

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»

БАКАЛАВРСКАЯ РАБОТА

на тему

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

Студент	_____	Белошапко 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 _____ № _____)

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

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

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2023

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов «__»
_____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент _____ Белошапко Кристина Дмитриевна _____ гр. 19ВП1 факультета _____ ВТ
<Фамилия, имя, отчество полностью>

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

Тема утверждена приказом ПГУ № _____ от «__» _____ 2023 г.

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

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

1) Назначение: Программные средства поддержки языкового представления метаданных
для сервиса потоковой обработки событий

2) Среда выполнения: Windows 10

Языковые средства разработки: TypeScript, JavaScript

Инструментальные средства: WebStorm, npm, Swagger, Docker, StarUML.

Технологии разработки: ООП, сервисно-ориентированный подход.

3) Основные функции разрабатываемого программного обеспечения:

а) взаимодействие с репозиторием метаданных;

б) управление метаданными посредством инструмента свободного ввода языковых
конструкций;

в) управление метаданными посредством языкового конструктора;

г) анализ корректности языковых конструкций метаданных;

д) оповещение и доставка данных микросервисам;

е) взаимодействие с клиентскими приложениями;

ж) предотвращение несанкционированного доступа к механизмам управления;

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;
2) Разработка требований и проектирование;
3) Реализация;
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 требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022г.

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 129 листов, 46 рисунков, 10 таблиц, 14 использованных источников и 6 приложений.

ПОТОКОВАЯ ОБРАБОТКА СОБЫТИЙ, ЯЗЫКОВОЕ ПРЕДСТАВЛЕНИЕ МЕТАДАННЫХ, УПРАВЛЕНИЕ МЕТАДААННЫМИ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, СОБЫТИЙНО-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА, WEB-ПРИЛОЖЕНИЕ, UML, REST, MVVM.

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

Цель выпускной квалификационной работы – разработать программные средства поддержки языкового представления метаданных для сервиса потоковой обработки событий.

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

Среда исполнения – Windows 10, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы программные средства поддержки языкового представления метаданных. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08ВР191.03 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства поддержки языкового представления метаданных для сервиса потоковой обработки событий Пояснительная записка	Лит.	Лист	Листов
Разраб.		Белошапко К.Д.						
Пров.		Шибанов С.В.					4	129
Н. Контр.		Такташкин Д.В.				Группа 19ВП1		
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку.....	9
1.1 Сервис потоковой обработки событий и исполнения активных правил	9
1.1.1 Назначение и функциональные возможности сервиса потоковой обработки событий и исполнения активных правил.....	9
1.1.2 Архитектура сервиса потоковой обработки событий и исполнения активных правил.....	11
1.1.3 Схема работы сервиса потоковой обработки событий и исполнения активных правил.....	12
1.2 Метаданные сервиса потоковой обработки событий и исполнения активных правил.....	14
1.2.1 Структура метаданных сервиса.....	14
1.2.2 Жизненный цикл метаданных сервиса	15
1.2.3 Задачи управления метаданными сервиса.....	16
1.3 Постановка задачи на разработку программных средств.....	17
1.3.1 Цели разработки и назначение программных средств.....	17
1.3.2 Классы пользователей программных средств.....	17
1.3.3 Перечень автоматизируемых функций.....	18
1.4 Планирование и функционально-стоимостной анализ разработки программных средств	19
2 Анализ требований на разработку программных средств	28
2.1 Функциональные требования к программным средствам.....	28
2.2 Требования к архитектуре программных средств	31
2.3 Требования к пользовательскому интерфейсу программных средств..	33

2.4	Обоснование выбора технологий и средств разработки.....	34
2.5	Требования к аппаратно-программному обеспечению.....	36
3	Разработка программных средств	38
3.1	Разработка архитектуры развертывания программных средств языкового представления	38
3.2	Разработка языковых конструкций объектов метаданных.....	43
3.3	Разработка пользовательского интерфейса.....	46
3.4	Разработка программных средств управления метаданными с помощью языкового представления.....	54
3.4.1	Разработка программных средств редактора языковых конструкций объектов метаданных.....	54
3.4.2	Разработка анализаторов языковых конструкций объектов метаданных	58
4	Контроль качества программных средств.....	62
4.1	Планирование процесса контроля качества программных средств	62
4.1.1	Мероприятия по контролю качества программных средств	62
4.1.2	Обоснование выбора модели тестирования	63
4.1.3	Обоснование выбора метрик программного кода	64
4.2	Тестирование программных средств	65
4.2.1	Функциональное тестирование	65
4.2.2	Модульное тестирование	67
4.3	Оценка качества программного кода	70
4.3.1	Метрика Холстеда.....	70
4.3.2	Устойчивость и абстрактность	72
	Заключение	77
	Список использованных источников	79

Приложение А. Глоссарий	81
Приложение Б. Листинг редакторов языковых конструкций	83
Приложение В. Листинг анализаторов языковых конструкций	98
Приложение Г. Пример системы для тестирования приложения	110
Приложение Д. Результаты функционального тестирования	114
Приложение Е. Листинг модульных тестов	122

Введение

В настоящее время для решения различных задач разработано большое количество систем, работающих в режиме реального времени. Эти системы генерируют большие объемы данных, которые необходимо обрабатывать в кратчайшие сроки, а полученную информацию необходимо поддерживать в актуальном состоянии.

