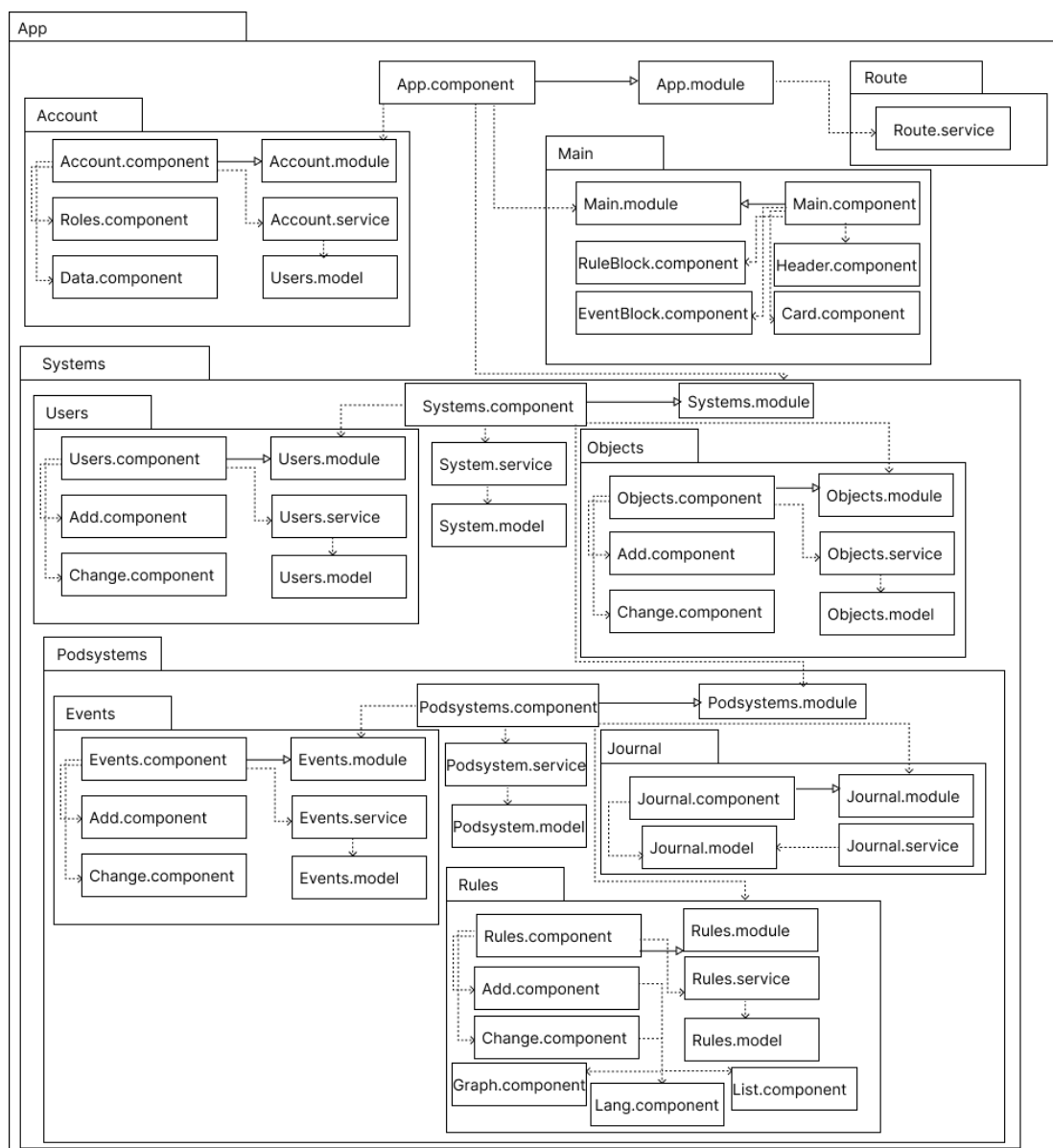


Рисунок 22 – Архитектура развертывания приложения

Описание сущностей диаграммы представлено в таблицах 13 – 16.

Таблица 13 – Описание модулей

Название сущности	Описание сущности
App.module	Точка входа приложения, содержит зависимости всех компонентов
Account.module	Модуль профиля текущего пользователя
Users.model	Модель данных пользователей
Main.module	Модуль взаимодействия с главной страницей
Systems.module	Модуль взаимодействия с системами
Objects.module	Модуль взаимодействия с устройствами выбранной системы
Podsystems.module	Модуль взаимодействия с подсистемами выбранной системы
Events.module	Модуль взаимодействия с событиями выбранной подсистемы
Journal.module	Модуль взаимодействия с журналом событий подсистемы
Rules.module	Модуль взаимодействия с активными правилами подсистемы

Таблица 14 – Описание компонентов

Название сущности	Описание сущности
App.component	Главный компонент приложения, предоставляющий web-страницу
Account.component	Интерфейс профиля пользователя
Roles.component	Список доступных систем и ролей пользователя
Data.component	Данные текущего пользователя
Main.component	Интерфейс главной страницы приложения
RuleBlock.component	Круговая диаграмма видов активных правил
EventBlock.component	Круговая диаграмма типов событий
Header.component	Шапка главной страницы
Card.component	Карточки количества компонентов приложения
Systems.component	Системы приложения
Users.component	Список пользователей выбранной системы
Add.component	Добавление системы, подсистемы, пользователя, подсистемы, устройства, события, активного правила
Change.component	Редактирование метаданных системы, подсистемы, пользователя, подсистемы, устройства, события, активного правила
Objects.component	Список устройств системы
Podsystems.component	Список подсистем выбранной системы
Events.component	Список событий подсистемы
Journal.component	Список активных правил подсистемы
Rules.component	Список активных правил подсистемы
Craph.component	Графическое отображение активного правила
Lang.component	Списочное отображение активного правила
List.component	Языковое отображение активного правила

Таблица 15 – Описание сервисов

Название сущности	Описание сущности
Systems.service	Сервис, обеспечивающий получение списка доступных систем, редактирование метаданных систем, добавление и удаление системы
Users.service	Сервис, обеспечивающий получение списка пользователей системы, редактирование метаданных, добавление и удаление пользователя
Account.service	Сервис, реализующий редактирование данных
Objects.service	Сервис, обеспечивающий получение списка устройств системы, редактирование метаданных устройств, добавление и удаление устройства
Podsystems.service	Сервис, обеспечивающий получение списка подсистем системы, редактирование метаданных подсистемы, их добавление и удаление
Events.service	Сервис, обеспечивающий получение списка событий подсистемы, редактирование метаданных событий, их добавление и удаление
Journal.service	Сервис, обеспечивающий получение списка произошедших событий
Rules.service	Сервис, обеспечивающий получение списка активных правил подсистемы, редактирование метаданных правил, их добавление и удаление

Таблица 16 – Описание моделей

Название сущности	Описание сущности
Rules.model	Модель данных активных правил
Systems.model	Модель данных систем

Продолжение таблицы 16

Название сущности	Описание сущности
Objects.model	Модель данных устройств
Podsystems.model	Модель данных подсистем
Events.model	Модель данных событий
Journal.model	Модель произошедших событий

Компоненты реализуют пользовательский интерфейс и функционал приложения, включают в себя файлы с расширениями `tsx` и `css`. Модули представляют собой логические единицы, которые также представлены на диаграмме компонентов. Сервисы обеспечивают передачу данных от клиента сервису и наоборот, включают в себя файлы с расширением `js`. Модели представляют собой локальное хранилище метаданных и содержат структуру метамодели, включают в себя файлы с расширениями `json` и `ts`.

3.2 Разработка пользовательского интерфейса приложения

3.2.1 Разработка сценариев взаимодействия пользователя с приложением

Пользовательское web-приложение управления метаданными в составе информационно-управляющей платформы должно обладать удобным и интуитивно понятным интерфейсом.

Разработка пользовательского интерфейса состоит из следующих этапов:

- определение необходимой функциональности системы;
- создание пользовательских сценариев;
- проектирование общей структуры;
- конструирование отдельных блоков;
- сбор полной схемы интерфейса.

В процессе анализа требования и постановки задачи был выполнен первый этап разработки пользовательского интерфейса, а именно были определены функции и возможности приложения управления метаданными.

Второй этап заключается в написании словесного описания взаимодействия пользователя с системой, не конкретизируя, как именно

проходит взаимодействие, но уделяя возможно большее внимание всем целям пользователей.

Третий этап заключается в выделении независимых блоков и определении связи между ними. В результате этого этапа были построены схемы функциональных блоков, представленные на рисунках 23 – 25.

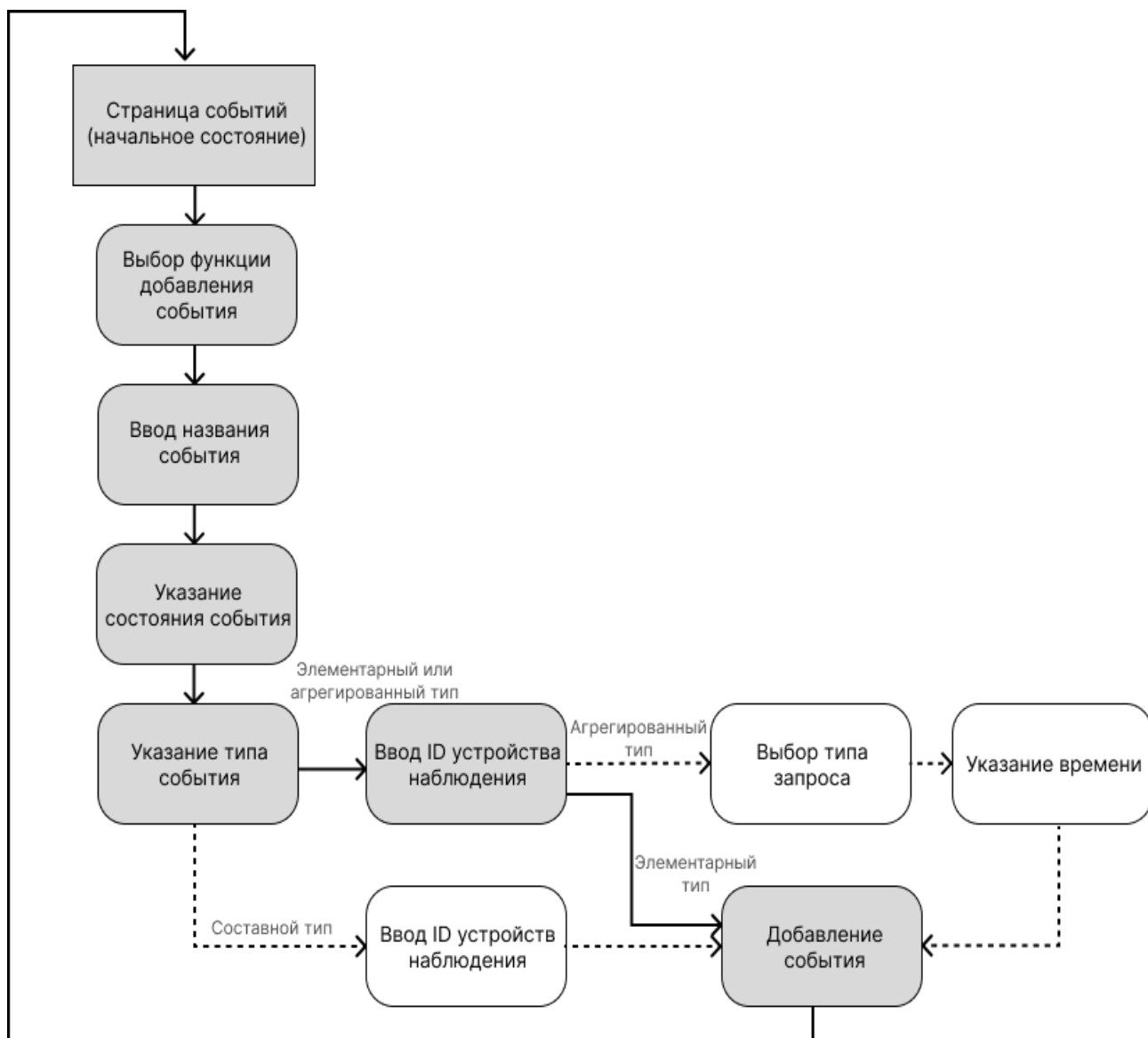


Рисунок 23 – Схема функциональных блоков при реализации функции
«Добавление события»

На схемах прямоугольник обозначает отдельный экран, прямоугольник со скругленными углами – область экрана, пунктирная линия – альтернативное действие [13].

Для добавления события пользователю необходимо заполнить поля метаданных события. С помощью схемы было определено, что поля для заполнения зависят от выбора типа события.

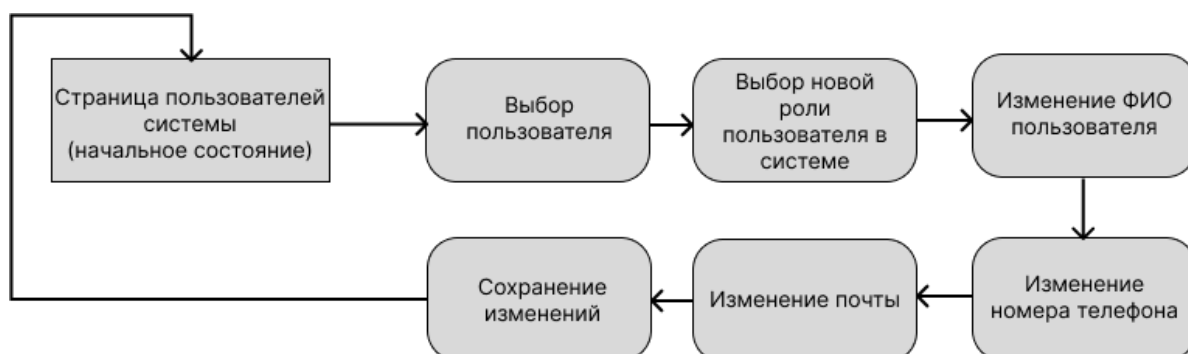


Рисунок 24 – Схема функциональных блоков при реализации функции «Редактирование данных пользователя»

Для редактирования данных пользователю необходимо выбрать из списка пользователей актуального и изменить его метаданные в полях интерфейса.

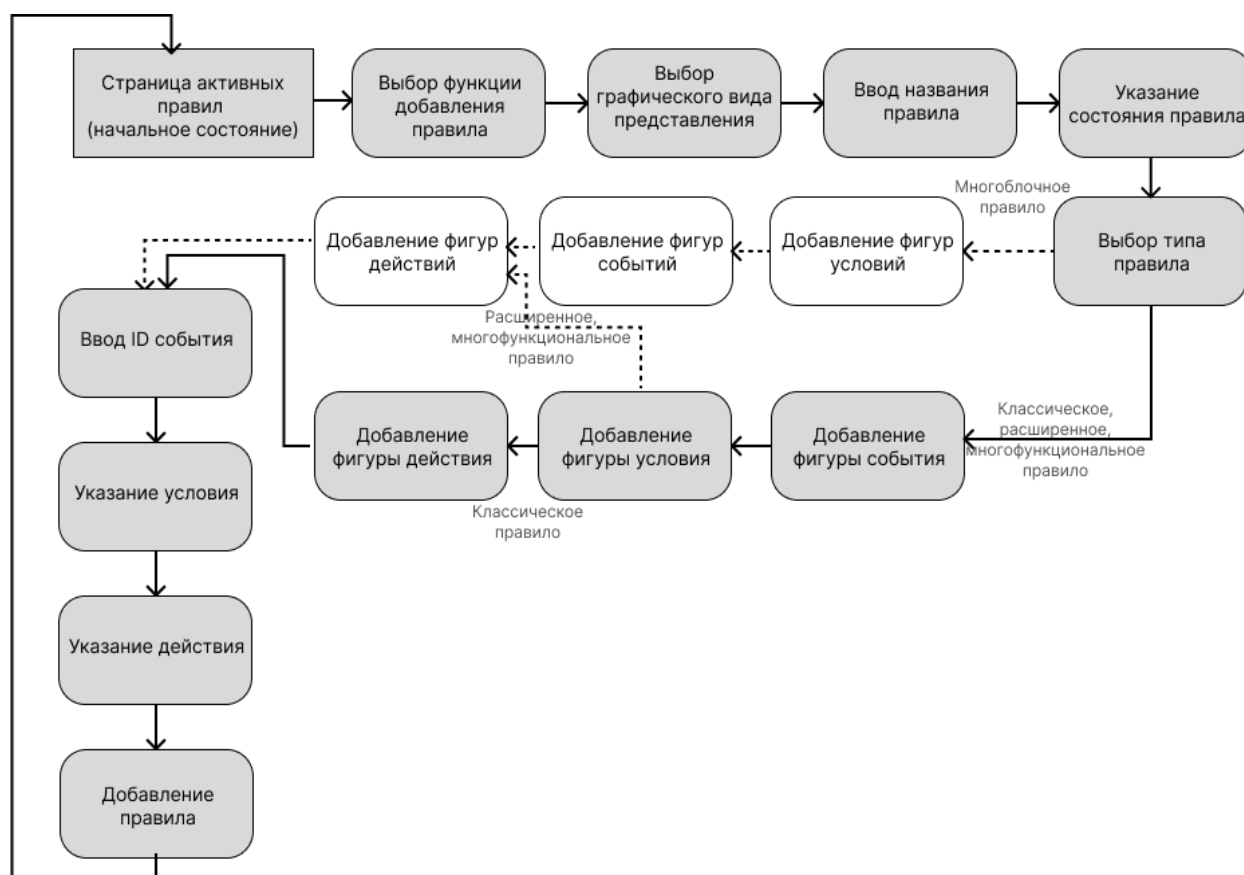


Рисунок 25 – Схема функциональных блоков при реализации функции «Добавление активного правила в графическом представлении»

Для добавления активного правила в графическом представлении пользователю необходимо заполнить основные данные о правиле, после чего выбрать его тип и добавить необходимые фигуры. Аналогично добавлению события, действия при добавлении правила зависят от его типа.

Построение схем функциональных блоков позволило детализировать функционал необходимых страниц.

3.2.2 Разработка визуальных прототипов пользовательского интерфейса

Визуальные прототипы пользовательского интерфейса были созданы на основе разработанных сценариев взаимодействия пользователя с приложением и сформированной архитектуры интерфейса. Прототип представляет собой визуальный шаблон пользовательского интерфейса.

Пользовательский интерфейс приложения управления метаданными в составе информационно-управляющей платформы содержит 11 страниц:

- страница авторизации;
- страница регистрации;
- главная страница;
- страница профиля пользователя;
- страница взаимодействия с метаданными систем;
- страница взаимодействия с метаданными подсистем;
- страница взаимодействия с метаданными устройств;
- страница взаимодействия с метаданными пользователей;
- страница взаимодействия с метаданными активных правил;
- страница взаимодействия с метаданными событий;
- страница журнала произошедших событий.

Визуальные прототипы основных страниц и окон интерфейса представлены на рисунках 26 – 32. Представленный пользовательский

интерфейс имеет пользователь с ролью «Главный администратор», которому доступен полный функционал приложения управления метаданными.

Active Rules service

1 Введите ФИО

2 Телефон Введите номер телефона

3 Email Введите email

4 Login Введите логин

5 Пароль Введите пароль

Зарегистрироваться

Отмена

Рисунок 26 – Страница регистрации

Основные элементы интерфейса страницы регистрации, представленной на рисунке 26:

- 1 – поля для ввода данных пользователя;
- 2 – заголовки полей ввода данных;
- 3 – кнопка регистрации пользователя;
- 4 – кнопка выхода со страницы регистрации.

После регистрации пользователю доступен функционал приложения для управления метаданными.



Рисунок 27 – Главная страница

Основные элементы интерфейса главной страницы, представленной на рисунке 27:

- 1 – поля для ввода данных пользователя;
- 2 – раскрывающийся список, предоставляющий возможность перехода на страницу профиля или выхода из текущего аккаунта;
- 3 – карточки, показывающие количество компонентов в приложении;
- 4 – карточки с круговыми диаграммами, показывающие статистику использования разных видов активных правил и событий.

С главной страницы пользователь имеет возможность перехода на страницу профиля, систем и настроек.

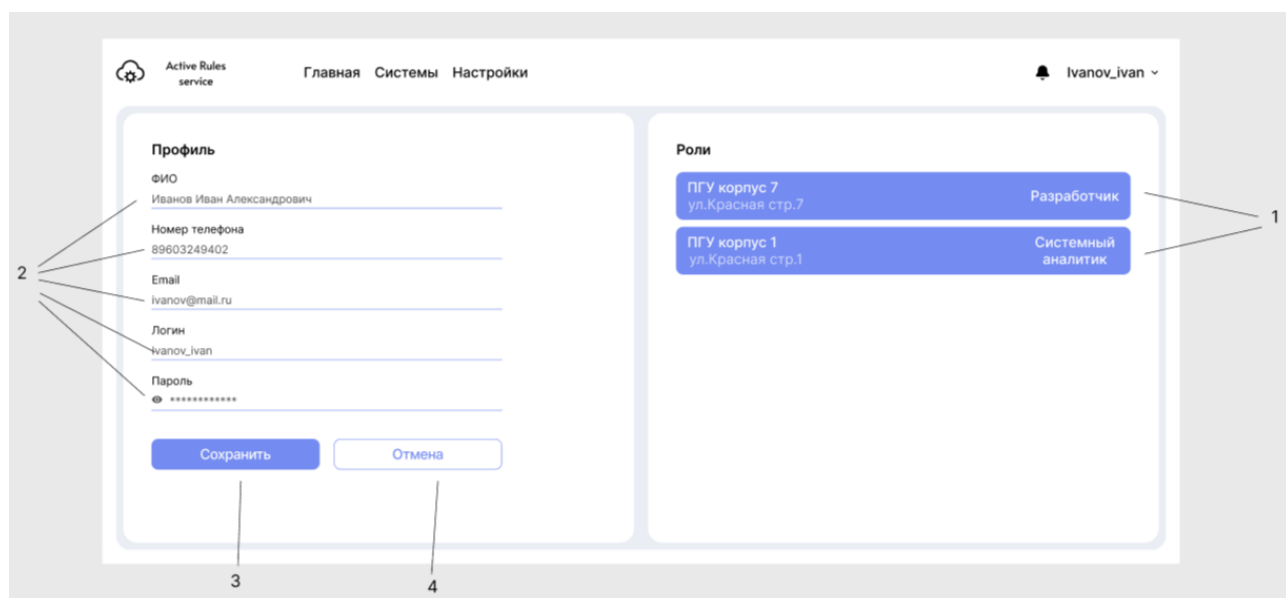


Рисунок 28 – Страница профиля пользователя

Основные элементы интерфейса страницы профиля пользователя, представленной на рисунке 28:

- 1 – блоки с информацией о ролях пользователя в доступных системах;
- 2 – поля метаданных пользователя;
- 3 – кнопка сохранения изменений при редактировании данных;
- 4 – кнопка сброса введенных изменений.

Страница профиля содержит два основных блока: блок с данными пользователя и блок с доступными ролями.

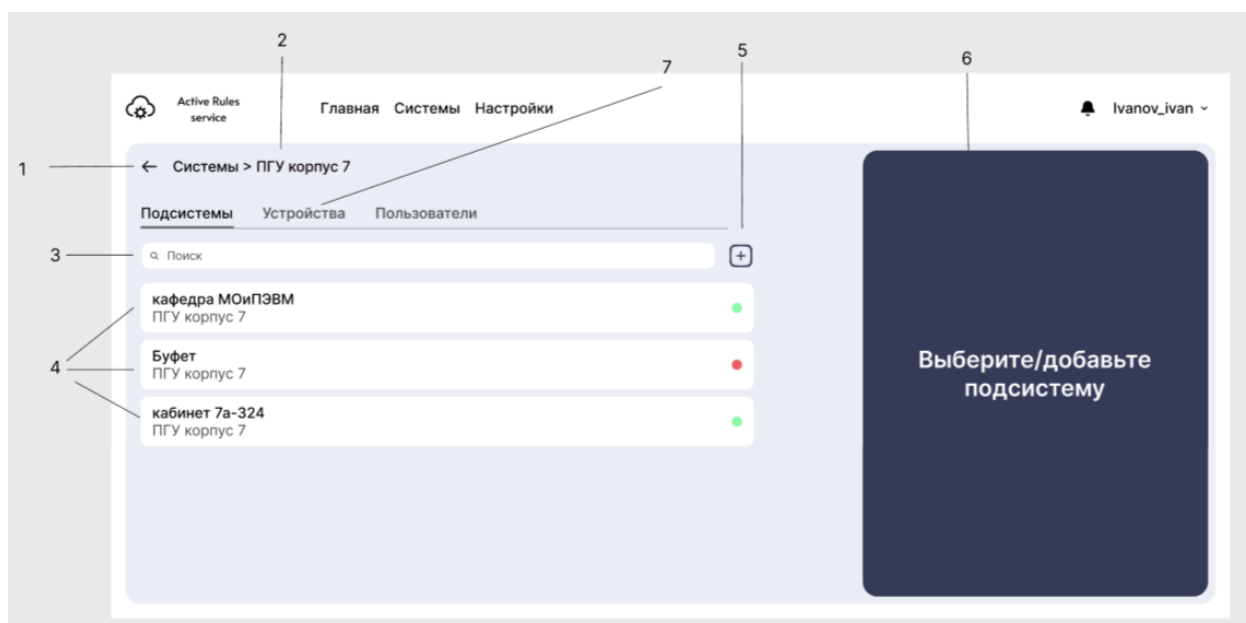


Рисунок 29 – Страница взаимодействия с метаданными подсистем

Основные элементы интерфейса страницы взаимодействия с метаданными подсистем, представленной на рисунке 29:

1 – иконка, при клике на которую происходит перенаправление на предыдущую страницу;

2 – информация о пути до текущей страницы;

3 – поле поиска подсистемы по названию;

4 – список подсистем текущей системы;

5 – кнопка добавления новой подсистемы;

6 – блок добавления/редактирования подсистемы;

7 – панель переходов между разделами системы.

Чтобы перейти на страницу подсистем, необходимо в шапке страницы выбрать «Системы», после чего в списке систем выбрать интересующую и нажать кнопку «Перейти в систему».

Со страницы подсистем возможен переход на страницу устройств и пользователей системы. При выборе интересующей подсистемы из списка подсистем и при нажатии на кнопку «Перейти в подсистему» пользователь попадает на страницу событий, активных правил и журнала произошедших событий.

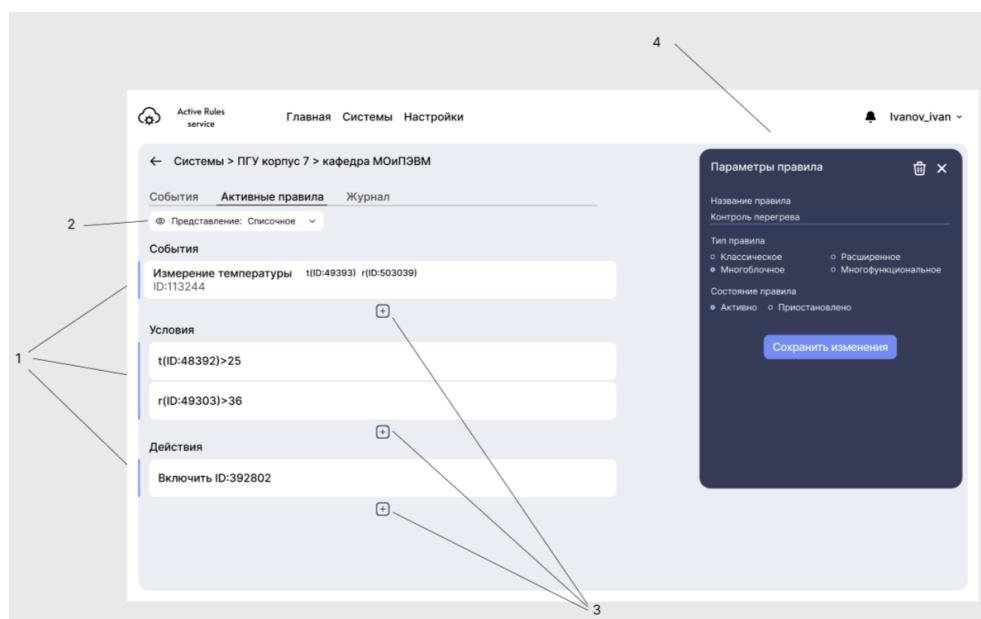


Рисунок 30 – Страница активного правила в списочном представлении

Основные элементы интерфейса страницы активного правила в списочном представлении, представленной на рисунке 30:

- 1 – поля ввода метаданных активного правила в виде блоков списка;
- 2 – раскрывающийся список выбора типа представления;
- 3 – кнопки добавления блоков метаданных активных правил;
- 4 – блок редактирования и удаления активного правила.

На рисунке 30 представлена страница многоблочного активного правила.

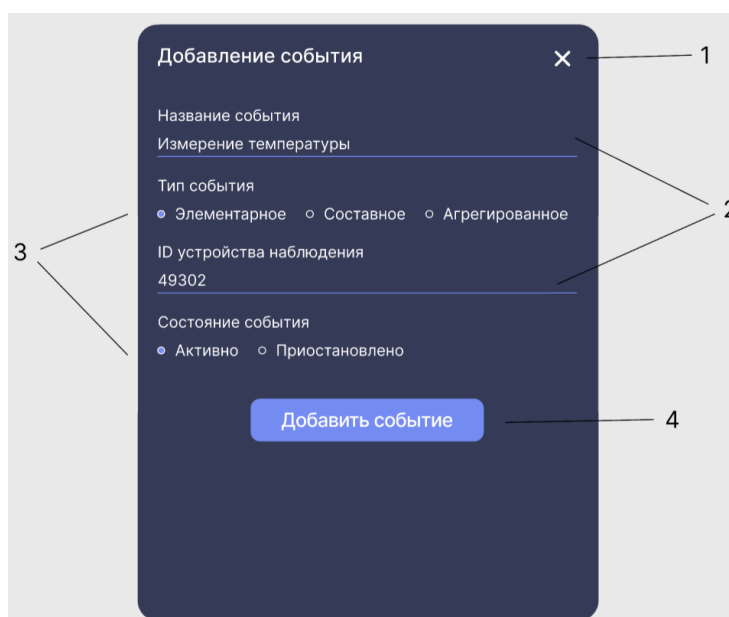


Рисунок 31 – Окно добавления события

Основные элементы окна добавления события, представленного на рисунке 31:

- 1 – кнопка закрытия окна добавления события;
- 2 – поля ввода текстовых значений метаданных события;
- 3 – поля ввода в виде переключателей;
- 4 – кнопка добавления события.

Окно добавления является общим компонентом пользовательского интерфейса при добавлении систем, подсистем, устройств, пользователей, активных правил и событий. На страницах меняются только поля ввода, так как компоненты имеют разные метаданные.

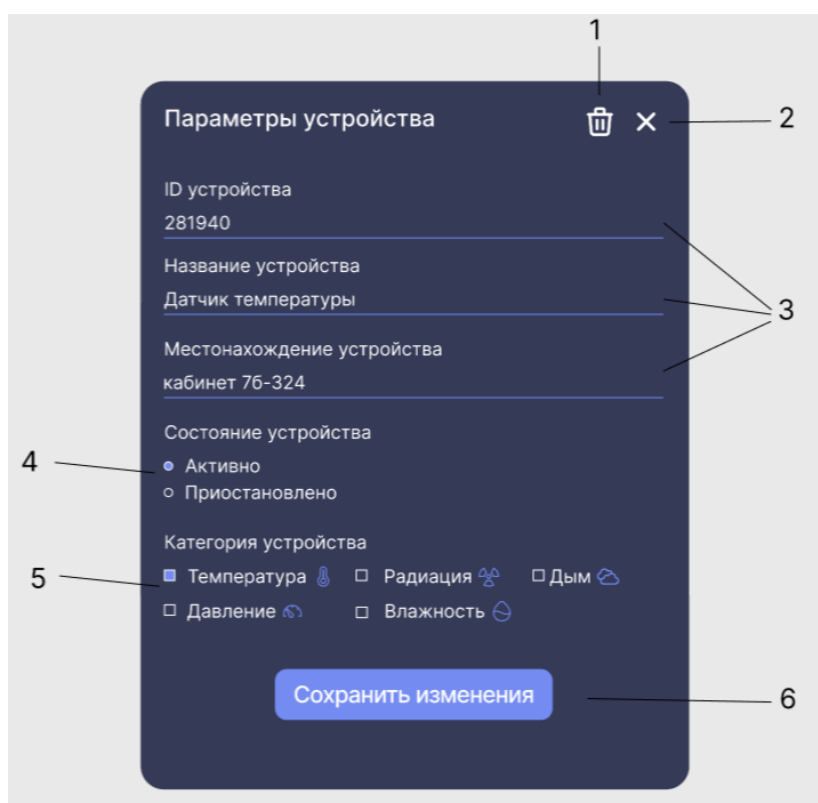


Рисунок 32 – Окно редактирования метаданных устройств

Основные элементы окна редактирования метаданных устройств, представленного на рисунке 32:

- 1 – кнопка удаления устройства;
- 2 – кнопка закрытия окна редактирования метаданных устройства;
- 3 – поля ввода текстовых значений метаданных устройства;

- 4 – поля ввода в виде переключателей;
- 5 – поля ввода в виде флажков;
- 6 – кнопка сохранения изменений.

Окно редактирования является общим компонентом пользовательского интерфейса при редактировании метаданных систем, подсистем, устройств, пользователей, активных правил и событий. Аналогично окну добавления, на страницах меняются только поля ввода, так как компоненты имеют разные метаданные.

Визуальные прототипы пользовательского интерфейса других страниц представлены в приложении Д.

3.3 Разработка программного обеспечения приложения

3.3.1 Разработка программных модулей уровня модели

Разработка программного обеспечения пользовательского web-приложения осуществлялась с использованием шаблона программирования Model-View-Controller (MVC).

Приложение, построенное с использованием паттерна проектирования MVC, разбивается на три слабосвязанных между собой логических компонента: модель, представление и контроллер.

Использование шаблона MVC имеет ряд преимуществ:

- разделение ответственности за счёт разбиения приложения на слабосвязанные компоненты;
- гибкости и масштабируемость (например, при изменении представления нет необходимости менять модель или контроллер);
- структурированный подход к организации кода программного обеспечения.

В рамках этого и следующих разделов была рассмотрена разработка программных модулей уровня модели, контроллера и представления на примере реализации управления метаданными систем пользователем.

Модель представляет собой компонент приложения, отвечающий за взаимодействие с источником данных, а также содержащий описание объектов, описывающих данные, с которыми работает приложение [14].

Модель системы представляет собой интерфейс, описанный в листинге 1.

Листинг 1 – Модель управления метаданными системы

```
export interface System {
  id: string;
  name: string;
  address: string;
  users: {[login: string]: string};
  descriptions: string;
  podsystems: string[];
  objects: string[];
}
```

Интерфейс представляет собой описание метаданных системы без реализации функций. Модель системы хранит в себе идентификатор, название, адрес и описание системы в виде текстовых значений. Также модель системы содержит пользователей системы в формате ассоциативного массива, где ключом является логин пользователя, а значением – роль пользователя в системе. Устройства и подсистемы системы хранятся в виде массивов, которые содержат идентификаторы устройств и подсистем.

Модели всех компонентов были разработаны на языке TypeScript, который поддерживает строгую типизацию данных. Листинги моделей всех компонентов представлены в приложении Б.

3.3.2 Разработка программных модулей уровня контроллера

Контроллер – компонент, описывающий логику взаимодействия с пользователем – в случае веб-приложения логику обработки HTTP-запросов к веб-приложению [14]. Контроллер принимает пользовательские запросы и выполняет необходимые манипуляции с моделью, которые приводят к изменению представления [15].

Контроллер управления метаданными систем содержит 4 основных функции: `getSystems`, `addSystem`, `updateSystem`, `deleteSystem`.

Функция `getSystems` позволяет получить данные всех доступных систем. Функция `addSystem` реализует возможность добавления новой системы. Функция `deleteSystem` удаляет выбранную систему из приложения. Функция `updateSystem` реализует редактирование метаданных системы (листинг 2). Функции добавления, удаления и редактирования системы доступны только пользователю, имеющему роль главного администратора.

Листинг 2 – Функция редактирования метаданных системы

```
exports.updateSystem = (req, res) => {
  const { id } = req.params;
  const { name, address, descriptions, admin } = req.body;
  const systems = JSON.parse (fs.readFileSync
    (SYSTEMS_FILE, 'utf-8'));
  const users = JSON.parse (fs.readFileSync (USERS_FILE,
    'utf-8'));
  const index = systems.findIndex(s => s.id === id);

  if (index === -1) {
    return res.status(404).json({error: 'Система не
      найдена'});
  }

  systems[index].name = name;
  systems[index].address = address;
  systems[index].descriptions = descriptions;
  const oldAdminEntry = Object.entries
    (systems[index].users.find(([, role]) =>
      role === 'Администратор'));

  if (oldAdminEntry) {
    delete systems[index].users[oldAdminEntry[0]];
  }

  systems[index].users[admin] = 'Администратор';

  if (oldAdminEntry) {
    const oldUserIndex = users.findIndex(u => u.login ===
      oldAdminEntry[0]);

    if (oldUserIndex !== -1 && users[oldUserIndex].systems) {
      delete users[oldUserIndex].systems[id];
    }
  }

  const newUserIndex = users.findIndex(u => u.login === admin);

  if (newUserIndex !== -1) {
```

```

        if (!users[newUserIndex].systems)
            users[newUserIndex].systems = {};
        users[newUserIndex].systems[id] = 'Администратор';
    }

    fs.writeFileSync(SYSTEMS_FILE, JSON.stringify(systems, null,
        2), 'utf-8');
    fs.writeFileSync(USERS_FILE, JSON.stringify(users, null, 2),
        'utf-8');
    res.json({ message: ` `, system: systems[index]});
};

```

Изначально функция получает измененные метаданные выбранной системы. Если пользователь изменил администратора системы, то удаляется предыдущий администратор из системы и удаляется запись у пользователя о роли администратора в текущей системе. В систему добавляется новый пользователь с ролью администратора и запись у выбранного пользователя о новой роли.