Множество компаний придерживается концепции быстрой разработки и низким порогом вхождения на рынок, то есть самыми важными аспектами являются высокая скорость разработки и низкая себестоимость продукта. Такие приложения, по большей части, состоят из основного приложения и хранилища данных. Однако такие монолитные системы порождают множество проблем с поддержанием продукта, его развитием, масштабированием, внедрением нового функционала и т. д. Вариантом решения таких проблем является разбитие монолитной структуры на части и построение системы на основе микросервисов.

Микросервисы представляют из себя небольшие, независимые приложения, которые отвечают за конкретные логические элементы основной большой системы. Сервис потоковой обработки данных состоит из нескольких микросервисов, которые общаются между собой через стандартизированные интерфейсы, например, через HTTP. При этом можно отметить, что такая система будет обладать рядом преимуществ: высокая доступность, простая масштабируемость, отказоустойчивость и вывод в реальном времени. При этом происходит баланс нагрузки на приложения, что повышает скорость работы и надежность системы.

Почти все данные, поступающие в системы такого вида, генерируются в виде непрерывных потоков событий. Для хранения и обработки этих потоков событий, выделения отдельных событий из потоков, поиска связей и закономерностей между ними, а также реагирования на возникновение определенных моделей событий разрабатываются приложения в архитектурах, управляемых событиями.

Для управления такими системами необходимы специализированные инструменты, нацеленные на разного рода специалистов, в том числе и разработчиков. Ежедневно такие сотрудники сталкиваются с языковыми конструкциями различных языков программирования, которые расширяют возможности управления множеством объектов с помощью редакторов кода. Поэтому появляется необходимость в разработке специальных инструментов управления метаданными, нацеленных не только на обычных администраторов продукта, но и на разработчиков.

Целью выпускной квалификационной работы является разработка программных средств поддержки языкового представления метаданных для сервиса потоковой обработки событий. Работа включает в себя анализ требований, разработку требований, реализацию и контроль качества программного обеспечения.

1 Анализ предметной области и постановка задачи на разработку

1.1 Сервис потоковой обработки событий и исполнения активных правил

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

Потоковая обработка данных необходима для автоматизации и поддержки принятия решения реального или близкого к реальному времени.

Сервис потоковой обработки событий и исполнения активных правил представляет систему для обработки потока событий, которые поступают от наблюдаемой системы или выявленных по шаблону событий. Также данный сервис подразумевает использование активных правил для воздействия на управляемые системы.

За основу сервиса взято правило ECA (Event Condition Action) [1], которое состоит из трех компонентов:

- событие, которое инициирует вызов правила, при этом оно может быть композицией, в этом случае событие является составным;
- условие, которое необходимо для выполнения действия (проверяется только при наступлении события);
- действие, которое необходимо выполнить над данными.

Эта технология применима к приложениям, которые используются для контроля и управления техническими объектами. Так, например, на производстве при условии неисправности станок прекратит работу, а диспетчеру поступит соответствующее сообщение, медицинских учреждениях приложения с данной технологией облегчают отслеживание состояния здоровья пациентов. Одним из современных примером применения технологии является приложение «Умный дом», с помощью которого пользователь может как контролировать состояние объектов, так и управлять ими, например, включить чайник.

Во всех вышеописанных случаях сервис можно представить, как систему, на вход которой подается событие с условиями и на выходе получается соответствующая реакция.

На рисунке 1 представлена диаграмма вариантов использования сервиса потоковой обработки событий [2].

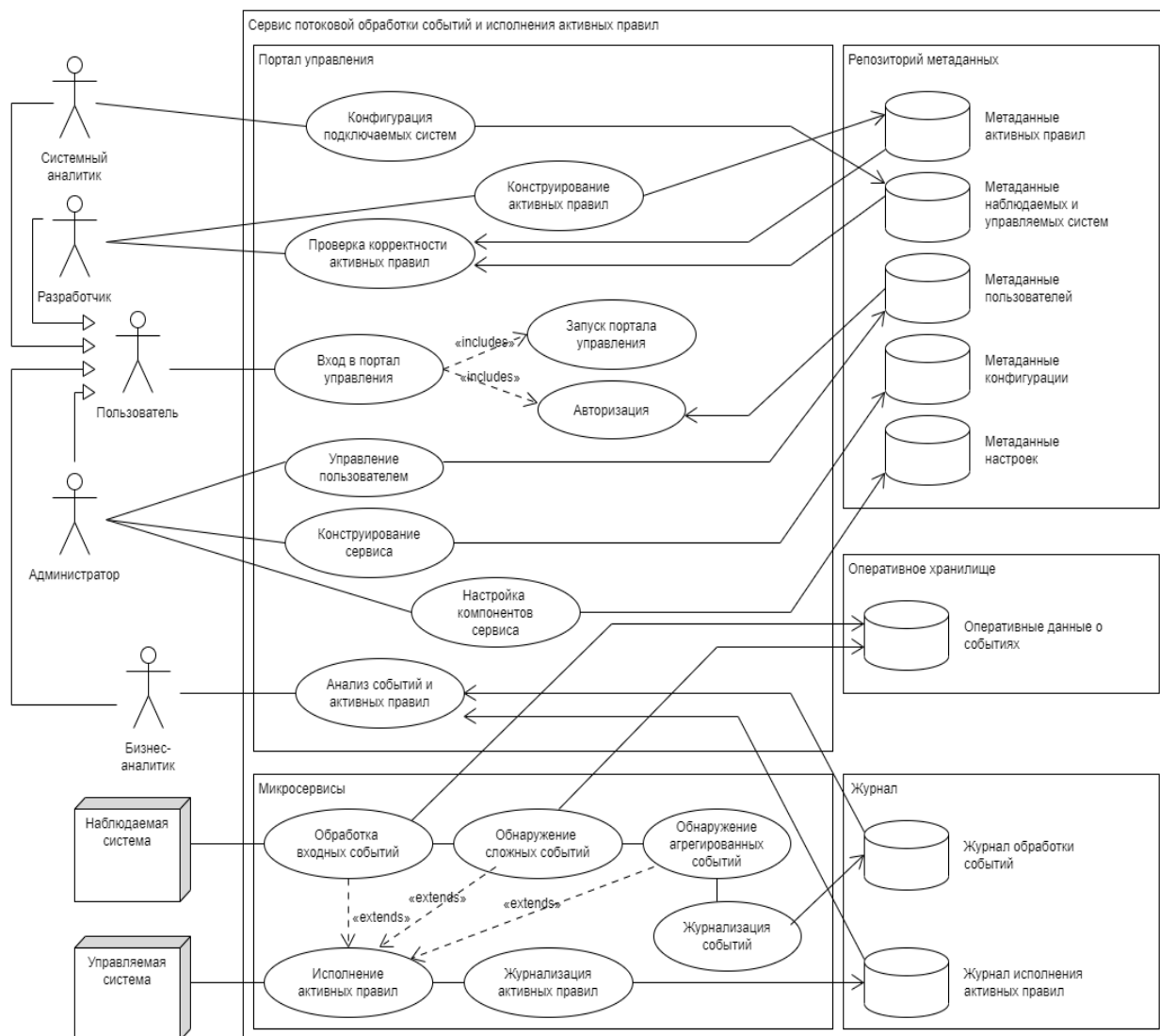


Рисунок 1 – Функциональные возможности сервиса потоковой обработки событий

Основные функции сервиса:

- создание активных правил с помощью специального портала управления в различных представлениях;
- управление активными правилами, в том числе их модификация и настройка;

- управление метаданными различных типов событий – их создание, модификация, настройка в различных представлениях сервиса;
- обработка потока событий, полученных от наблюдаемых систем, установление событий по заданным шаблонам и доведение их до соответствующих компонентов сервиса;
- исполнение активных правил – выбор образца активного правила для исполнения, вычисление условия экземпляра активного правила или группы правил, решение о выполнении действия в рамках одного из активных правил и применение действий к исполняемому объекту;
- журнализация обработки событий и исполнения активных правил для последующего анализа журналов событий, выявления проблем и построения событий и активных правил на основе предыдущего анализа;
- настройка и развитие сервиса – изменение параметров работы отдельных компонентов с учетом данных аудита, изменения параметров, распределения потоков обработки и сервиса в целом, а также подключение или отключение отдельных компонентов.

По мере развития сервиса его функциональные возможности могут расширяться или быть скорректированы.

1.1.2 Архитектура сервиса потоковой обработки событий и исполнения активных правил

Для построения архитектуры сервиса потоковой обработки событий и исполнения активных правил использованы микросервисы и событийно-ориентированный подход. Благодаря микросервисной архитектуре приложение можно разделить на отдельные части или модули. Такой подход позволяет легко поддерживать каждый модуль, их проще обновлять, масштабировать и разворачивать. Также каждый модуль может быть реализован на разных языках программирования и с применением различных фреймворков и библиотек, выполняться в разных средах и с помощью различного программного обеспечения. Это говорит о независимости модулей друг от друга.