Для маршрутизации HTTP-запросов от пользователя контроллеру использовался компонент Router, который позволяет определять конкретные маршруты и промежуточное программное обеспечения для различных частей приложения.

Контроллеры управления метаданными и маршрутизация для запросов для всех компонентов были разработаны на языке программирования JavaScript. Полный листинг реализации контроллера управления метаданными системы и компонента маршрутизации запросов представлен в приложении В.

3.3.3 Разработка программных модулей уровня представления

Представление – компонент, отвечающий за отображение пользовательского интерфейса и прослушивание любых взаимодействий пользователя с ним [16].

Представление компонента систем состоит из трёх основных частей:

- подключение модулей в текущий файл (изображений, иконок, стилей);

- определение состояний компонентов, констант, функций представления;

- HTML-разметка с использованием стилей CSS.

Представление содержит следующие основные функции:

- `filteredSystems`: реализует фильтрацию систем по названию;
- `validateForm`: проверяет метаданные системы на соответствие указанным требованиям;
- `handleDeleteSystem`: реализует перенаправление HTTP-запроса удаления системы контроллеру;
- `handleAddSystem`: реализует перенаправление HTTP-запроса добавления системы контроллеру;
- `handleSaveChanges`: реализует перенаправление HTTP-запроса редактирования метаданных системы контроллеру;
- `handlePerexodChanges`: перенаправляет на страницу подсистем выбранной системы;
- `getAdminLogin`: получает логин администратора.

Реализация функции `handleDeleteSystem` представлена в листинге 3. Использование функции в HTML-разметке представлено в листинге 4.

Листинг 3 – Функция перенаправления запроса удаления системы контроллеру

```
const HandleDeleteSystem = async () => {
  if (!selectedSystem) return;

  if (!window.confirm('Удалить систему "$selectedSystem.name"')) return;

  try{
    const response = await fetch('http://localhost:5000/api/systems/${selectesSystem.id}', {method: 'DELETE'});
    if (!response.ok) throw new Error('Ошибка при удалении');
    const [userDataRes, systemRes] = await promise.all([
      fetch('http://localhost:5000/api/users/${username}'),
      fetch('http://localhost:5000/api/systems')
    ]);
    const userData = await userDataRes.json();
    const systemsData = await systemRes.json();
    setUserSystems(userData.systems || {});
    setSystems(systemsData);
  }
```

```

        setSelectedSystemId(null);
    } catch (e: any) {
        alert(e.message);
    }
};

```

Листинг 4 – Использование функции удаления в HTML-разметке

```

<div className="add-header">
  <div className="add-title"> Параметры системы </div>
  {isRootUser && (
    <img src={trashIcon} alt="delete" title="Удалить систему"
      onClick={handleDeleteSystem} className="trashicon"/>
  )}
</div>

```

При нажатии пользователя на кнопку `trashIcon` срабатывает обработчик событий `onClick`, связанный с асинхронной функцией `handleDeleteSystem`. При удалении системы у пользователя запрашивается подтверждение действия. Для удаления системы HTTP-запрос `DELETE` отправляется к контроллеру для выполнения запроса. Параллельно выполняются сразу два запроса: удаление пользователя из метаданных системы и удаление системы и роли пользователя в ней из метаданных пользователя. После успешного выполнения запроса обновляются состояния приложений.

Представления для всех компонентов были разработаны на языке программирования `TypeScript` с использованием HTML-разметки и таблицы стилей `CSS`. Полный листинг реализации представления и описание стилей представлено в приложении Г.

4 Контроль качества программного обеспечения пользовательского web-приложения управления метаданными

4.1 Планирование процесса контроля качества приложения

4.1.1 Мероприятия по контролю качества

Качество программного обеспечения – совокупность свойств, характеризующих способность программного обеспечения удовлетворять потребность пользователя в соответствии с назначением. Обеспечение качества – совокупность планируемых и систематически проводимых мероприятий, необходимых для уверенности в том, что продукция или процессы удовлетворяют определенным требованиям к качеству [17].

Для гарантии качества программного продукта необходимо обеспечить следующие его характеристики:

- надёжность – свойство, обеспечивающее способность сохранять работоспособность за определённый период времени;
- функциональность – свойство, позволяющее выполнять функции в соответствии с назначением;
- удобство – свойство, обеспечивающее простое освоение программы, внесение изменений в программу;
- эффективность – свойство программного продукта, обеспечивающее требуемый уровень производительности в соответствии с выделенными ресурсами, временем и другими условиями;
- сопровождаемость – свойство, обеспечивающее адаптацию к диагностике отказов, сбоев и последствий внесения изменений;
- переносимость – свойство, обеспечивающее адаптацию к новой среде функционирования [18].

Мероприятия по контролю качества необходимы для обеспечения соответствия разработанного программного обеспечения установленным требованиям.

Мероприятия по контролю качества позволяют:

- выявить и предотвратить ошибки и несоответствия на ранних этапах разработки;
- гарантировать соответствие между требованиями и программным продуктом;
- повысить надежность за счёт выявления ошибок;
- повысить качество программного продукта за счёт устранения недочётов.

Для обеспечения качества программного обеспечения были применены различные методы тестирования, направленные на выявление дефектов. Кроме того, была проведена оценка качества программного кода.

4.1.2 Обоснование выбора модели тестирования

Тестирование – это процесс проверки программного обеспечения на соответствие поставленным требованиям. Тестирование включает в себя выполнение тестовых сценариев и анализ результатов их выполнения.

Основные виды тестирования:

- модульное тестирование;
- интеграционное тестирование;
- функциональное тестирование;
- конфигурационное тестирование;
- сквозное тестирование;
- тестирование производительности.

Модульное тестирование – тестирование, направленное на проверку отдельных модулей программного продукта. Такой вид тестирования обеспечивает гарантию того, что каждый компонент работает без ошибок и сбоев.

Интеграционное тестирование применяется для проверки работы программных компонентов как общая единая система. Позволяет выявить и

устранить недочёты, которые могут возникнуть при изменении одного модуля или компонента.

Функциональное тестирование направлено на проверку корректности работы функций приложения, гарантируя, что каждая заявленная функция выполняется корректно и без ошибок.

Конфигурационное тестирование – тестирование, которое используется для проверки совместимости с аппаратной платформой и операционной системой. Применяется для гарантии того, что программное обеспечение устойчиво к изменениям в конфигурации системы и продолжает свою работу без сбоев.

Сквозное тестирование направлено на проверку работы приложения при разных сценариях использования. Используется для гарантии того, что все компоненты системы работают корректно.

Тестирование производительности направлено на исследование показателей скорости реакции приложения на внешние воздействия при различной по характеру и интенсивности нагрузке [19]. Целью тестирования является выявление недочётов производительности для обеспечения эффективной работы системы.

Для проверки работоспособности разработанного веб-приложения использовалось два вида тестирования: функциональное и конфигурационное.

Функциональное тестирование обладает следующими преимуществами:

- обеспечение соответствия программного продукта поставленным требованиям, так как на этапе тестирования проверяется выполнение всех функций;
- снижение рисков за счёт выявления ошибок на ранних этапах;
- повышение надёжности программного обеспечения в связи с устранением ошибок и исправлением некорректного выполнения функций.

Преимущества конфигурационного тестирования заключаются в обеспечении совместимости, так как гарантируется корректная работа на

различных аппаратных и программных конфигурациях. Также конфигурационное тестирование улучшает пользовательский опыт за счёт обеспечения удобства использования программного продукта в различных средах.

4.1.3 Обоснование выбора метрик программного кода

Метрика – мера количественной оценки качества по заданному критерию, система или способ измерений качества программного обеспечения [17].

Определение метрик программного кода позволяет:

- оценить и улучшить качество разработанного программного продукта;
- оценить сложность программы;
- оценить зависимость между компонентами программного обеспечения;
- принять решение о необходимости внесения изменений в код;
- повысить производительности и эффективность кода.

Для оценки характеристик программного кода на основе определения длины реализации и объёма программы используются метрики Холстеда, Джилба и Чепина.

Метрики Холстеда основаны на подсчете количества операторов и операндов, содержащихся в программном коде, и используются для оценки характеристик кода (объема программы, сложности реализации).

Метрики Джилба основаны на определении логической сложности программы с помощью подсчета количества операторов цикла и условия, числа модулей и связей.

Метрики Чепина определяются путём подсчёта количества вводимых, модифицируемых, управляющих, неиспользуемых переменных.

Для оценки качества веб-приложения в качестве метрик программного кода использовались: метрики Холстеда, метрики устойчивости и абстрактности.

Метрики Холстеда используются для выявления наличия:

- последовательности дополняющих друг друга операторов к одному и тому же операнду;
- неоднозначных операндов;
- синонимичных операндов;
- общих подвыражений;
- ненужных присваиваний;
- выражений, которые не представлены в свёрнутом виде как произведение множителей [20].

Метрики устойчивости и абстрактности используются для оценки и улучшения качества программного кода. Абстрактность – это отношение количества абстрактных классов к общему количеству классов в программном компоненте. Устойчивость – это способность одного программного компонента сохранять своё состояние вне зависимости от изменений другого. Высокое значение абстрактности и устойчивости говорит о том, что программный компонент реализован грамотно, так как обеспечивается легкое внесение изменений.

4.1.4 Обоснование выбора методик оценки качества пользовательского интерфейса

Для оценки качества пользовательского интерфейса использовался метод оценки скорости работы с системой Keystroke-level Model, который заключается в представлении всех действий пользователя в виде набора типовых операций (нажатие кнопки, передвижение курсора). Метод позволяет определить время выполнения пользователем определенного сценария. Обычно тот интерфейс лучше, при котором время выполнения задачи меньше [13].

Для использования метода необходимо знать правила разбиения задачи на составляющие и длительность каждой составляющей.

Правила метода Keystroke-level Model:

- нажатие на клавишу (K): 0.28 секунд;
- перемещение курсора мыши (П): 1.1 секунды;
- нажатие на кнопку мыши (М): 0.1 секунды;
- взятие или бросание мыши (В): 0.4 секунды;
- продолжительность выбор действия (Д): 1.2 секунды;
- время реакции системы (Р): от 0.1 секунды.

Метод оценки скорости работы с системой Keystroke-level Model позволяет выявить неэффективные методы выполнения задач и сравнить различные варианты дизайна.

4.2 Тестирование приложения

4.2.1 Функциональное тестирование

Для проверки корректности работы функциональности разработанного веб-приложения было выполнено функциональное тестирование методом «черного ящика», используя функциональную диаграмму и таблицу решений.

В качестве проверяемого компонента использовалось создание составного события. Пользователь должен придумать название события, ввести ID устройств, входящих в это событие, и выбрать состояние события (по умолчанию выбирается «Активно»). Устройства, с указанным пользователем ID, должны уже быть добавлены в систему. Название события может содержать от 4 до 30 символов: цифры, буквы русского и английского алфавита, дефисы, подчеркивания, пробелы. Пользователь не может создать составное событие, если указано меньше 2 существующих устройств.

Реакция программы на пользовательский ввод:

- программа должна сообщать о неверной длине названия события, если его размер не соответствует требуемому;
- программа должна сообщать об использовании некорректных символов в составе названия события, если таковые имеются;

- программа должна сообщать о том, что устройство с таким ID не найдено, если устройства нет в системе;
- программа должна сообщать о том, что указано менее двух устройств, если указано ноль или одно ID устройства;
- программа должна сообщать об успешном создании составного события, если все введенные пользователем данные оказались допустимыми.

На рисунке 33 определены причины и следствия событий.

Причины	
П1	Корректная длина названия события
П2	Название события состоит из корректных символов
П3	Устройство с указанным ID существует
П4	Введено не менее 2 устройств
Следствия	
С1	Сообщение «Некорректная длина названия события»
С2	Сообщение «Некорректные символы в составе названия события»
С3	Сообщение «Устройства с указанным ID не существует»
С4	Сообщение «Введите от 2 ID устройств»
С5	Сообщение «Составное событие успешно создано»

Рисунок 33 – Причины и следствия событий

На основе причин и следствий была составлена функциональная диаграмма (рисунок 34).

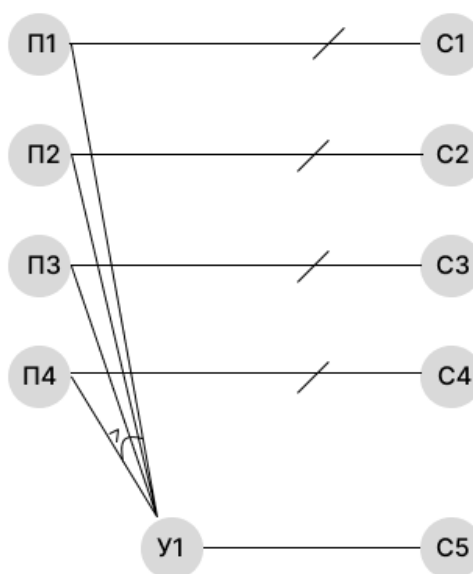


Рисунок 34 – Функциональная диаграмма

На основе диаграммы была составлена таблица решений (таблица 17) и построены наборы тестов (таблица 18).

Таблица 17 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
П1	0	1	0	0	0	1	1	0	0	0	1	1	1	1	0	1
П2	0	0	1	0	0	1	1	1	1	0	0	0	1	0	1	1
П3	0	0	0	1	0	0	1	1	1	1	0	1	0	1	0	1
П4	0	0	0	0	1	0	0	0	1	1	1	1	1	0	1	1
С1	1	0	1	1	1	0	0	1	1	1	0	0	0	0	1	0
С2	1	1	0	1	1	0	0	0	0	1	1	1	0	1	0	0
С3	1	1	1	0	1	1	0	0	0	0	1	0	1	0	1	0
С4	1	1	1	1	0	1	1	1	0	0	0	0	0	1	0	0
С5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Таблица 18 – Тест-кейсы

№	Входные данные	Выходные данные
1	Название: а% ID-устройства:1234	Некорректная длина названия Некорректные символы в названии ID-устройства 1234 не существует Введено менее 2 устройств (рисунок Е.1)
2	Название: абаба\$ ID-устройства: 1234	Некорректные символы в названии ID-устройства 1234 не существует Введено менее 2 устройств (рисунок Е.2)
3	Название: аб ID-устройства:1234	Некорректная длина названия ID-устройства 1234 не существует Введено менее 2 устройств (рисунок Е.3)
4	Название: аб) ID-устройства:1	Некорректная длина названия Некорректные символы в названии Введено менее 2 устройств (рисунок Е.4)
5	Название: аб* ID-устройства:1, 2, 1234	Некорректная длина названия Некорректные символы в названии ID-устройства 1234 не существует (рисунок Е.5)
6	Название: Измерение влаги ID-устройства: 1234	ID-устройства 1234 не существует Введено менее 2 устройств (рисунок Е.6)
7	Название: Измерение влаги ID-устройства: 1	Введено менее 2 устройств (рисунок Е.7)
8	Название: Ипк ID-устройства: 1	Некорректная длина названия Введено менее 2 устройств (рисунок Е.8)
9	Название: Ипк ID-устройства:1, 2	Некорректная длина названия (рисунок Е.9)
10	Название: а+= ID-устройства:1, 2	Некорректная длина названия Некорректные символы в названии (рисунок Е.10)
11	Название: Измерение/ал ID-устройства:1, 2, 1234	Некорректные символы в названии ID-устройства 1234 не существует (рисунок Е.11)
12	Название: Измерение/ал ID-устройства:1, 2	Некорректные символы в названии (рисунок Е.12)

Продолжение таблицы 18

№	Входные данные	Выходные данные
13	Название: Измерение влаги ID-устройства: 1, 2, 1234	ID-устройства 1234 не существует (рисунок Е.13)
14	Название: rao@ ID-устройства: 1	Некорректные символы в названии Введено менее 2 устройств (рисунок Е.14)
15	Название: Агр ID-устройства: 1, 2, 1234	Некорректная длина названия ID-устройства 1234 не существует (рисунок Е.15)
16	Название: Измерение температуры ID-устройства: 1, 2	Событие создано (рисунок Е.16)

На основе результатов функционального тестирования можно сделать вывод: функции выполняются корректно и без ошибок. Результаты функциональных тестов приведены в приложении Е.

4.2.2 Конфигурационное тестирование

Для проверки работы программного обеспечения в различных условиях было проведено конфигурационное тестирование. Основная задача при тестировании состояла в том, чтобы проверить работу веб-приложения в различных браузерах.

Результаты конфигурационного тестирования приведены в таблице 19.

Таблица 19 – Результаты конфигурационного тестирования

Браузер	Тестируемая ситуация	Ожидаемый результат	Наблюдаемый результат
Яндекс Браузер (версия 25.2.1.926 corp-ext)	Главная страница	Главная страница отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.1)
	Страница профиля	Страница профиля отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.2)
	Страница просмотра параметров устройства	Страница просмотра параметров отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.3)
	Страница добавления активного правила в списочном представлении	Страница добавления правила в списочном виде отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.4)
	Страница добавления активного правила в языковом представлении	Страница добавления правила в языковом виде отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.5)

Продолжение таблицы 19

Браузер	Тестируемая ситуация	Ожидаемый результат	Наблюдаемый результат
Яндекс Браузер (версия 25.2.1.926 corp-ext)	Страница добавления активного правила в графическом представлении	Страница добавления активного правила в графическом представлении отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.6)
Google Chrome (версия 137.0.7151.56)	Главная страница	Главная страница отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.7)
	Страница профиля	Страница профиля отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.8)
	Страница просмотра параметров устройства	Страница просмотра параметров устройства отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.9)
	Страница добавления активного правила в списочном представлении	Страница добавления активного правила в списочном представлении отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.10)
	Страница добавления активного правила в языковом представлении	Страница добавления правила в языковом виде отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.11)
	Страница добавления активного правила в графическом представлении	Страница добавления правила в графическом виде отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.12)
Microsoft Edge (версия 136.0.3240.92)	Главная страница	Главная страница отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.13)
	Страница профиля	Страница профиля отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.14)
	Страница просмотра параметров устройства	Страница просмотра параметров отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.15)
	Страница добавления активного правила в списочном представлении	Страница добавления правила в списочном виде отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.16)

Окончание таблицы 19

Браузер	Тестируемая ситуация	Ожидаемый результат	Наблюдаемый результат
Microsoft Edge (версия 136.0.3240.92)	Страница добавления активного правила в языковом представлении	Страница добавления активного правила в языковом представлении отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.17)
	Страница добавления активного правила в графическом представлении	Страница добавления активного правила в графическом представлении отображается корректно	Страница отображена корректно, тест пройден (рисунок Ж.18)

На основе результатов конфигурационного тестирования можно сделать вывод: разработанное пользовательское web-приложение работает корректно в различных браузерах. Результаты конфигурационных тестов приведены в приложении Ж.