Для взаимодействия серверной и клиентской частей системы, для управления метаданными используется REST-сервер. Это программное средство необходимо для взаимодействия компонентов, обмена данными и упрощенного масштабирования.

Входной поток событий поступает в сервис от источников (наблюдаемых систем), происходит их обработка и, применяются действия к системам управления, если имеются соответствующие активные правила.

Для хранения метаданных необходим центральный репозиторий. Набор метаданных может состоять из описания событий, активных правил и систем управления, информацию об объектах наблюдения и их объектах, а также результаты анализа событий и управляющих воздействий [3].

1.1.3 Схема работы сервиса потоковой обработки событий и исполнения активных правил

Обработка потоковых событий происходит в режиме реального времени, если не непрерывно. Это означает, что обнаружение событий и реагирование на них должно происходить в очень короткие сроки, чтобы поддерживать непрерывное и своевременное взаимодействие с окружением. Сервис поддерживает эти характеристики с помощью механизмов активных правил и промежуточных служб передачи данных.

На рисунке 2 представлена диаграмма деятельности сервиса потоковой обработки событий и исполнения активных правил, описывающая схему работы рассматриваемой системы.

Микросервисы системы обрабатывают поток событий из контролируемой системы, используя созданные пользователем метаданные. Они обрабатывают полученные события, определяют последовательное возникновение шаблонов событий, идентифицирует и выполняют активные правила. Статистика о работе сервиса журналируется и становятся доступными для специалистов аналитики.