4.3 Оценка качества программного кода

4.3.1 Метрики устойчивости и абстрактности

Для определения таких характеристик программного обеспечения как гибкость и поддерживаемость были рассчитаны метрики устойчивости и абстрактности на основе программных модулей. Для реализации веб-приложения использовалось 9 модулей, а именно Main, Account, Systems, Objects, Users, Podsystems, Events, Rules, Journal.

Устойчивость рассчитывается по следующей формуле:

$$I \text{ (Устойчивость)} = \frac{C_e}{(C_a + C_e)},$$

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

C_a (входящие связи) – количество классов вне данного компонента, зависящих от классов внутри компонента;

Се (исходящие связи) – количество классов внутри данного компонента, зависящих от классов вне компонента.

Был выполнен расчет показателя устойчивости для каждого программного компонента приложения:

$$I_1 = \frac{4}{(8 + 4)} = 0,333;$$

$$I_2 = \frac{4}{(8 + 4)} = 0,333;$$

$$I_3 = \frac{4}{(8 + 4)} = 0,333;$$

$$I_4 = \frac{3}{(4 + 3)} = 0,429;$$

$$I_5 = \frac{4}{(6 + 4)} = 0,4;$$

$$I_6 = \frac{5}{(4 + 5)} = 0,555;$$

$$I_7 = \frac{5}{(4 + 5)} = 0,555;$$

$$I_8 = \frac{5}{(2 + 5)} = 0,714;$$

$$I_9 = \frac{3}{(3 + 3)} = 0,5.$$

Абстрактность рассчитывается по следующей формуле:

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

где А – показатель абстрактности, значение которого изменяется в диапазоне от 0 до 1 (если А = 0, то в компоненте нет абстрактных классов, если А = 1 – компонент содержит только абстрактные классы);

Нс – количество классов в компоненте;

На – количество абстрактных классов внутри компонента.

Был выполнен расчет показателя абстрактности для каждого программного компонента приложения:

$$A_1 = \frac{0}{1} = 0;$$

$$A_2 = \frac{1}{2} = 0,5;$$

$$A_3 = \frac{1}{3} = 0,333;$$

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

$$A_5 = \frac{1}{3} = 0,333;$$

$$A_6 = \frac{1}{4} = 0,25;$$

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

$$A_8 = \frac{1}{3} = 0,333;$$

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

В виде графика можно представить соотношение между устойчивостью и абстрактностью. Максимально устойчивые и абстрактные компоненты находятся в левом верхнем углу, а максимально неустойчивые и конкретные – в правом нижнем углу. График соотношения представлен на рисунке 35.

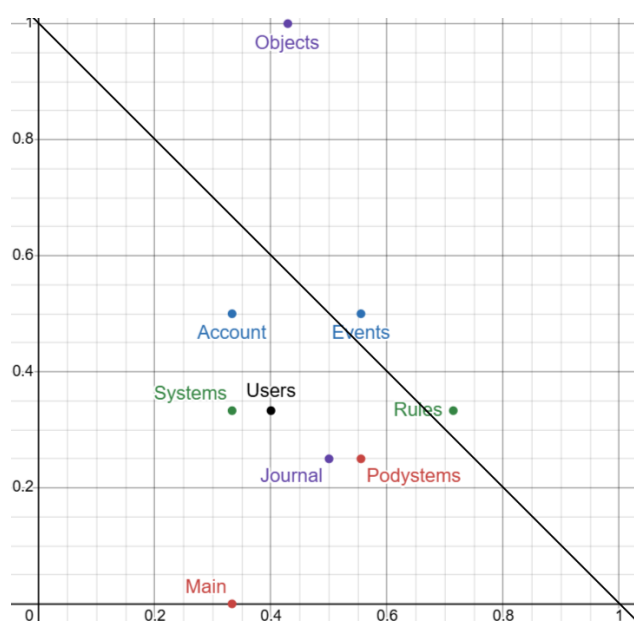


Рисунок 35 – График соотношения между устойчивостью и абстрактностью

Для измерения, насколько далеко компонент отстоит от главной последовательности, используется следующая формула:

$$D = \frac{|A + I - 1|}{\sqrt{2}},$$

где D – показатель расстояния компонента от главной последовательности, который изменяется в диапазоне от 0 до 0.707 (если D равно 0, то компонент находится на главной последовательности, если D равно 0.707 – компонент максимально удален от нее [21]).

Был выполнен расчет показателя D для каждого программного компонента:

$$D_1 = \frac{|0 + 0,333 - 1|}{\sqrt{2}} = 0,472;$$

$$D_2 = \frac{|0,5 + 0,333 - 1|}{\sqrt{2}} = 0,118;$$

$$D_3 = \frac{|0,333 + 0,333 - 1|}{\sqrt{2}} = 0,236;$$

$$D_4 = \frac{|0,429 + 1 - 1|}{\sqrt{2}} = 0,303;$$

$$D_5 = \frac{|0,4 + 0,333 - 1|}{\sqrt{2}} = 0,189;$$

$$D_6 = \frac{|0,25 + 0,555 - 1|}{\sqrt{2}} = 0,139;$$

$$D_7 = \frac{|0,5 + 0,555 - 1|}{\sqrt{2}} = 0,039;$$

$$D_8 = \frac{|0,714 + 0,333 - 1|}{\sqrt{2}} = 0,033;$$

$$D_9 = \frac{|0,25 + 0,5 - 1|}{\sqrt{2}} = 0,177.$$

По графику соотношения и по показателю расстояния компонента от главной последовательности можно заметить, что практически все компоненты находятся рядом с главной последовательностью. Это означает, что

компоненты имеют верное соотношение устойчивости и абстрактности. Компонент Main располагается в зоне неприятностей, так как показатель его абстрактности равен 0 за счет независимости от других компонентов. Компонент Object также располагается в зоне неприятностей, так как имеет высокий показатель абстрактности.

4.3.2 Метрики Холстеда

Для оценки характеристик программного обеспечения были использованы метрики Холстеда.

Для расчета показателей использовался контроллер компонента управления метаданными пользователя. Код компонента представлен в листинге 5.

Листинг 5 – Код контроллер компонента управления метаданными пользователя

```
const fs = require('fs');
const path = require('path');
const USERS_FILE = path.join(__dirname, '../data/users.json');

exports.getUser = (req, res) => {
  const login = req.params.login;
  const users = JSON.parse(fs.readFileSync(USERS_FILE, 'utf-8'));
  const user = users.find(u => login === login);

  if (!user) return res.status(404).json({ message:
    'Пользователь не найден' });

  res.json(user);
};

exports.updateUser = (req, res) => {
  const login = req.params.login;
  const {fullName, phone, email, login: newLogin, password,
    systems } = req.body;
  const users = JSON.parse(fs.readFileSync(USERS_FILE, 'utf-8'));
  const userIndex = users.findIndex(u => u.login === login);

  if (userIndex === -1) return res.status(404).json({ message:
    'Пользователь не найден' });

  if (login !== newLogin && users.find(u => u.login ===
    newLogin)) {
```

```

        return res.status(404).json({ message: 'Логин уже
            используется другим пользователем' });
    }

    users[userIndex] = {fullName, phone, email, login: newLogin,
        password, systems: systems || {} };
    fs.writeFileSync(USERS_FILE, JSON.stringify(users, null, 2),
        'utf-8');
    res.json({ message: 'Профиль обновлён' });
};

exports.registerUser = (req, res) => {
    const {fullName, phone, email, login, password } = req.body;

    if (!fullName || !phone || !email || !login || !password) {
        return res.status(400).json({ message: 'Все поля
            обязательны' });
    }

    const users = JSON.parse(fs.readFileSync(USERS_FILE, 'utf-8'));

    if (users.find(user => user.login === login)) {
        return res.status(409).json({ message: 'Пользователь с
            таким логином уже существует' });
    }

    const newUser = {fullName, phone, email, login, password,
        systems: {} };
    users.push(newUser);
    fs.writeFileSync(USERS_FILE, JSON.stringify(users, null, 2),
        'utf-8');
    res.status(201).json({ message: 'Регистрация успешна' });
};

```

В таблице 20 приведены операторы и число их вхождений в компонент.

Таблица 20 – Операторы

Операторы	Число вхождений
:	11
const	13
require()	2
=	16
===	5
()	12
[]	1
.	11
!==	1
&&	1
	5
if	7
return	5

Продолжение таблицы 20

Операторы	Число вхождений
.json()	9
.status()	6
.readFileSync()	3
.writeFileSync()	2
JSON.parse	3
JSON.stringify	2
.find	4
.findIndex()	1
.push	1
=>	7
{ }	11
,	34
;	28
n1 = 27	N1 = 201

В таблице 21 приведены операнды и число их вхождений в программный компонент.

Таблица 21 – Операнды

Операнды	Число вхождений
fs	7
path	3
USERS_FILE	6
req	7
res	11
login	18
fullName	5
email	5
users	12
user	5
userIndex	3
systems	4
newUser	2
password	5
newLogin	4
'../data/users.json'	1
'utf-8'	5
message	7
'Пользователь не найден'	2
404	2
'Логин уже используется другим пользователем'	1
'Профиль обновлён'	1
'Все поля обязательны'	1
'Регистрация успешна'	1
400	1

Продолжение таблицы 21

Операнды	Число вхождений
409	1
$n2 = 26$	$N2 = 120$

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 321.$$

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 53.$$

Вычислительная сложность программы рассчитывается по формуле:

$$H_V = N * \log_2 n = 1838,688.$$

Структурная сложность программы рассчитывается по формуле:

$$CC = \text{кол_во_операторов_if} + \text{кол_во_операторов_цикла} + 1 = 8.$$

Так, как значение показателя структурной сложности меньше 10, то программный код не требует упрощения.

Количество строк кода (LoC) – 37.

Комплексный показатель качества кода рассчитывается по формуле:

$$MI = \max(0, ((171 - 5,2 * \ln H_V - 0,23 * CC - 16,2 * \ln LoC) * 100 / 171)) = 41.$$

Комплексный показатель качества кода указывает на то, что программа имеет средний уровень сопровождаемости.

4.4 Оценка качества пользовательского интерфейса

Для оценки качества пользовательского интерфейса методом Keystroke-level Model использовалась функция добавления классического активного правила в списочном представлении в систему со следующими данными:

- название активного правила – «Работа кондиционера»;
- тип правила – «Классическое»;
- состояние правила – «Активно»;
- ID события – «1»;
- условие – «t(ID:4323)>28»;
- действие – «Включить ID:5943».

Сценарий добавления классического активного правила в списочном представлении приведен в таблице 22.

Таблица 22 – План тестирования методом Keystroke-level Model

№	Тип действия	Продолжительность	Примечание
1	Д	1,2	Пользователь анализирует задачу и создает алгоритм ее решения
2	В	0,4	Пользователь берёт мышь
3	П	1,1	Пользователь передвигает курсор на кнопку добавления правила
4	М	0,1	Пользователь нажимает на кнопку добавления
5	П	1,1	Пользователь передвигает курсор мыши на поле заполнения названия
6	М	0,1	Пользователь нажимает на поле
7	В	0,4	Пользователь кладет мышь
8	К	0,28*19	Пользователь вводит на клавиатуре 19 символов
9	В	0,4	Пользователь берет мышь
10	П	1,1	Пользователь перемещает курсор на выбор типа
11	М	0,1	Пользователь нажимает на классическое правило
12	П	1,1	Пользователь перемещает курсор на выбор состояния правила
13	М	0,1	Пользователь нажимает на активное состояние
14	П	1,1	Пользователь перемещает курсор на поле ввода события
15	М	0,1	Пользователь нажимает на поле ввода
16	В	0,4	Пользователь убирает мышь
17	К	0,28*1	Пользователь вводит на клавиатуре 1 символ
18	В	0,4	Пользователь берет мышь
19	П	1,1	Пользователь перемещает курсор на поле ввода условия
20	М	0,1	Пользователь нажимает на поле ввода
21	В	0,4	Пользователь убирает мышь
22	К	0,28*13	Пользователь вводит на клавиатуре 13 символов
23	В	0,4	Пользователь берет мышь
24	П	1,1	Пользователь перемещает курсор на поле ввода действия
25	М	0,1	Пользователь нажимает на поле ввода
26	В	0,4	Пользователь убирает мышь
27	К	0,28*16	Пользователь вводит на клавиатуре 16 символов
28	В	0,4	Пользователь берет мышь
29	П	1,1	Пользователь перемещает курсор на кнопку добавления
30	М	0,1	Пользователь нажимает на кнопку добавления
31	Р	0,1	Система уведомляет об успешном добавлении

Таким образом, для добавления классического активного правила в списочном представлении необходимо, чтобы пользователь выполнил 30 действий общей длительностью 28,22 секунд. Полученное время для выполнения добавления активного правила мало, что говорит о грамотном построении пользовательского интерфейса.

Заключение

Разработки выпускной квалификационной работы состояла из 4 основных этапов: анализ предметной области и планирование процесса разработки, анализ требований и постановка задачи на разработку, разработка приложения, контроль качества программного обеспечения.

На первом этапе были определены назначение и основные возможности платформы, также была спроектирована архитектура платформы для развертывания, эксплуатации и сопровождения информационно-управляющих систем. Была определена концептуальная модель системы и спроектирована диаграмма деятельности системы. Была определена объектная модель метаданных платформы, а также составлена диаграмма классов метамодели. Были определены жизненный цикл метаданных и бизнес-процессы управления метаданными, для чего была составлена диаграмма деятельности. Был проведен сравнительный анализ пользовательских приложений управления метаданными. Также был спланирован процесс разработки приложения, который определил, что для реализации проекта потребуется 107330 рублей и 352 часа работы.

На втором этапе были определены функциональные и нефункциональные требования, проанализированы требования к пользовательскому интерфейсу, к аппаратно-программному обеспечению и к средствам разработки. На основе требований была поставлена задача на разработку приложения.

На третьем этапе была описана архитектура развертывания приложения с помощью проектирования диаграммы компонентов и диаграммы пакетов. Были разработаны сценарии взаимодействия пользователя. На их основе были разработаны визуальные прототипы интерфейса. Приложение реализовывалось по принципу MVC, поэтому были разработаны программные модули уровня модели, представления и контроллера.

На четвертом этапе был спланирован и реализован процесс контроля качества приложения. Проведено функциональное и конфигурационное

тестирование, определены метрики устойчивости и абстрактности, метрики Холстеда. Проведена оценка качества пользовательского интерфейса методом Keystroke-level Model.

В результате выполнения всех этапов была разработано пользовательское web-приложение для управления метаданными в составе информационно-управляющей платформы.

Список использованных источников

1. Ключев, А. О. Аппаратные и программные средства встраиваемых систем. Учебное пособие / А. О. Ключев, Д. Р. Ковязина, П. В. Кустарев, А. Е. Платунов. – СПб. : СПбГУ ИТМО, 2010. – 290 с.
2. Шибанов, С. В. Интегрированная модель активных правил / С. В. Шибанов, А. А. Скоробогатько, Э. В. Лысенко // Математическое и программное обеспечение систем в промышленной и социальной сферах : Междунар. сб. науч. тр. – Магнитогорск : Изд-во Магнитогорского гос. техн. ун-та им. Г.И. Носова, 2011. – С. 41 – 46.
3. Митра, Р. Микросервисы. От архитектуры до релиза / Р. Митра, И. Надареишвили. – СПб. : Питер, 2023. – 336 с.
4. Richards, M. Software Architecture Patterns / M. Richards. – Sebastopol : O'Reilly Media, Inc., 2022. – 73 с.
5. Уэске, Ф. Поточковая обработка данных с Apache Flink / Ф. Уэске, В. Калаври. – М. : ДМК Пресс, 2021. – 298 с.
6. Литвинова, Н. Н. Методические рекомендации для разработчиков по организации работы открытых репозиторий: корректное создание метаданных / Н. Н. Литвинова. – М. : Ваше цифровое издательство, 2020. – 20 с.
7. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс, Д. Битти. – М. : Издательство «Русская редакция», 2014. – 726 с.
8. Мардан, А. React быстро / А. Мардан, М. Барклунд. – СПб. : Питер, 2024. – 512 с.
9. Мержевич, В. В. HTML и CSS на примерах / В. В. Мержевич. – СПб. : БХВ-Петербург, 2005. – 448 с.
10. Моисеев, А. Н. Основы языка UML : учеб. пособие / А. Н. Моисеев, М. И. Литовченко. – Томск : Издательство Томского государственного университета, 2023. – 96 с.
11. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – М. : ДМК Пресс. – 496 с.

12. Леоненков, А. В. Самоучитель UML 2 / А. В. Леоненков. – СПб. : БХВ-Петербург, 2007. – 576 с.
13. Головач, В. В. Дизайн пользовательского интерфейса / В. В. Головач. – СПб. : Изд-во Usethics, 2000. – 146 с.
14. Магдануров, Г.И. ASP.NET MVC Framework / Г. И. Магдануров, В. А. Юнев. – СПб. : БХВ-Петербург, 2010. – 320 с.
15. Akshat, P. React Native for Mobile Development / P. Akshat, N. Abhishek. – New York : Apress, 2019. – 237 с.
16. Mezzalira, L. Front-End Reactive Architectures / L. Mezzalira. – New York : Apress, 2018. – 188 с.
17. Парфенова, А. Ю. Стандартизация, сертификация и управление качеством программного обеспечения / А. Ю. Парфенова. – Самара : Издательство Самарского университета, 2023. – 84 с.
18. Семахин, А. М. Методы верификации и оценки качества программного обеспечения / А. М. Семахин. – Курган : Изд-во КГУ, 2018. – 150 с.
19. Куликов, С. Тестирование программного обеспечения. Базовый курс / С. Куликов. – Минск : Четыре четверти, 2020. – 300 с.
20. Решетникова, Е. В. Метрология и качество программного обеспечения (Метрики, основанные на лексическом анализе программ) / Е. В. Решетникова. – Новокузнецк : НФИ КемГУ, 2020. – 80 с.
21. Гурьянов, Л. В. Выпускная квалификационная работа: стандарты и программные технологии / Л. В. Гурьянов, В. Н. Князев, И. Ю. Балашова, Е. А. Дзюба, Н. А. Попова, Д. В. Такташкин, С. В. Самуйлов. – Пенза: Изд-во ПГУ, 2020. – 184 с.

Приложение А

(справочное)

Глоссарий

Web-приложение – программное обеспечение, которое функционирует на web-сервере и доступно пользователю через web-браузер.

Фреймворк – программная платформа, предоставляющая готовые шаблоны и набор инструментов для разработки программного обеспечения.

Low-code – подход к разработке, заключающийся в минимальном написании кода.

No-code – подход к разработке, заключающийся в отсутствии необходимости написания кода.

REST (Representational State Transfer) – архитектурный стиль разработки web-сервисов, который основан на принципе взаимодействия клиента и сервера посредством HTTP-запроса.

Event Stream Processing (ESP) – технология обработки потоков данных в реальном времени.

Интернет вещей (Internet of Things) – физические устройства, оснащенные технологиями взаимодействия с другими устройствами и системами.