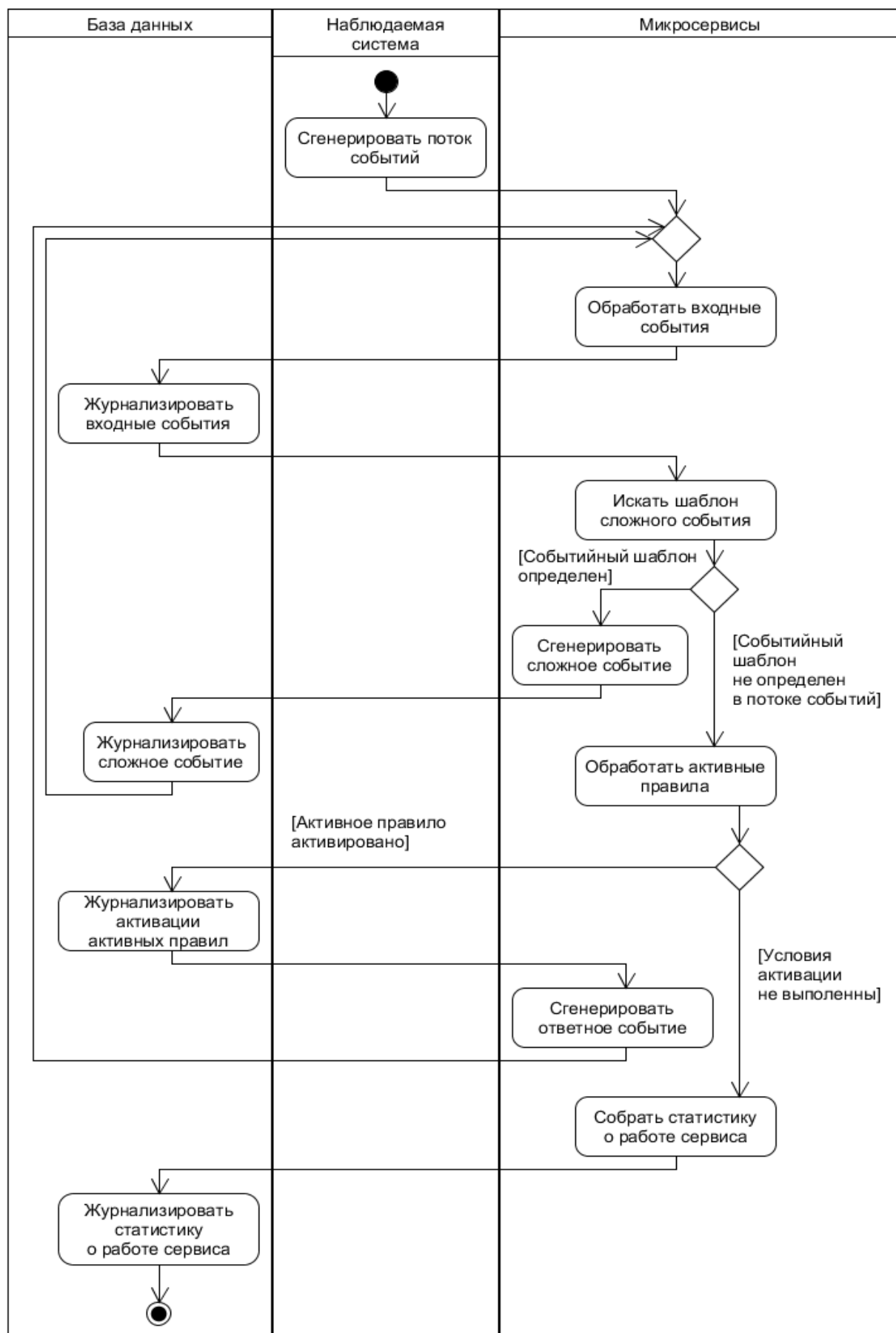


Рисунок 2 – Диаграмма деятельности сервиса потоковой обработки событий

Потоковая обработка событий дает уверенность в незамедлительной реакции на predetermined ситуации, такие как отказ оборудования или пожарная опасность.

Внедряя этот тип технологии, компании могут значительно снизить затраты, позволяя проводить обслуживание оборудования «по мере необходимости», а не по жесткому графику.

1.2 Метаданные сервиса потоковой обработки событий и исполнения активных правил

1.2.1 Структура метаданных сервиса

Метаданные, описывающие все хранимые и обрабатываемые объекты сервиса потоковой обработки событий, выполняют ведущую роль в работе системы.

Различаются три типа метаданных сервиса:

- метаданные устройств;
- метаданные систем;
- метаданные событий.

Метаданные определенной системы, которая управляет жизненным циклом и автоматизирует процессы, могут быть, например, общей структурой реакций на модели поведения или правами доступа пользователя.

Каждая система содержит ряд компонентов или устройств, которые являются источниками, из которых генерируются события. Метаданными устройств могут быть, например, данные статистики.

Метаданные событий определяют события, поступающие извне контролируемых устройств, или различные типы событий, происходящих в пользовательской среде. Различают следующие типы метаданных событий:

- метаданные агрегированных событий;
- метаданные сложных или комплексных событий;
- метаданные элементарных или простых событий.

Агрегирующие события характеризуются шаблонами агрегации, а сложные события характеризуются списком других событий, частью которых они являются. Метаданные простых событий определяют атомарные события от определенного источника с одним определенным параметром.

Также, существуют метаданные о пользователях, работающих в системе. Большой частью метаданных могут управлять пользователи с соответствующей ролью в сервисе.

1.2.2 Жизненный цикл метаданных сервиса

Жизненный цикл метаданных представляет из себя полную последовательность этапов, которую проходят метаданные от создания до удаления. Рассмотрим основные из этих этапов (рисунок 3):

- 1) создание или модификация метаданных;
- 2) обслуживание метаданных;
- 3) использование метаданных;
- 4) журнализация метаданных;
- 5) уничтожение метаданных.

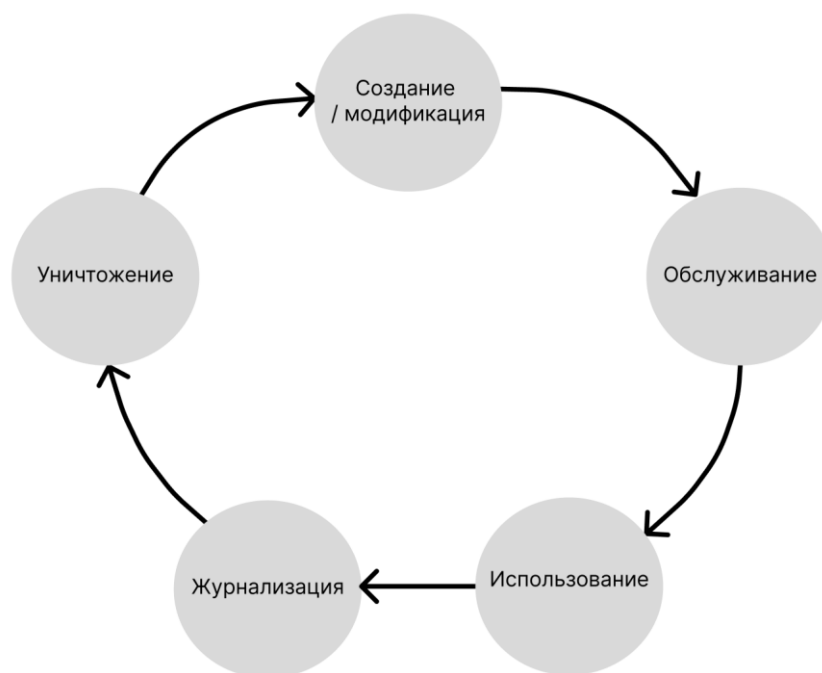


Рисунок 3 – Жизненный цикл метаданных сервиса

Этап создания или модификации метаданных включает в себя получение объектов метаданных, проверку ограничений и запись их в репозиторий.

Этап обслуживания состоит из анализа коллизий на взаимодействие с другими объектами метаданных, их устранение при необходимости и преобразование для использования метаданных.

Использование метаданных заключается в доставке набора метаданных объектам сервиса.

Далее, метаданные журналируются для последующего сбора статистики. В завершении метаданные уничтожаются посредством отключения от использования и удаления из репозитория.

1.2.3 Задачи управления метаданными сервиса

Взаимодействие и управление метаданными осуществляется посредством портала управления сервисом. В зависимости от роли у каждого пользователя поставлены свои задачи управления метаданными.

Основная задача разработчика состоит в конструировании и управлении объектами метаданных с помощью их различных представлений в портале управления. Для работы с языковым представлением объектов метаданных разработчик должен знать параметры каждого из объектов и их типов, а также уметь четко и грамотно составлять языковые конструкции объектов.

Кроме основных задача управления метаданными таких как просмотр, создание, изменение и удаление метаданных, определена задача управления метаданными авторизации пользователя. Например, бизнес-аналитик не должен иметь доступ к изменению и созданию событий и активных правил.

После создания или модификации метаданных происходит проверка сохраняемых данных на корректность. Например, невозможно сохранить активное правило без заполнения обязательных полей данных, это приведет к их неверному исполнению.

1.3 Постановка задачи на разработку программных средств

1.3.1 Цели разработки и назначение программных средств

Разрабатываемые программные средства по управлению метаданными должны обеспечивать пользователя набором удобных инструментов для просмотра и управления как метаданными, так и сервисом в целом.

Основные цели разработки:

- работа с метаданными авторизации пользователей;
- создание инструмента свободного ввода кода объектов метаданных для создания и редактирования посредством языкового представления;
- создание конструктора кода объектов метаданных для создания и редактирования посредством языкового представления;
- создание анализатора кода объекта метаданных для выявления синтаксических и структурных ошибок.

Кроме того, после окончания основного процесса разработки и публикации сервиса потоковой обработки событий, требуется поддержать его функционал, проверять на неисправности, проводить анализ работы сервиса с помощью статистики, а также внедрять новый функционал при необходимости для упрощения работы в сервисе.

1.3.2 Классы пользователей программных средств

Существует четыре основных класса пользователей сервиса:

- администратор;
- системный аналитик;
- разработчик;
- бизнес-аналитик.

В зависимости от роли каждый тип пользователей обладает своими функциональными возможностями в системе.

Задачи администратора сервиса состоят в создании окружения и управлении конфигурациями сервиса, регистрации пользователей различных ролей и предоставлении им доступа к управлению метаданными.

Системные аналитики подключают и настраивают системы, определяют объекты управления, группируют их и анализируют взаимодействие различных объектов систем.

В задачи разработчика входит контроль и управление метаданными событий и активных правил.

Бизнес-аналитики изучают собранную статистику о производительности сервиса в целом или в частности о конкретных наборах событий.

Данные программные средства реализуют функционал, который нацелен на разработчика.

1.3.3 Перечень автоматизируемых функций

Программные средства потоковой обработки событий и активных правил должны обладать следующими автоматизируемыми функциями:

- автоматический сбор данных полей ввода и упаковка в JSON для отправки на сервер;
- предоставление инструментов управления языкового представления, доступных пользователю согласно его правам доступа в сервисе;
- расшифровка данных, пришедших с сервера, и автоматическая вставка данных в поля ввода при необходимости;
- автоматическая проверка корректности введенных пользователем данных, в том числе проверка на пустые поля и проверку структуры метаданных;
- автоматическая вставка токена авторизации при отправке любого запроса на сервер.

Кроме того, программные средства должны обеспечивать автоматическую работу с доставленными данными от сервера на клиентскую часть, например, сортировку, фильтрацию, сохранение в локальное или сессионное хранилище.

1.4 Планирование и функционально-стоимостной анализ разработки программных средств

Одной из первоначальных задач процесса разработки является планирование. От качественного планирования зависит не только то, будет ли выполнена работа в срок, но и качество разработанного приложения. Основной целью планирования является составление графика выполнения работ по проекту и приблизительный расчет затрат по проекту.

В таблице 1 представлены задачи проекта. Для графика была выбрана шестидневная рабочая неделя.