Document Object Model (DOM) – интерфейс, представляющий структуру HTML-документа в виде дерева объектов.

Application Programming Interface (API) – инструмент для взаимодействия программ, определяющий набор правил и протоколов для обмена данными.

Wireframe – схема с низким уровнем детализации, которая показывает расположение всех основных компонентов приложения (навигации, текстовых блоков, иллюстраций и кнопок).

HTTP-запрос – сообщение, отправляемое клиентом на сервер для выполнения действия или получения информации.

Приложение Б
(обязательное)
Реализация моделей

Листинг Б.1 – Реализация модели компонента пользователя

```
export interface User {
  id: string;
  fullName: string;
  phone: string;
  email: string;
  login: string;
  password: string;
  systems: {[systemId: string]: string};
}
```

Листинг Б.2 – Реализация модели компонента активного правила

```
export interface Rule {
  id: string;
  name: string;
  type: string;
  state: string;
  events: string[];
  conditions: string[];
  actives_true: string[];
  active_false: string;
  categories?: string[];
}
```

Листинг Б.3 – Реализация модели компонента журнала

```
export interface Journal {
  idEvent: string;
  idRule: string;
  value: string[];
  data: string[];
}
```

Листинг Б.4 – Реализация модели компонента подсистемы

```
export interface Podsystem {
  id: string;
  name: string;
  state: string;
  descriptions: string;
  events: string[];
}
```

Листинг Б.5 – Реализация модели компонента устройства

```
export interface Objectt {
  id: string;
  name: string;
  place: string;
  state: string;
  category: string[];
}
```

Листинг Б.6 – Реализация модели компонента события

```
export interface Event{  
    id: string;  
    name: string;  
    type: string;  
    objects: string[];  
    state: string;  
    query: string;  
    time: string;  
}
```

Приложение В

(обязательное)

Реализация контроллера компонента системы

Листинг В.1 – Реализация контроллера управления метаданными компонента системы

```

const fs = require('fs');
const path = require('path');
const SYSTEMS_FILE = path.join(__dirname, '../data/systems.json');
const USERS_FILE = path.join(__dirname, '../data/users.json');

exports.getSystems = (req, res) => {
  const systems = JSON.parse(fs.readFileSync(SYSTEMS_FILE,
    'utf-8'));
  res.json(systems);
}

exports.addSystem = (req, res) => {
  const { name, address, admin, descriptions } = req.body;

  if (!name || !address || !admin){
    return res.status(400).json({error: 'Отсутствуют
      обязательные поля'});
  }

  let systems = JSON.parse(fs.readFileSync(SYSTEMS_FILE,
    'utf-8'));
  let users = JSON.parse(fs.readFileSync(USERS_FILE, 'utf-8'));
  const adminIndex = users.findIndex(u => u.login === admin);

  if (adminIndex === -1){
    return res.status(400).json({error: 'Пользователь с таким
      логином не найден'});
  }

  let rootUser = users.find(u => u.login === 'root');

  if (!rootUser){
    rootUser = {login: 'root', systems: {}};
    users.push(rootUser);
  }

  const lastSystem = systems[systems.length - 1];
  const newId = lastSystem ? (parseInt(lastSystem.id) + 1
    ).toString() : '1';
  const newSystem = {
    id: newId,
    name,
    address,
    users: {
      [admin]: 'Администратор',
      root: 'Главный администратор',
    },
    descriptions: descriptions || '',
  };
};

```

```

systems.push(newSystem);

if (!users[adminIndex].systems) users[adminIndex].systems =
  {};

users[adminIndex].systems[newId] = 'Администратор';

if (!rootUser.systems) rootUser.systems = {};

rootUser.systems[newId] = 'Главный администратор';
fs.writeFileSync(SYSTEMS_FILE, JSON.stringify(systems, null,
  2), 'utf-8');
fs.writeFileSync(USERS_FILE, JSON.stringify(users, null, 2),
  'utf-8');
res.json({ message: 'Система добавлена', system: newSystem});
}

exports.updateSystem = (req, res) => {
  const { id } = req.params;
  const { name, address, descriptions, admin } = req.body;
  const systems = JSON.parse (fs.readFileSync
    (SYSTEMS_FILE, 'utf-8'));
  const users = JSON.parse (fs.readFileSync (USERS_FILE,
    'utf-8'));
  const index = systems.findIndex(s => s.id == id);

  if (index === -1) {
    return res.status(404).json({error: 'Система не
      найдена' });
  }

  systems[index].name = name;
  systems[index].address = address;
  systems[index].descriptions = descriptions;
  const oldAdminEntry = Object.entries
    (systems[index].users.find(([, role]) =>
      role === 'Администратор'));

  if (oldAdminEntry) {
    delete systems[index].users[oldAdminEntry[0]];
  }

  systems[index].users[admin] = 'Администратор';

  if (oldAdminEntry) {
    const oldUserIndex = users.findIndex(u => u.login ===
      oldAdminEntry[0]);

    if (oldUserIndex !== -1 && users[oldUserIndex].systems) {
      delete users[oldUserIndex].systems[id];
    }
  }
}

```

```

    }

    const newUserIndex = users.findIndex(u => u.login === admin);

    if (newUserIndex !== -1){
        if (!users[newUserIndex].systems)
            users[newUserIndex].systems = {};
        users[newUserIndex].systems[id] = 'Администратор';
    }

    fs.writeFileSync(SYSTEMS_FILE, JSON.stringify(systems, null,
        2), 'utf-8');
    fs.writeFileSync(USERS_FILE, JSON.stringify(users, null, 2),
        'utf-8');
    res.json({ message: 'Система обновлена', system:
        systems[index]});
};

exports.deleteSystem = (req, res) => {
    const {id} = req.params;
    let systems = JSON.parse(fs.readFileSync(SYSTEMS_FILE,
        'utf-8'));
    let users = JSON.parse(fs.readFileSync(USERS_FILE, 'utf-8'));
    const systemIndex = systems.findIndex(s => s.id === id);

    if (systemIndex === -1){
        return res.status(404).json({error: 'Система не найдена'
            });
    }

    systems.splice(systemIndex, 1);

    users = users.map(user => {
        if (user.systems && user.systems[id]){
            delete user.systems[id];
        }
        return user;
    });

    fs.writeFileSync(SYSTEMS_FILE, JSON.stringify(systems, null,
        2), 'utf-8');
    fs.writeFileSync(USERS_FILE, JSON.stringify(users, null, 2),
        'utf-8');
    res.json({ message: 'Система удалена'});
}

```

Листинг В.2 – Реализация маршрутизации запросов компонента системы

```

const express = require('express');
const router = express.Router();
const systemController =
    require('../controllers/systemController');

```

```
router.get('/', systemController.getSystems);
router.post('/', systemController.addSystem);
router.put('/:id', systemController.updateSystem);
router.delete('/:id', systemController.deleteSystem);

module.exports = router;
```

Приложение Г

(обязательное)

Реализация представления компонента системы

Листинг Г.1 – Реализация представления компонента системы

```

import React, {useEffect, useState} from 'react';
import Header_log from '../components/Header_log';
import './SystemsPage.css';
import poiskIcon from '../image/poisk.png';
import addIcon from '../image/add.png';
import closeIcon from '../image/close.png';
import trashIcon from '../image/trash.png';
import {User as UserData} from '../models/UserModel';
import {System} from '../models/SystemModel';
import { useNavigate } from 'react-router-dom';
import { search } from '../routes/userRoutes';

const SystemsPage: React.FC = () => {
  const [userSystems, setUserSystems] = useState<{[id: string]:
    string }>({});
  const [systems, setSystems] = useState<System[]>([]);
  const [searchText, setSearchText] = useState('');
  const [isAdding, setIsAdding] = useState(false);
  const [selectedSystemId, setSelectedSystemId] =
    useState<string | null>(null);
  const selectedSystem = systems.find(s => s.id ===
    selectedSystemId) || null;
  const navigate = useNavigate();
  const [newName, setNewName] = useState('');
  const [newAddress, setNewAddress] = useState('');
  const [newAdmin, setNewAdmin] = useState('');
  const [newDescription, setNewDescription] = useState('');
  const [editName, setEditName] = useState('');
  const [editAddress, setEditAddress] = useState('');
  const [editAdmin, setEditAdmin] = useState('');
  const [editDescription, setEditDescription] = useState('');
  const [errors, setErrors] =
    useState<{[key:string]:string}>({});
  const username = localStorage.getItem('userLogin') || 'Гость';
  const isRootUser = username === 'root';

  useEffect(() => {
    fetch('http://localhost:5000/api/users/' + username)
      .then(res => res.json())
      .then ((data: UserData) => {
        setUserSystems(data.systems || {});
      });
    fetch('http://localhost:5000/api/systems')
      .then(res => res.json())
      .then ((data: System[]) => {
        setSystems(data);
      });
  }, [username])

  const filteredSystems = Object.entries(userSystems)
    .map(([id, role]) => {

```

```

        const system = systems.find(s => s.id === id);
        if (!system) return null;
        return {system, role};
    })
    .filter(
        item => item &&
        item.system.name.toLowerCase(
            ).includes(searchText.toLowerCase())
    ) as {system: System; role: string }[];

const validateForm = () => {
    const newErrors: {[key:string]:string} = {};

    if(!newName.trim()){
        newErrors.name = 'Введите название системы';
    } else if (newName.length > 30){
        newErrors.name = 'Название не должно превышать 30
        СИМВОЛОВ';
    } else if (!/^[\\p{L}0-9\\s&%.,\\/_-]+$/u.test(newName)){
        newErrors.name = 'Название должно содержать только
        буквы и пробелы';
    }

    if(!newAddress.trim()){
        newErrors.address = 'Введите адрес системы';
    } else if (!/^[\\p{L}0-9\\s\\\\.\\-]{8,30}$/u.test(newAddress)){
        newErrors.address = 'Адрес содержит недопустимые
        СИМВОЛЫ';
    }

    if(!newAdmin.trim()){
        newErrors.admin = 'Введите логин администратора';
    }

    if(!newDescription.trim()){
        newErrors.descriptions = 'Введите адрес системы';
    } else if (!/^[\\p{L}0-9\\s.,!?:;'()"\\-]{4,100}$/u.test(newDescription)){
        newErrors.descriptions = 'Описание системы от 4 до 100
        СИМВОЛОВ';
    }

    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
};

const handleDeleteSystem = async () =>{
    if (!selectedSystem) return;

    if(!window.confirm('Удалить систему
    "${selectedSystem.name}"?')) return;

```

```

try{
  const response = await fetch('http://localhost:5000/
    api/systems/${selectedSystem.id}',{
    method: 'DELETE'
  })

  if (!response.ok) throw new Error('Ошибка при
    удалении');

  const [userDataRes, systemsRes] = await Promise.all([
    fetch('http://localhost:5000/api/users/${username}
      '),
    fetch('http://localhost:5000/api/systems')
  ])

  const userData = await userDataRes.json();
  const systemsData = await systemsRes.json();
  setUserSystems(userData.systems || {});
  setSystems(systemsData);
  setSelectedSystemId(null);
} catch (e: any){
  alert(e.message);
}
};

const handleAddSystem = async () =>{
  if (!validateForm()) return;

  try{
    const res = await fetch('http://localhost:5000/api/
      users/${newAdmin.trim()}');

    if (!res.ok){
      setErrors(prev => ({ ...prev, admin: 'Пользователя
        с таким логином нет' }));
      return;
    }

    const user = await res.json();

    if (!user || !user.login){
      setErrors(prev => ({ ...prev, admin: 'Пользователя
        с таким логином нет' }));
      return;
    }

    const response = await fetch(
      'http://localhost:5000/api/systems',{
      method: 'POST',

```

```

        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({
            name: newName.trim(),
            address: newAddress.trim();
            admin: newAdmin.trim();
            descriptions: newDescription.trim(),
        })
    });

    if (!response.ok) throw new Error('Ошибка сервера');

    setIsAdding(false);
    setNewName('');
    setNewAddress('');
    setNewAdmin('');
    setNewDescription('');
    setErrors({});
    const [userDataRes, systemsRes] = await Promise.all([
        fetch('http://localhost:5000/api/users/${username}'),
        fetch('http://localhost:5000/api/systems')
    ])
    const userData = await userDataRes.json();
    const systemsData = await systemsRes.json();
    setUserSystems(userData.systems || {});
    setSystems(systemsData);
} catch (e: any){
    alert(e.message);
}

};

const handleSaveChanges = async () =>{
    const newErrors: {[key: string]: string } = {};
    const res = await fetch('http://localhost:5000/api/users/${editAdmin.trim()}');

    if (!res.ok){
        setErrors(prev => ({ ...prev, admin: 'Пользователя с таким логином нет' }));
        return;
    }

    const user = await res.json();

    if (!user || !user.login){
        setErrors(prev => ({ ...prev, admin: 'Пользователя с таким логином нет' }));
        return;
    }

    if(!editName.trim()){
        newErrors.name = 'Введите название системы';
    }

```

```

} else if (editName.length > 30){
    newErrors.name = 'Название не должно превышать 30
    символов';
} else if (!/^[\\p{L}0-9\\s&%.,\\/\\-\\_]+$/u.test(editName)){
    newErrors.name = 'Название должно содержать только
    буквы и пробелы';
}

if(!editAddress.trim()){
    newErrors.address = 'Введите адрес системы';
} else if (!/^[\\p{L}0-9\\s\\/\\.\\-]{8,30}
$/u.test(editAddress)){
    newErrors.address = 'Адрес содержит недопустимые
    символы';
}

if(!editAdmin.trim()){
    newErrors.admin = 'Введите логин администратора';
}

if(!editDescription.trim()){
    newErrors.descriptions = 'Введите адрес системы';
} else if (!/^[\\p{L}0-9\\s.,!?:;'\"()\\-]{4,100}
$/u.test(editDescription)){
    newErrors.descriptions = 'Описание системы от 4 до 100
    символов';
}

if (Object.keys(newErrors).length > 0){
    setErrors(newErrors);
    return;
}

try{
    const response = await fetch('http://localhost:5000
    /api/systems/${selectedSystem?.id}',{
        method: 'PUT',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({
            name: editName.trim(),
            address: editAddress.trim();
            admin: editAdmin.trim();
            descriptions: editDescription.trim(),
        })
    });

    if (!response.ok) throw new Error('ошибка при
        сохранении');

    const [userDataRes, systemsRes] = await Promise.all([
        fetch('http://localhost:5000/api/users/${username}
        '),

```

```

        fetch('http://localhost:5000/api/systems')
    ])

    const userData = await userDataRes.json();
    const systemsData = await systemsRes.json();
    setUserSystems(userData.systems || {});
    setSystems(systemsData);
    setSelectedSystemId(null);
  } catch (e: any) {
    alert(e.message);
  }
};

const handlePerexodChanges = () => {
  navigate('/podsystem/${selectedSystemId}');
};

const getAdminLogin = (system: System) => {
  return Object.entries(system.users).find(([, role]) => role
    === 'Администратор')?.[0] || '';
};

return(
  <div className="systems-page">
    <Header_log />
    <div className="block_main">
      <div className="systems-title"> Системы</div>
      <div className="search-section">
        <div className="search-input" style={{ width:
          isRootUser ? 776 : 828 }}>
          <img src={poiskIcon} alt="search"
            className="search-icon"/>
          <input
            type="text"
            placeholder="Поиск"
            className="search-placeholder"
            value={searchText}
            onChange={e => setSearchText
              (e.target.value)}
            style={{
              border: 'none',
              outline: 'none',
              fontFamily: 'Inter, sans-serif',
              fontWeight: 200,
              fontSize: 16,
              color: 'rgba(0,0,0,0.7)',
              backgroundColor: 'transparent',
              flexGrow: 1,
              padding: 0,
              marginLeft: '10px',
              height: '35px',
              lineHeight: '35px',

```

```

    }}
  />
</div>
{isRootUser && (
  <img
    src={addIcon}
    alt="add"
    className="add-icon"
    onClick={() => setIsAdding(true)}
  />
)}
</div>
<div className="systems-list">
  {Object.entries(userSystems)
    .map(({id, role}) => {
      const system = systems.find(s => s.id
        === id);
      if (!system) return null;
      return {system, role};
    })
    .filter((item): item is {system: System;
      role: string} => item !== null)
    .filter(item => item.system.name.
      toLowerCase().includes(searchText.
        toLowerCase()))
    .map(({systems, role})=> (
      <div className="systems-block"
        key={system.id}
        onClick={() => {
          setSelectedSystemId
            (system.id);
          setEditName(system.name);
          setEditAddress(system.address
            );
          setEditAdmin(getAdminLogin
            (system));
          setEditDescription(system.
            descriptions);
        }}
        style={{
          border: selectedSystemId ===
            system.id ? '1 px solid
              #353A57' : 'none',
          cursor: 'pointer',
        }}
      >
        <div className="systems-info">
          <div className="systems-name">
            {systems.name}
          </div>
          <div className="systems-

```

address

```

        ">
        {systems.address}
      </div>
    </div>
    <div className="systems-role">
      {role}</div>
  )))
  {Object.entries(userSystems)
    .map(([id, role])=>{
      const system = systems.find(s => s.id
        === id);
      if (!system) return null;
      return {system, role};
    })
    .filter(item =>
      !item || !item.system.name.
        toLowerCase().includes
        (searchText.toLowerCase()))

  )
  .length === Object.entries(userSystems).
    length && (
    <div style={{ padding: '10px', color:
      '#666'}}> Системы не найдены
    </div>
  )
}
</div>
<div className="blocktwo">
  {isAdding? (
    selectedSystem ? (
      <div className="add-system-block">
        <div className="add-header">
          <div className="add-title">
            Параметры системы</div>
            {isRootUser && (
              <img
                src={trashIcon}
                alt="delete"
                title="Удалить
                  систему"
                onClick={
                  handleDeleteSystem
                }
                style={{
                  width: 30,
                  height: 30,
                  marginLeft: 144,
                  cursor: 'pointer'
                }}
              />
            )
          }
        )
      )
    }
  )
}

```

```

        <img
            src={closeIcon}
            alt="close"
            className="close-icon"
            onClick={() => {
                setSelectedSystemId
                    (null);
                setErrors({});
            }}
            style={{cursor: 'pointer'
                }}
        />
    </div>
    <div className="label-with-error">
        <label className="label">
            Название системы
        </label>
        {errors.name && <span
            className="error-inline">
                {errors.name}</span>}
    </div>
    <input
        type="text"
        className="input-text"
        value={editName}
        onChange={e => setEditName
            (e.target.value)}
        disabled={!isRootUser}
    />
    <div className="underline"/>
    <div className="label-with-error">
        <label className="label">Адрес
            системы</label>
        {errors.address && <span
            className="error-inline">
                {errors.address}</span>}
    </div>
    <input
        type="text"
        className="input-text"
        value={editAddress}
        onChange={e => setEditAddress
            (e.target.value)}
        disabled={!isRootUser}
    />
    <div className="underline"/>
    <div className="label-with-error">
        <label className="label">
            Администратор системы
        </label>
        {errors.admin && <span
            className="error-inline">