Таблица 1 – Список задач проекта

№	Задача	Длительность	Предшественник
1	Начало работы над проектом	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Анализ функциональных возможностей сервиса	2 дня	1
4	Анализ архитектуры и схемы работы сервиса	2 дня	3
5	Анализ структуры и жизненного цикла метаданных	1 день	4
6	Анализ задач управления метаданными	2 дня	5
7	Составление перечня автоматизируемых функций	1 день	6
8	Планирование и функционально-стоимостный анализ процесса разработки	2 дня	7
9	Анализ предметной области и постановка задач завершены	-	8
10	Анализ требований на разработку программных средств	-	2
11	Разработка функциональных требований к программным средствам	2 дня	9
12	Разработка требований к архитектуре программных средств	2 дня	11

Продолжение таблицы 1

№	Задача	Длительность	Предшественник
13	Постановка требований к интерфейсу программных средств	2 дня	12
14	Выбор технологий разработки	1 день	13
15	Постановка требований к аппаратно-программному обеспечению	1 день	14
16	Анализ требований на разработку программных средств завершен	-	15
17	Разработка программных средств	-	10
18	Разработка пользовательского интерфейса	5 дней	16
19	Разработка программных средств взаимодействия с серверной частью сервиса	2 дня	18
20	Разработка программных средств языкового представления в режиме свободного ввода	5 дней	19
21	Разработка программных средств языкового представления в режиме конструктора	5 дней	20
22	Разработка программных средств проверки корректности метаданных	4 дня	21
23	Разработка программных средств завершена	-	22
24	Контроль качества программных средств	-	17
25	Планирование процесса контроля качества программных средств	3 дня	23
26	Тестирование программных средств	3 дня	25
27	Оценка качества программного кода	2 дня	26
28	Контроль качества программных средств завершен	-	27
29	Сдача проекта	-	24
30	Оформление и сбор документации	4 дня	28
31	Ввод в эксплуатацию	1 день	30
32	Сдача проекта окончена	-	31
33	Конец работы над проектом	-	32

Далее необходимо определить используемые ресурсы и распределить их по каждой из задач. В таблице 2 представлен перечень ресурсов проекта.

Таблица 2 – Перечень ресурсов проекта

Название ресурса	Тип ресурса	Стандартная ставка	Сверхурочная ставка
Аналитик	Трудовой	200 Руб/ч	300 Руб/ч
Программист	Трудовой	250 Руб/ч	350 Руб/ч
Дизайнер	Трудовой	170 Руб/ч	300 Руб/ч
Тестировщик	Трудовой	170 Руб/ч	300 Руб/ч
Интернет	Затраты	-	-
Компьютер	Материальный	35 000 Руб	-

Теперь, имея перечень ресурсов проекта, необходимо распределить их по задачам проекта. Данная информация представлена в таблице 3.

Таблица 3 – Распределение ресурсов проекта по задачам

№	Задача	Ресурсы	Единицы
1	Начало работы над проектом	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Анализ функциональных возможностей сервиса	Аналитик	100%
4	Анализ архитектуры и схемы работы сервиса	Аналитик	100%
5	Анализ структуры и жизненного цикла метаданных	Аналитик	100%
6	Анализ задач управления метаданными	Аналитик	100%
7	Составление перечня автоматизируемых функций	Аналитик	100%
8	Планирование и функционально-стоимостный анализ процесса разработки	Аналитик	100%
9	Анализ предметной области и постановка задач завершены	-	-
10	Анализ требований на разработку программных средств	-	-
11	Разработка функциональных требований к программным средствам	Программист	100%

Продолжение таблицы 3

12	Разработка требований к архитектуре программных средств	Программист	100%
13	Постановка требований к интерфейсу программных средств	Программист	100%
14	Выбор технологий разработки	Программист	100%
15	Постановка требований к аппаратно-программному обеспечению	Программист	100%
16	Анализ требований на разработку программных средств завершен	-	-
17	Разработка программных средств	-	-
18	Разработка пользовательского интерфейса	Дизайнер	100%
		Программист	50%
		Интернет	200 Руб
19	Разработка программных средств взаимодействия с серверной частью сервиса	Программист	100%
		Интернет	150 Руб
20	Разработка программных средств языкового представления в режиме свободного ввода	Программист	100%
		Интернет	450 Руб
21	Разработка программных средств языкового представления в режиме конструктора	Программист	100%
		Интернет	400 Руб
22	Разработка программных средств проверки корректности метаданных	Программист	100%
		Интернет	200 Руб
23	Разработка программных средств завершена	-	-
24	Контроль качества программных средств	-	-
25	Планирование процесса контроля качества программных средств	Тестировщик	100%
26	Тестирование программных средств	Тестировщик	100%
		Интернет	500 Руб
27	Оценка качества программного кода	Тестировщик	100%
28	Контроль качества программных средств завершен	-	-
29	Сдача проекта	-	-
30	Оформление и сбор документации	Программист	100%

Окончание таблицы 3

31	Ввод в эксплуатацию	Программист	100%
		Интернет	300 Руб
32	Сдача проекта окончена	-	-
33	Конец работы над проектом	-	-

Для определения длительности работ, составлении графика задач и расчета общих затрат по проекту было решено использовать приложение для управления программными продуктами Microsoft Office Project.