```

```

        {errors.admin}</span>}
</div>
<input
    type="text"
    className="input-text"
    value={editAdmin}
    onChange={e => setEditAdmin
        (e.target.value)}
    disabled={!isRootUser}
/>
<div className="underline"/>
<div className="label-with-error">
    <label className="label">
        Описание системы
    </label>
    {errors.descriptions && <span
        className="error-inline">
            {errors.descriptions}
        </span>}
</div>
<input
    type="text"
    className="input-description"
    value={editDescriptions}
    onChange={e =>
        setEditDescriptions
            (e.target.value)}
    disabled={!isRootUser}
/>
{isRootUser && (<button
    className="add-button"
    style={{top: '20px'}}
    onClick={handleSaveChanges}>
        Сохранить изменения</button>)}
<button className="perexod-button"
    onClick={handlePerexodChanges}
    Перейти в систему</button>
</div>
) : (
    <div className="blocktwo-title">
        {isRootUser ? "Выберите/Добавьте
            систему" : "Выберите систему"}
    </div>
)
) : (
    isRootUser && (
        <div className="add-system-block"
            style={{color: 'white',
                fontFamily: 'Inter,
                    sans-serif'}}>
            <div className="add-header">
                <div className=

```

```

        "add-title">
        Добавление
        системы</div>
<img
    src={closeIcon}
    alt="close"
    className="close-icon"
    onClick={() => {
        setIsAdding
            (false);
        setErrors({});
        setNewName('');
        setNewAddress('');
        setNewAdmin('');
        setNewDescription
            ('');
    }}
/>
</div>
<div className="label-with-
error">
    <label className="label"
        >Название системы
    </label>
    {errors.name && <span
        className="error-
        inline">
            {errors.name}
        </span>}
</div>

<input
    type="text"
    className="input-text"
    value={newName}
    onChange={e =>
        setNewName
            (e.target.value)}
    placeholder="Введите
        название"
/>
<div className=
    "underline"/>
<div className=
    "label-with-error">
    <label className="label">
        Адрес системы
    </label>
    {errors.address && <span
        className="error
        -inline">
            {errors.address}

```

```

        </span>}
</div>
<input
    type="text"
    className="input-text"
    value={newAddress}
    onChange={e =>
        setNewAddress
        (e.target.value) }
    placeholder=
        "Введите адрес"
/>
<div className="underline"/>
<div className="label-with
-error">
    <label className=
        "label">Администратор
        системы</label>
    {errors.admin && <span
        className="error-
        inline">
        {errors.admin}
        </span>}
</div>
<input
    type="text"
    className="input-text"
    value={newAdmin}
    onChange={e =>
        setNewAdmin
        (e.target.value) }
    placeholder="Введите логин
администратора"
/>
<div className="underline"/>
<div className="label-
with-error">
    <label
        className="label">
        Описание системы
    </label>
    {errors.descriptions &&
        <span className=
            "error-inline">
            {errors.
                descriptions}
            </span>}
</div>
<input
    type="text"
    className=
        "input-description"

```

```

        value={newDescriptions}
        maxLength={150}
        onChange={e =>
            setNewDescriptions
                (e.target.value)}
        placeholder="Введите
            описание"
    />
    <button className="add-button"
        onClick={handleAddSystem}>
        Добавить в систему
    </button>
</div>
    )
    )}
</div>
</div>
</div>
);
};
export default SystemsPage;

```

Листинг Г.2 – Реализация представления компонента системы

```

.block_main {
    width: 1496px;
    height: 632px;
    background-color: #EAEDF4;
    border-radius: 20px;
    margin: 20px auto 0 auto;
    position: relative;
}

.blocktwo {
    width: 472px;
    height: 590px;
    background-color: #353A57;
    border-radius: 20px;
    position: absolute;
    top: 10px;
    right: 10px;
    color: white;
    display: flex;
    align-items: flex-start;
    justify-content: center;
    padding-top: 20px;
}

.blocktwo-title {
    width: 330px;
    height: 78px;
    margin-top: 260px;
    font-family: 'Inter', sans-serif;
}

```

```

        font-weight: 600;
        font-size: 28px;
        color: #FFFFFF;
        text-align: center;
    }

.add-system-block {
    width: 472px;
    height: 590px;
    background-color: #353A57;
    border-radius: 20px;
    position: absolute;
    color: white;
}

.add-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 40px;
    margin-left: 20px;
}

.add-title {
    font-weight: 600;
    font-size: 20px;
    color: white;
}

.close-icon {
    cursor: pointer;
    width: 30px;
    height: 30px;
    margin-right: 20px;
}

.label {
    height: 19px;
    font-weight: 400;
    font-size: 16px;
    margin-bottom: 10px;
    color: white;
}

.input-text {
    width: 432px;
    height: 19px;
    background: transparent;
    border: none;
    color: white;
    font-family: 'Inter', medium;
    font-weight: 400;
}

```

```

        font-size: 16px;
        outline: none;
        margin-left: 20px;
    }

    .input-description {
        width: 410px;
        height: 120px;
        margin-top: 10px;
        padding: 10px;
        background: white;
        border-radius: 10px;
        border: none;
        font-family: 'Inter', medium;
        font-weight: 400;
        font-size: 16px;
        color: black;
        resize: none;
        outline: none;
        margin-left: 20px;
    }

    .underline {
        width: 432px;
        height: 1px;
        background-color: #748BF1;
        margin-bottom: 20px;
        margin-left: 20px;
    }

    .add-button {
        width: 240px;
        height: 44px;
        background-color: #748BF1;
        color: white;
        border: none;
        border-radius: 10px;
        font-family: 'Inter', sans-serif;
        font-weight: 600;
        font-size: 16px;
        cursor: pointer;
        margin-left: 116px;
        margin-top: 10px;
    }

    .perexod-button {
        margin-top: 10px;
        width: 240px;
        height: 44px;
        background-color: #353A57;
        color: white;
        border: 1px solid white;
    }

```

```

    border-radius: 10px;
    font-family: 'Inter', sans-serif;
    font-weight: 600;
    font-size: 16px;
    cursor: pointer;
    margin-left: 116px;
}

.label-with-error {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin: 0 20px;
}

.error-inline {
    color: #ff5555;
    font-size: 12px;
    margin-left: 10px;
}

.systems-title {
    font-family: 'Inter', sans-serif;
    font-weight: 500;
    font-size: 20px;
    color: #000000;
    position: absolute;
    top: 20px;
    left: 20px;
}

.search-section {
    display: flex;
    align-items: center;
    margin: 20px 20px 20px 20px;
}

.search-input {
    width: 776px;
    height: 35px;
    background-color: #ffffff;
    position: relative;
    border-radius: 10px;
    font-family: 'Inter', sans-serif;
    display: flex;
    align-items: center;
    padding-left: 16px;
    margin-top: 64px;
    position: relative;
}

.search-icon {

```

```

        position: static;
        margin-right: 10px;
        width: 14px;
        height: 14px;
    }

    .search-placeholder {
        color: rgba(255, 255, 255, 0.7);
        font-size: 16px;
        font-weight: 200;
    }

    .add-icon {
        width: 31px;
        height: 31px;
        margin-left: 20px;
        margin-top: 66px;
        cursor: pointer;
    }

    .systems-list {
        margin: 20px;
        max-width: 842px;
    }

    .systems-block {
        width: 842px;
        height: 68px;
        background-color: #FFFFFF;
        border-radius: 10px;
        margin-bottom: 10px;
        position: relative;
        padding: 10px 16px;
        box-sizing: border-box;
    }

    .systems-info {
        display: flex;
        flex-direction: column;
    }

    .systems-name {
        font-family: 'Inter', sans-serif;
        font-weight: 500;
        font-size: 20px;
        color: #000000;
    }

    .systems-address {
        font-family: 'Inter', sans-serif;
        font-weight: 400;
        font-size: 16px;
    }

```

```
        color:#000000;
        margin-top: 0px;
    }

    .systems-role {
        font-family: 'Inter', sans-serif;
        font-weight: 500;
        font-size: 20px;
        color: #000000;
        position: absolute;
        top: 22px;
        right: 16px;
    }
```

Приложение Д
(рекомендуемое)

Визуальные прототипы пользовательского интерфейса

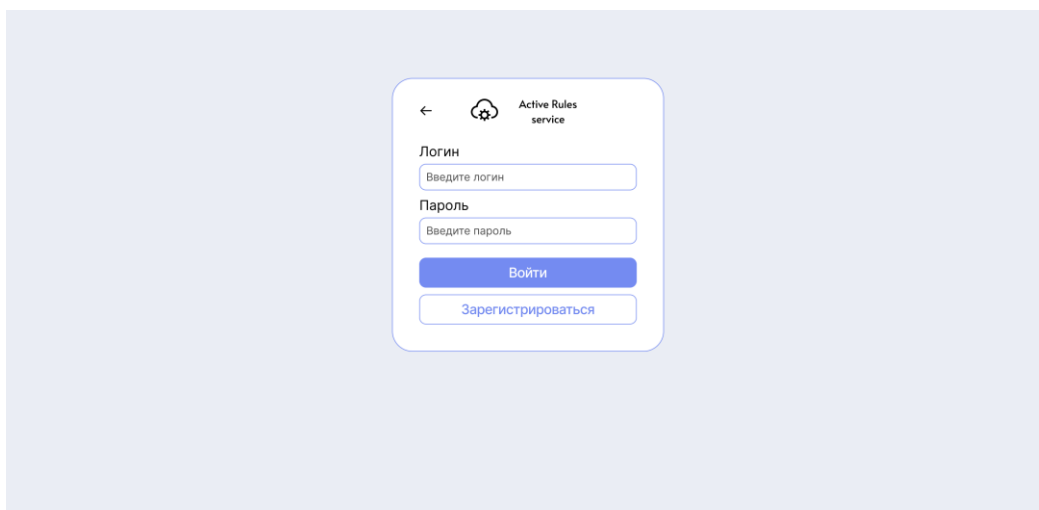


Рисунок Д.1 – Страница входа

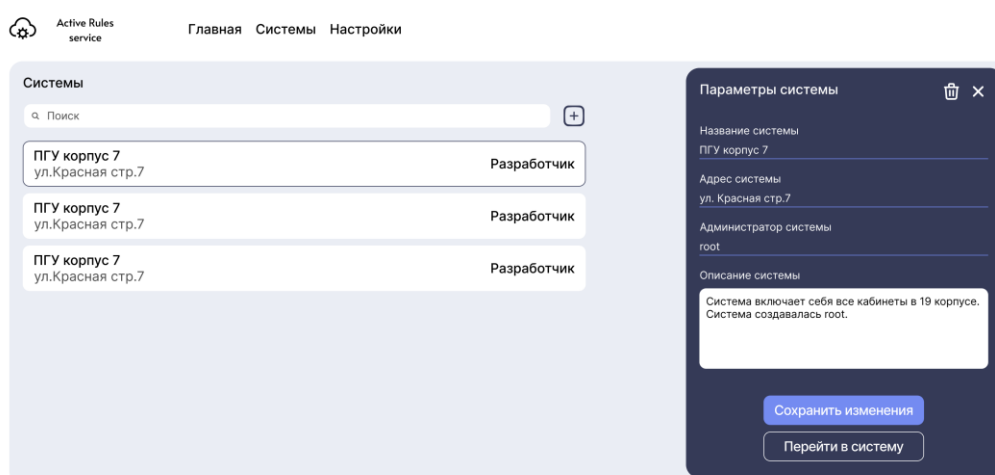


Рисунок Д.2 – Страница управления метаданными систем

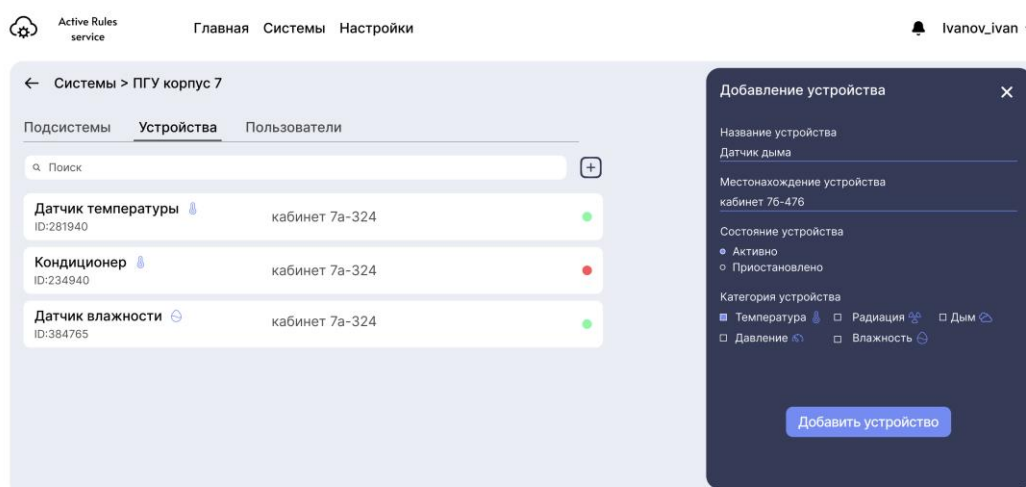


Рисунок Д.3 – Страница управления метаданными устройств

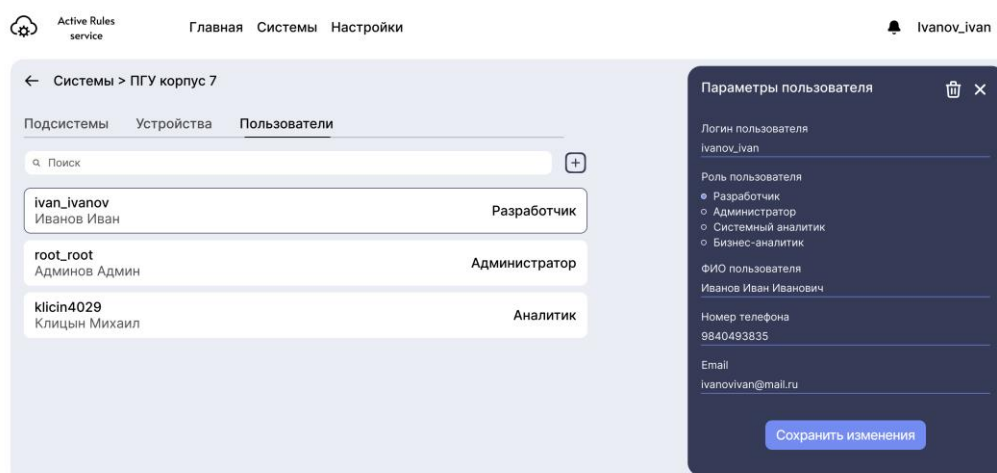


Рисунок Д.4 – Страница управления метаданными пользователей

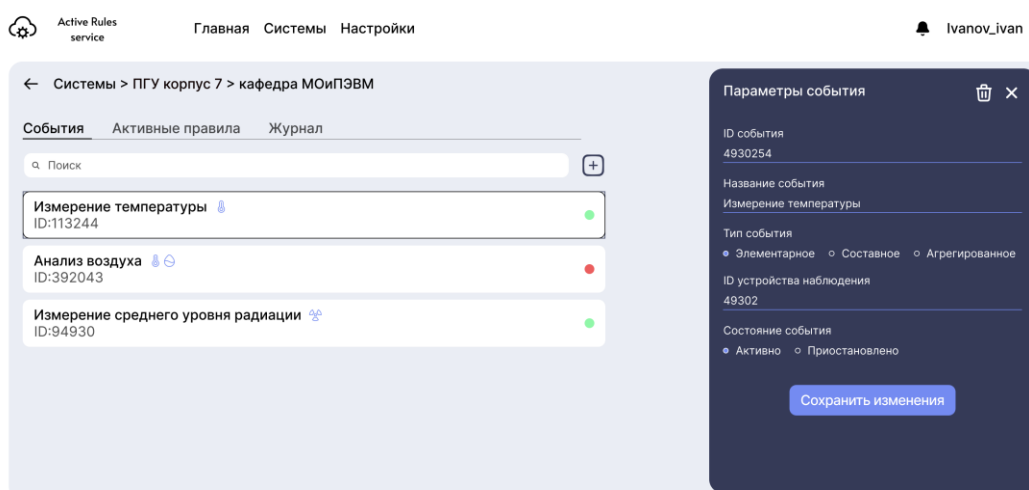


Рисунок Д.5 – Страница управления метаданными элементарного события

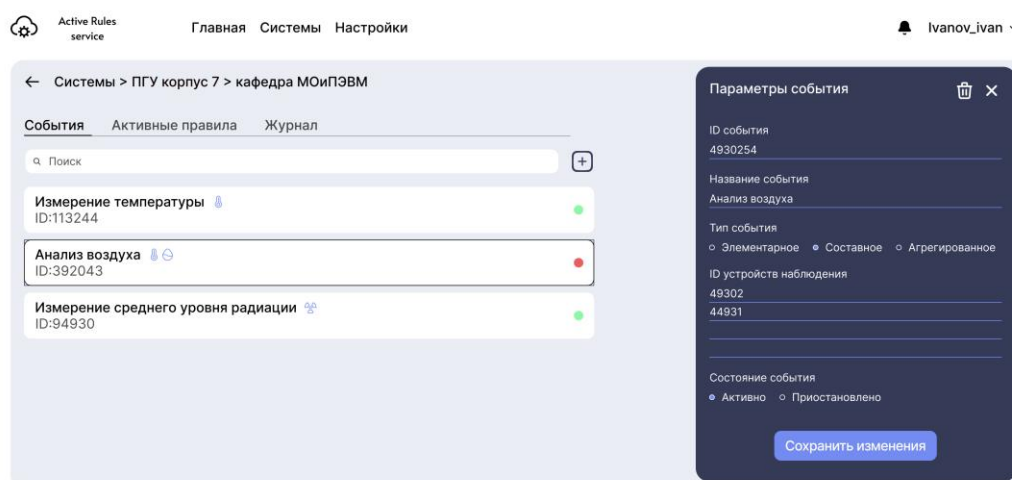


Рисунок Д.6 – Страница управления метаданными составного события

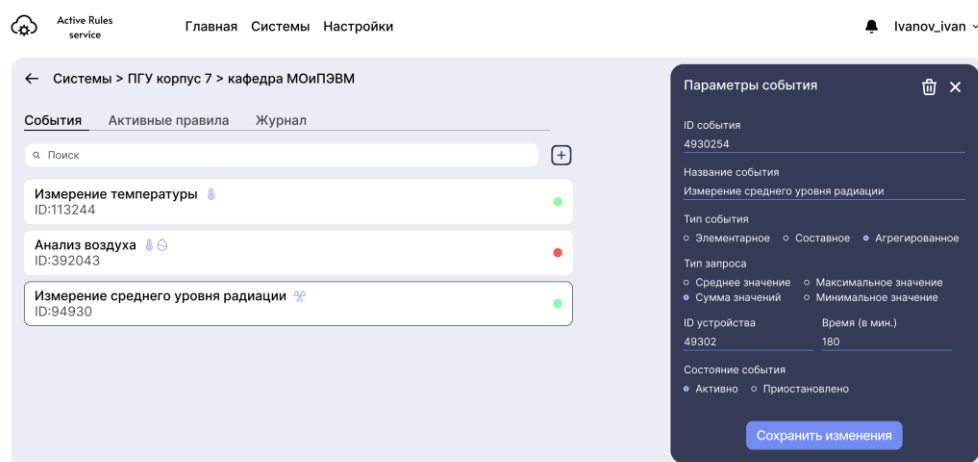


Рисунок Д.7 – Страница управления метаданными агрегированного события

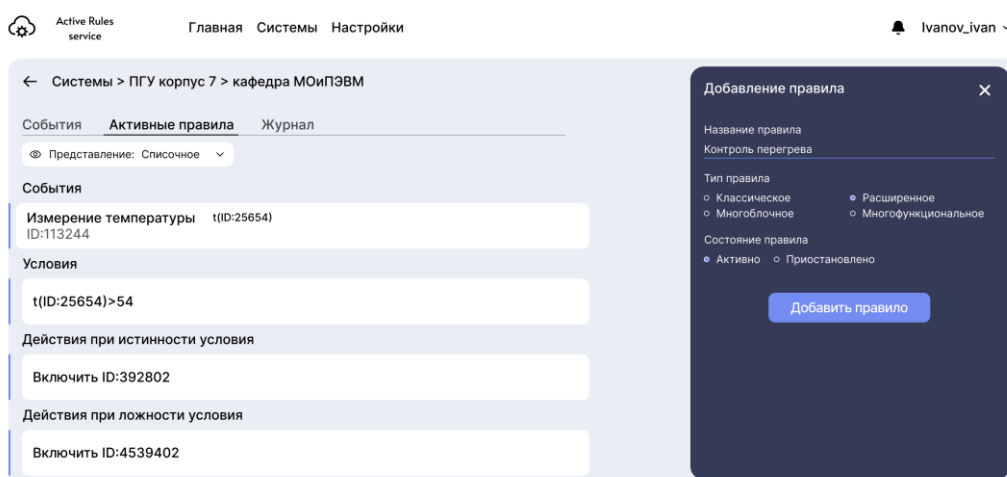


Рисунок Д.8 – Страница списочного представления активного правила

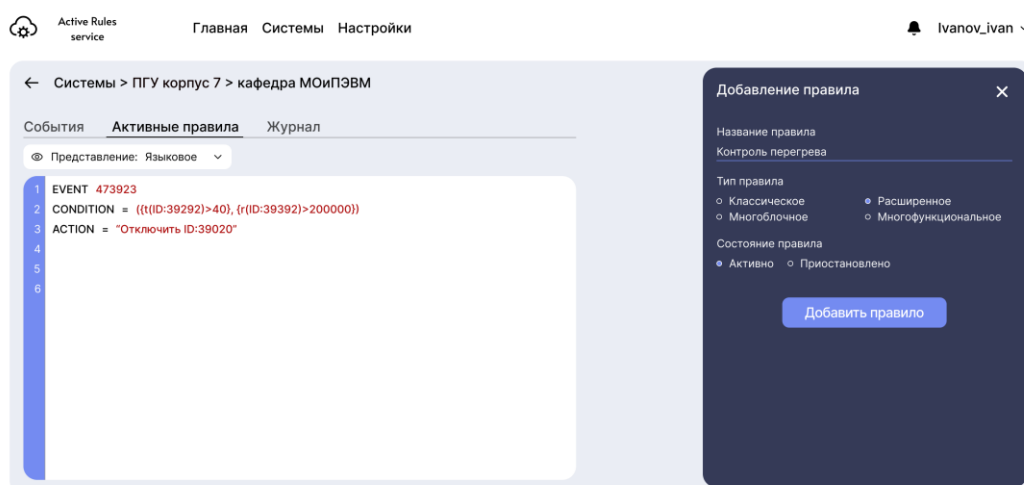


Рисунок Д.9 – Страница языкового представления активного правила

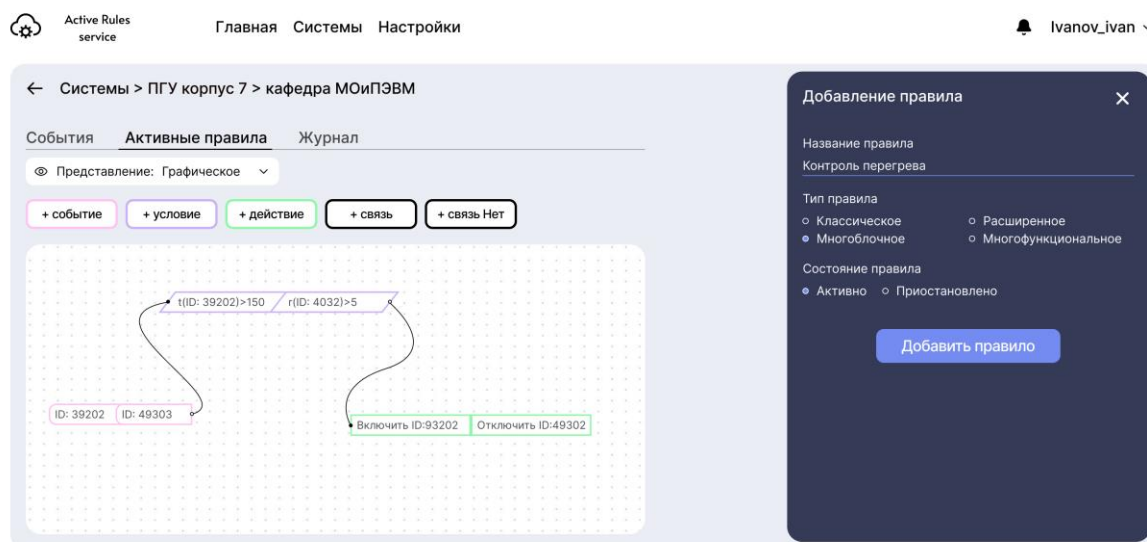


Рисунок Д.10 – Страница графического представления активного правила

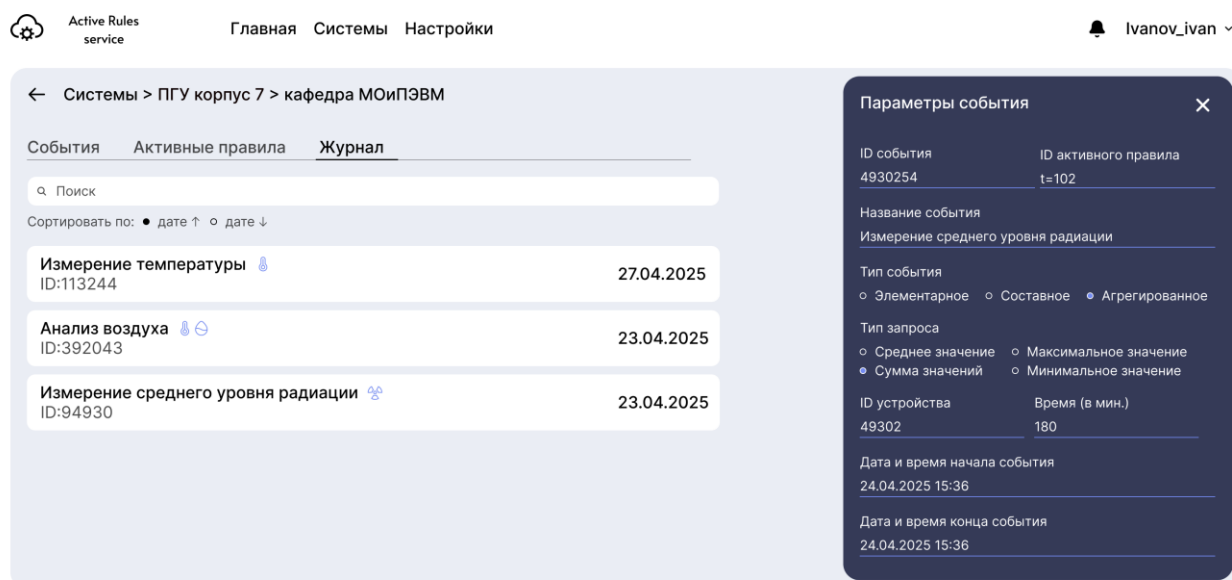


Рисунок Д.11 – Страница журнала событий

Приложение Е

(рекомендуемое)

Результаты функционального тестирования

Добавление события ✕

Название события Длина от 4 до 30, неверные символы
а%

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Должно быть от 2 устройств, не найдены в системе: 1234
1234

Введите ID устройства 2

Введите ID устройства 3

Введите ID устройства 4

Состояние события
☐ Активно ☒ Приостановлено

Добавить событие

Рисунок Е.1 – Функциональный тест 1

Добавление события ✕

Название события Допустимы буквы,цифры,пробел,_,-
абаба\$

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Должно быть от 2 устройств, не найдены в системе: 1234
1234

Введите ID устройства 2

Введите ID устройства 3

Введите ID устройства 4

Состояние события
☐ Активно ☒ Приостановлено

Добавить событие

Рисунок Е.2 – Функциональный тест 2

Добавление события ✕

Название события Длина от 4 до 30 символов.
аб

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Должно быть от 2 устройств, не найдены в системе: 1234
1234

Введите ID устройства 2

Введите ID устройства 3

Введите ID устройства 4

Состояние события
☒ Активно ☐ Приостановлено

Добавить событие

Рисунок Е.3 – Функциональный тест 3

Добавление события ✕

Название события Длина от 4 до 30, неверные символы
аб)

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Должно быть заполнено минимум два ID устройства
1

Введите ID устройства 2

Введите ID устройства 3

Введите ID устройства 4

Состояние события
☒ Активно ☐ Приостановлено

Добавить событие

Рисунок Е.4 – Функциональный тест 4

Добавление события ✕

Название события Длина от 4 до 30, неверные символы
аб*

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Устройства не найдены в системе: 1234
1
2
1234
Введите ID устройства 4

Состояние события
☒ Активно ☐ Приостановлено

Добавить событие

Рисунок Е.5 – Функциональный тест 5

Добавление события ✕

Название события
Измерение влаги

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Должно быть от 2 устройств, не найдены в системе: 1234
1234
Введите ID устройства 2
Введите ID устройства 3
Введите ID устройства 4

Состояние события
☒ Активно ☐ Приостановлено

Добавить событие

Рисунок Е.6 – Функциональный тест 6

Добавление события ✕

Название события
Измерение влаги

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Должно быть заполнено минимум два ID устройства
1

Введите ID устройства 2

Введите ID устройства 3

Введите ID устройства 4

Состояние события
☐ Активно ☐ Приостановлено

Добавить событие

Рисунок Е.7 – Функциональный тест 7

Добавление события ✕

Название события Длина от 4 до 30 символов.
Ипк

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Должно быть заполнено минимум два ID устройства
1

Введите ID устройства 2

Введите ID устройства 3

Введите ID устройства 4

Состояние события
☐ Активно ☐ Приостановлено

Добавить событие

Рисунок Е.8 – Функциональный тест 8

Добавление события ✕

Название события Длина от 4 до 30 символов.

Ипк

Тип события

- ☐ Элементарное
- ☒ Составное
- ☐ Агрегированное

ID устройств

1

2

Введите ID устройства 3

Введите ID устройства 4

Состояние события

- ☒ Активно
- ☐ Приостановлено

Добавить событие

Рисунок Е.9 – Функциональный тест 9

Добавление события ✕

Название события Длина от 4 до 30, неверные символы

a +=

Тип события

- ☐ Элементарное
- ☒ Составное
- ☐ Агрегированное

ID устройств

1

2

Введите ID устройства 3

Введите ID устройства 4

Состояние события

- ☒ Активно
- ☐ Приостановлено

Добавить событие

Рисунок Е.10 – Функциональный тест 10

Добавление события

×

Название события

Допустимы буквы,цифры,пробел,_,-

Измерение/ал

Тип события

●

 Элементарное

●

 Составное

●

 Агрегированное

ID устройств

Устройства не найдены в системе: 1234

1

2

1234

Введите ID устройства 4

Состояние события

●

 Активно

●

 Приостановлено

Добавить событие

Рисунок Е.11 – Функциональный тест 11

Добавление события

×

Название события

Допустимы буквы,цифры,пробел,_,-

Измерение/ал

Тип события

●

 Элементарное

●

 Составное

●

 Агрегированное

ID устройств

1

2

Введите ID устройства 3

Введите ID устройства 4

Состояние события

●

 Активно

●

 Приостановлено

Добавить событие

Рисунок Е.12 – Функциональный тест 12

Добавление события ✕

Название события
Измерение влаги

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Устройства не найдены в системе: 1234

1

2

1234

Введите ID устройства 4

Состояние события
☒ Активно ☐ Приостановлено

Добавить событие

Рисунок Е.13 – Функциональный тест 13

Добавление события ✕

Название события Допустимы буквы,цифры,пробел,_,-
rao@

Тип события
☐ Элементарное ☒ Составное ☐ Агрегированное

ID устройств Должно быть заполнено минимум два ID устройства

1

Введите ID устройства 2

Введите ID устройства 3

Введите ID устройства 4

Состояние события
☒ Активно ☐ Приостановлено

Добавить событие

Рисунок Е.14 – Функциональный тест 14

Добавление события

Название события Длина от 4 до 30 символов

Агр

Тип события

- Элементарное
- **Составное**
- Агрегированное

ID устройств Устройства не найдены в системе: 1234

1

2

1234

Введите ID устройства 4

Состояние события

- **Активно**
- Приостановлено

Добавить событие

Рисунок Е.15 – Функциональный тест 15

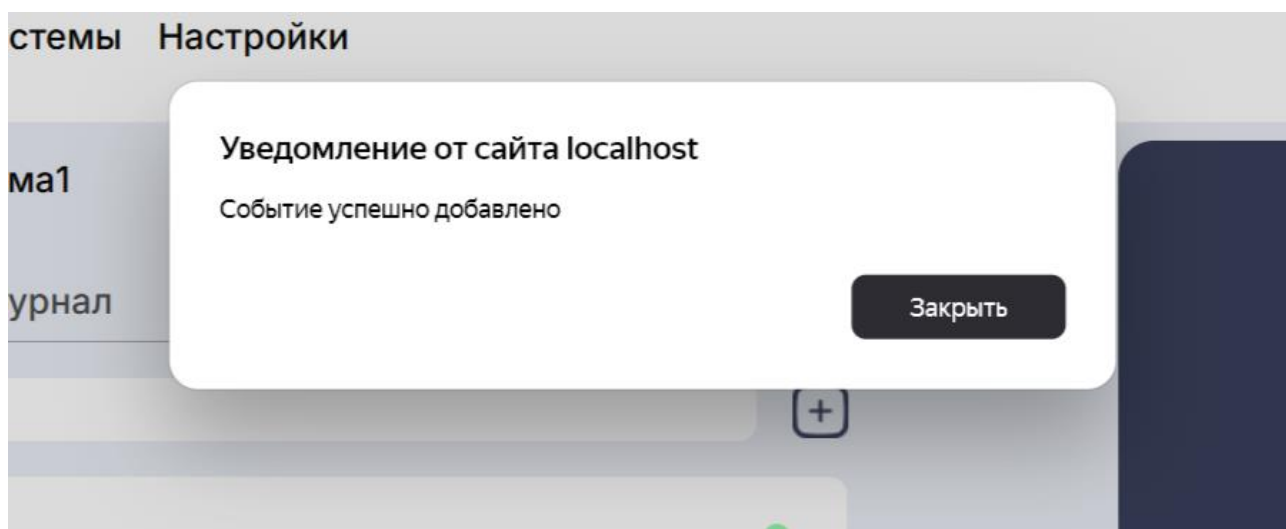


Рисунок Е.16 – Функциональный тест 16

Приложение Ж

(рекомендуемое)

Результаты конфигурационного тестирования

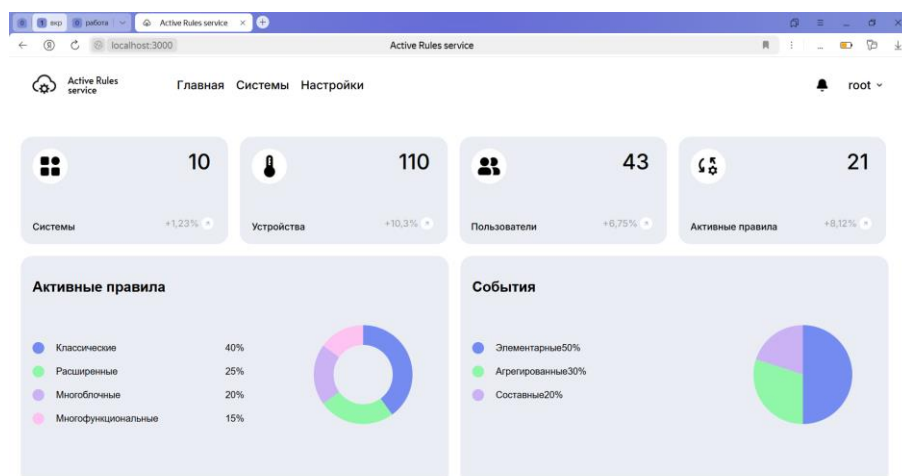


Рисунок Ж.1 – Конфигурационный тест 1

The screenshot shows the 'Active Rules service' user profile and roles configuration page. The user is logged in as 'root'. The page is divided into two main sections:

- Профиль**: A form for user profile information.
 - ФИО: Савинова Кристина
 - Номер телефона: 9601211202
 - Email: a1r@mao.ru
 - Логин: root
 - Пароль: (masked)
- Роли**: A list of roles assigned to the user.
 - Система 1 (Пугачева, 13): Главный администратор
 - Система 2 (Красная, 14): Главный администратор
 - Система 3 (Лермонтова, 12): Главный администратор

Buttons for 'Сохранить' (Save) and 'Отмена' (Cancel) are located at the bottom of the profile section.

Рисунок Ж.2 – Конфигурационный тест 2

The screenshot shows the 'Active Rules service' system configuration page. The user is logged in as 'root'. The page is divided into two main sections:

- Системы > Система 1**: A list of devices under the 'Система 1' system.
 - Устройство1 (ID: 1): этаж 1
 - Устройство2 (ID: 2): этаж 2
- Параметры устройства**: A modal window for configuring device parameters.
 - ID устройства: 1
 - Название устройства: Устройство1
 - Местонахождение устройства: этаж 1
 - Состояние подсистемы:
 - Активна
 - Приостановлена
 - Категория устройства:
 - Температура
 - Давление
 - Радиация
 - Влажность
 - Дым

A 'Сохранить изменения' (Save changes) button is located at the bottom of the modal.

Рисунок Ж.3 – Конфигурационный тест 3

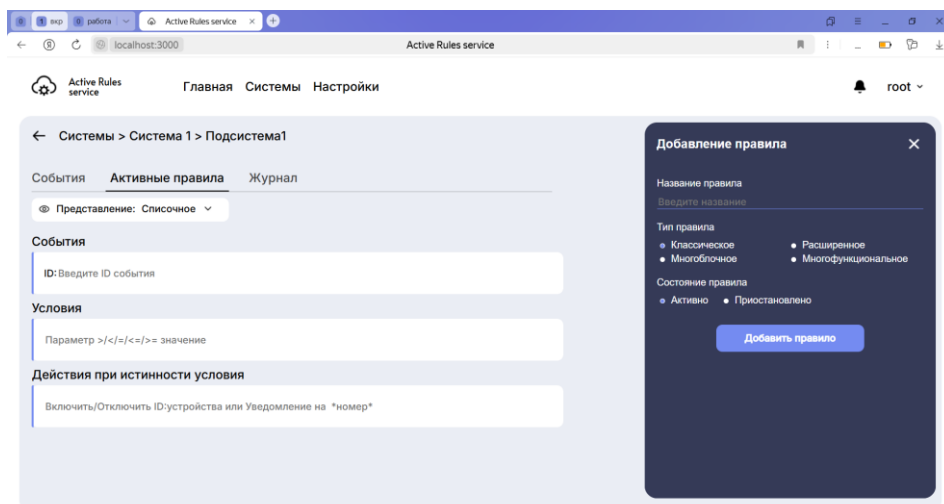


Рисунок Ж.4 – Конфигурационный тест 4

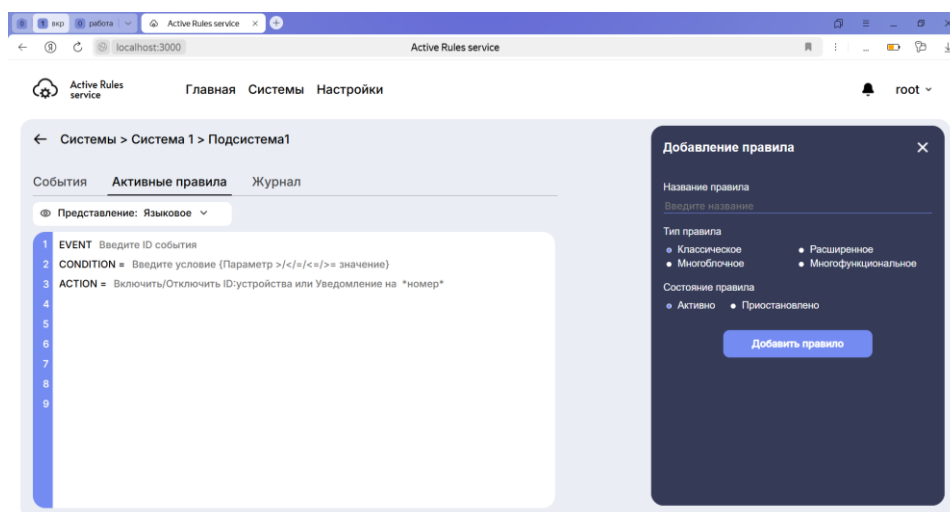


Рисунок Ж.5 – Конфигурационный тест 5

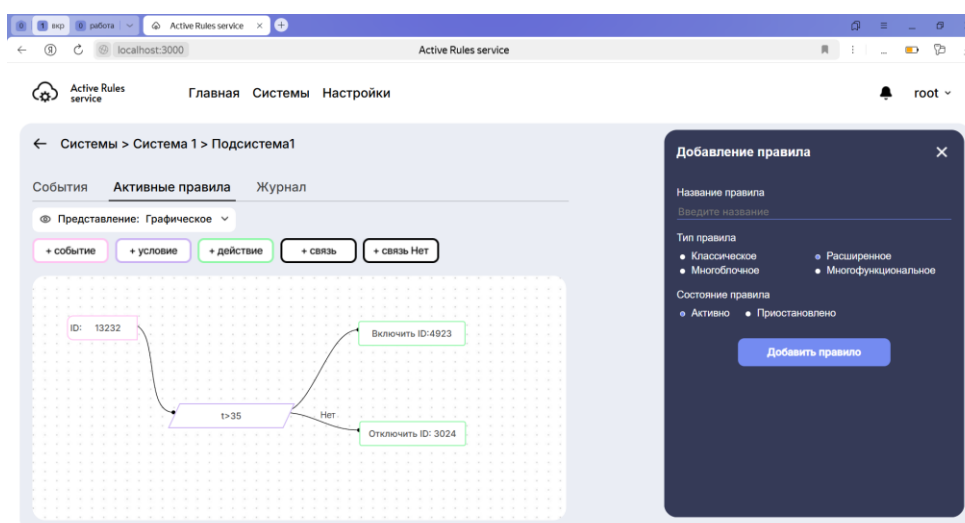


Рисунок Ж.6 – Конфигурационный тест 6

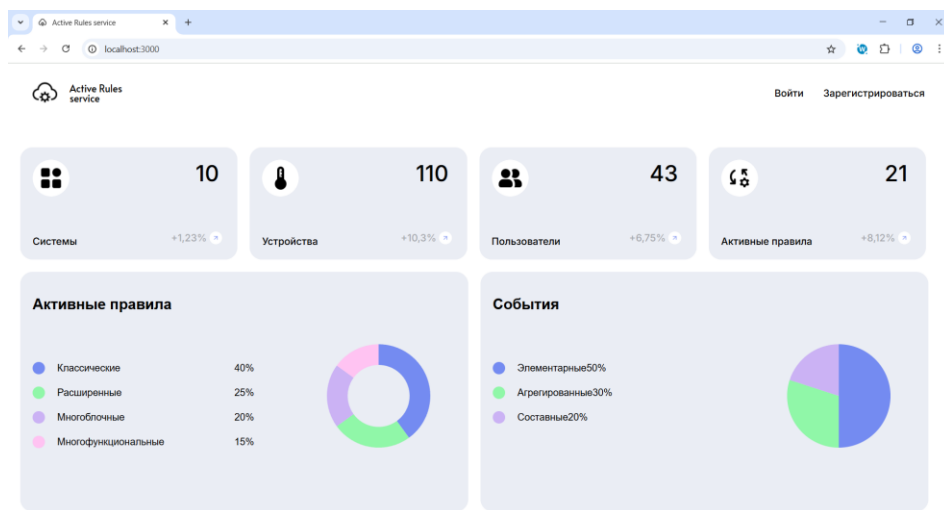


Рисунок Ж.7 – Конфигурационный тест 7

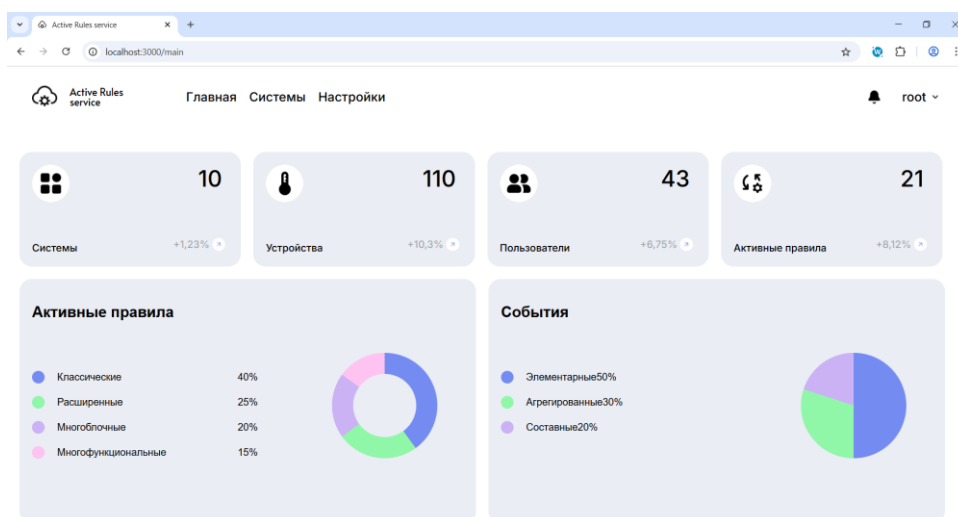


Рисунок Ж.8 – Конфигурационный тест 8

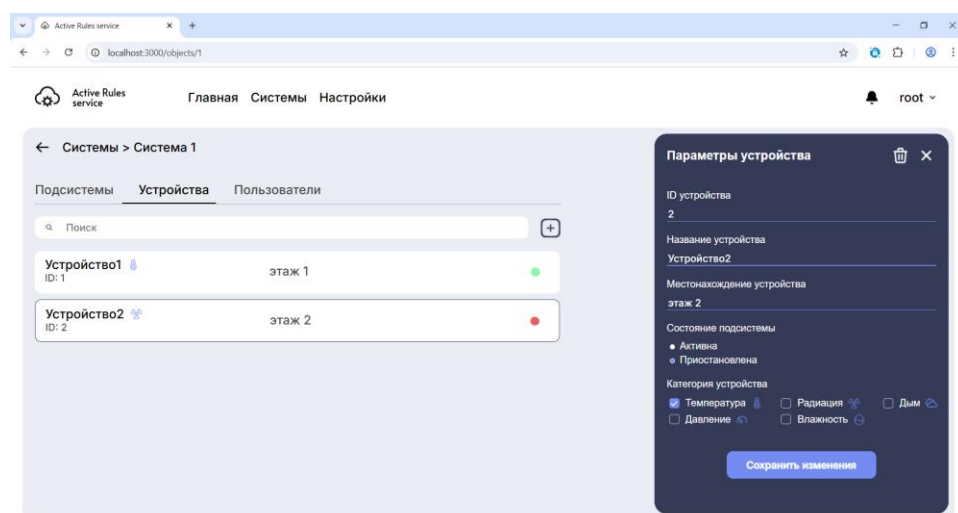


Рисунок Ж.9 – Конфигурационный тест 9

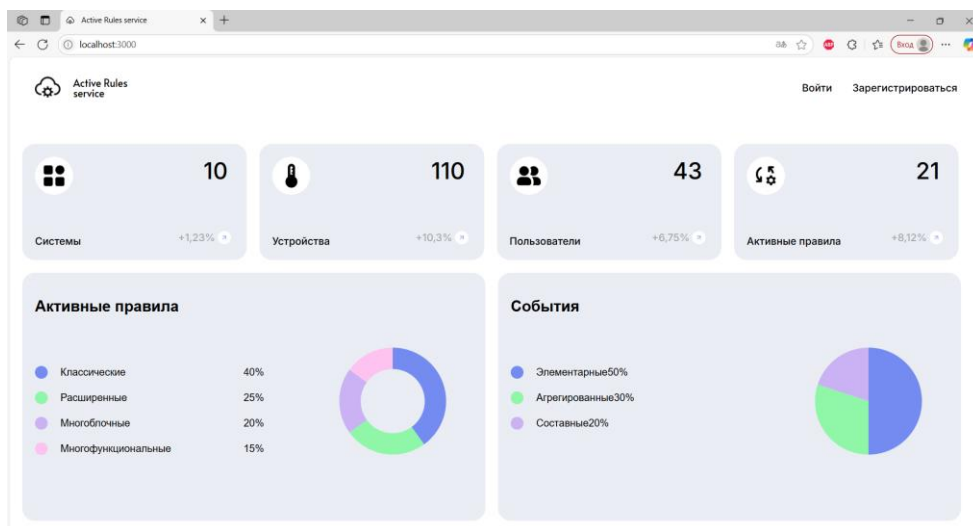


Рисунок Ж.13 – Конфигурационный тест 13

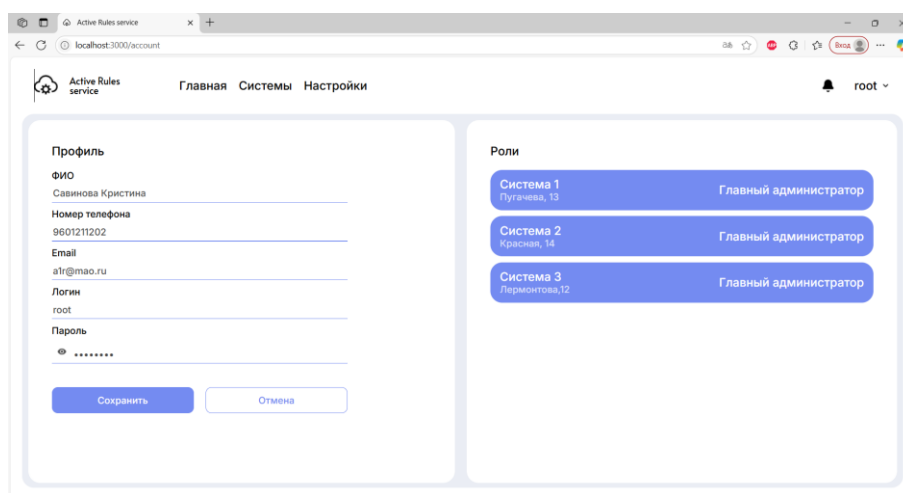


Рисунок Ж.14 – Конфигурационный тест 14

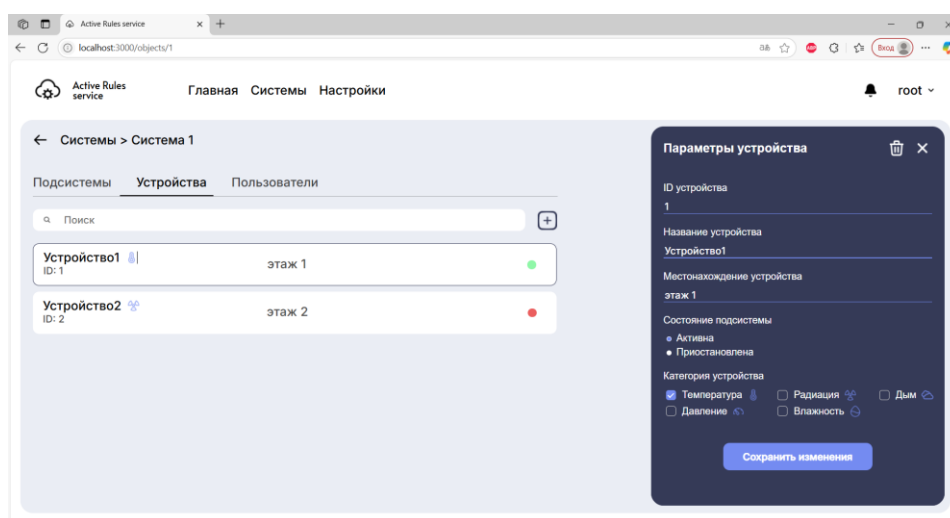


Рисунок Ж.15 – Конфигурационный тест 15

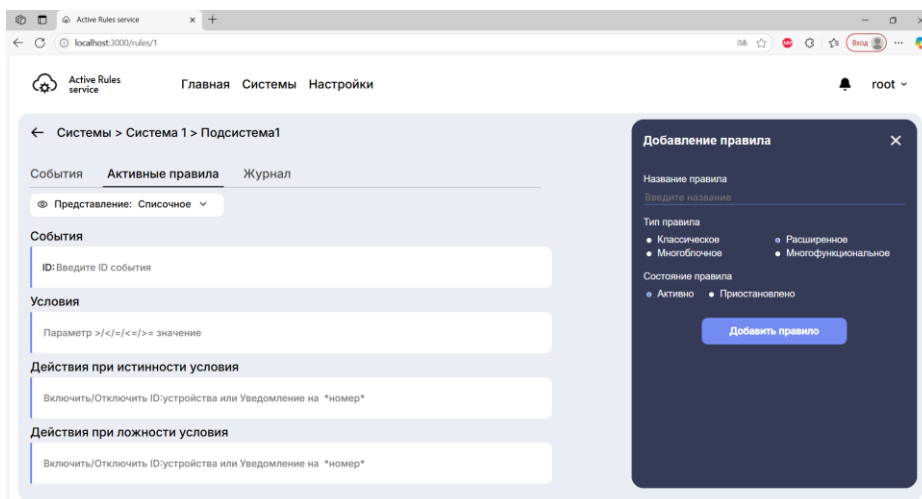


Рисунок Ж.16 – Конфигурационный тест 16

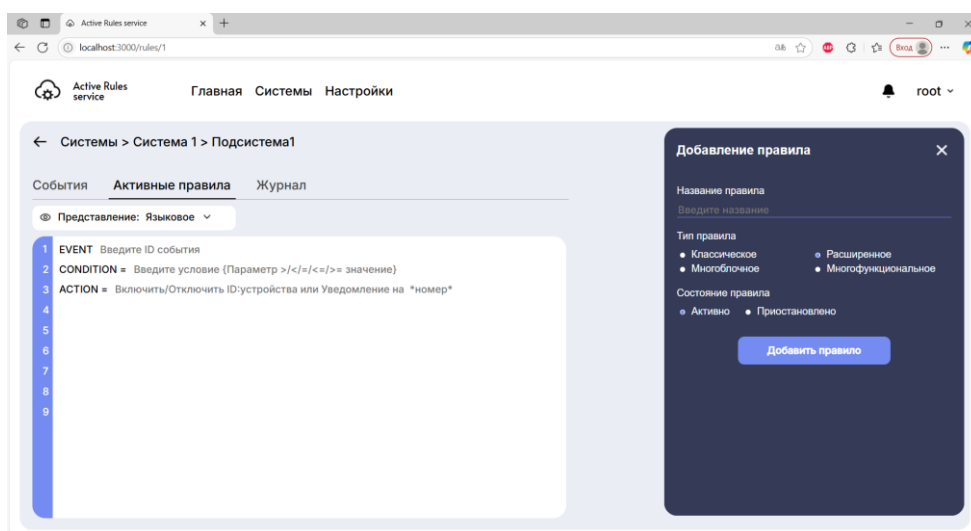


Рисунок Ж.17 – Конфигурационный тест 17

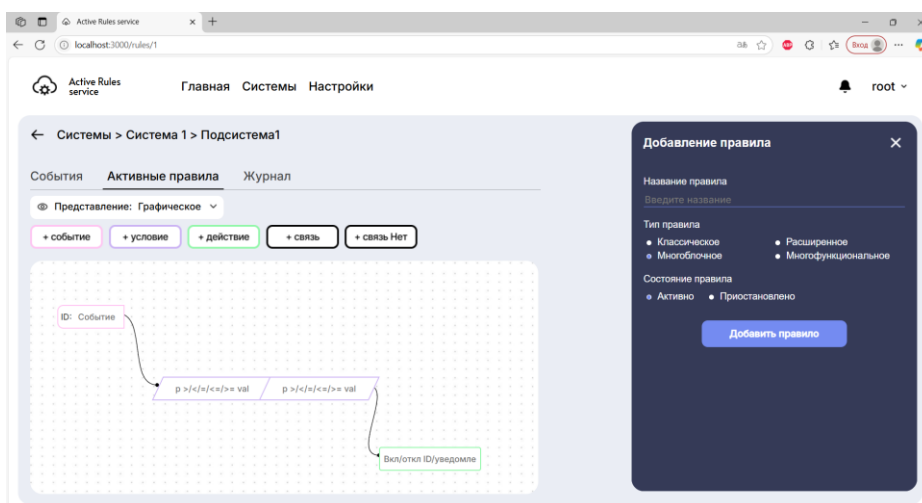


Рисунок Ж.18 – Конфигурационный тест 18