Microsoft Project – программа управления программными продуктами разработки Microsoft, поставляемая как самостоятельный продукт, так и в составе дополнительных редакций пакета Microsoft Office.

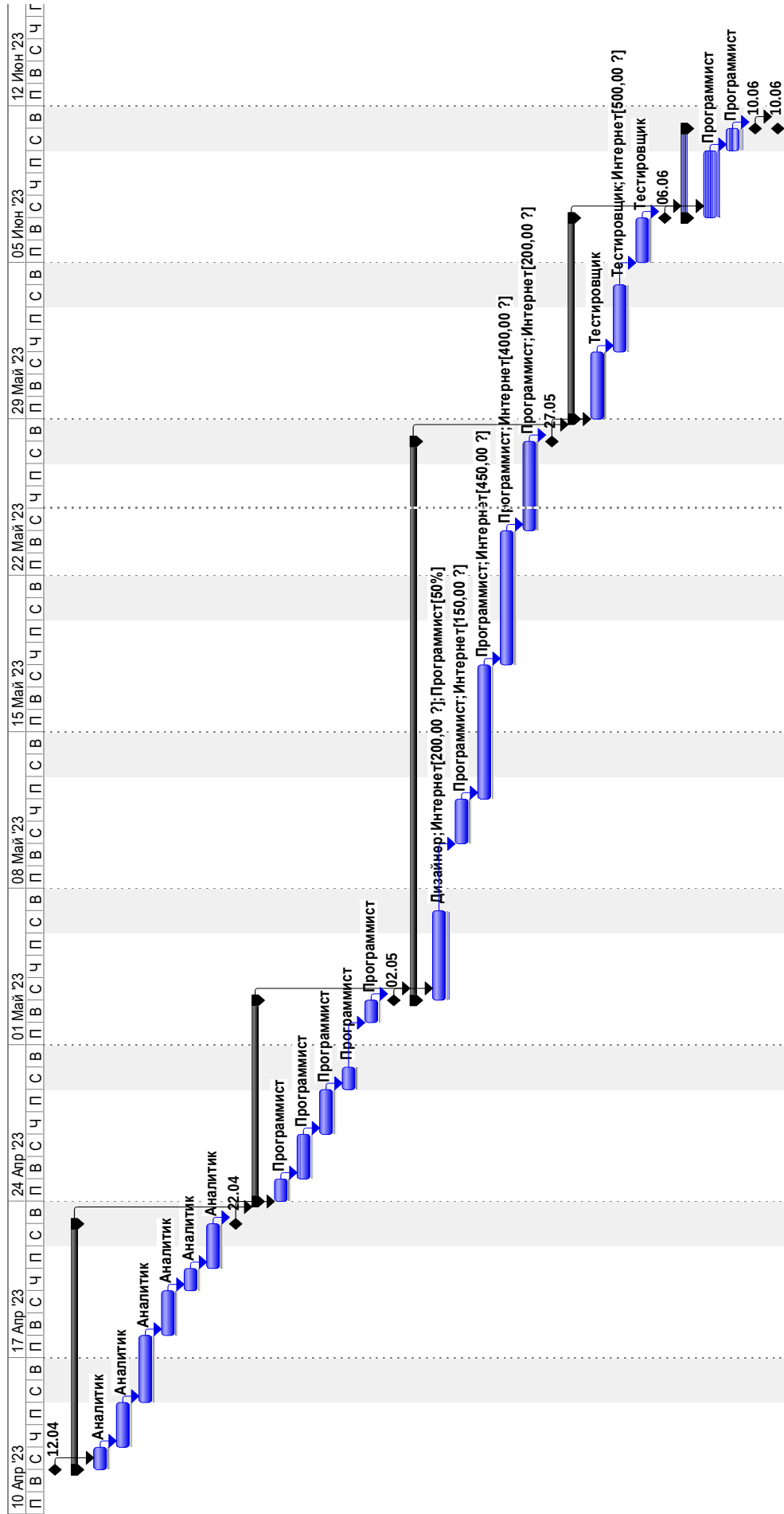
Данное программное обеспечение позволяет производить разработку планов, распределение ресурсов по задачам, отслеживание прогресса и анализ объёмов работ, позволяет генерировать расписания критического пути. Расписания могут быть составлены с учётом используемых ресурсов, цепочка которых визуализируется в диаграмме Ганта.

Аналогами данной программы являются следующие продукты:

- Naumen Project Ruler;
- ADVANTA;
- Spider Project и т. д.

В систему Microsoft Project были внесены перечни задач и ресурсов с их распределением, после чего программой была сгенерирована диаграмма Ганта для визуального отображения проведения работ с использованием соответствующим им ресурсам (рисунок 4). За дату начала работ было взято 12 апреля.

Согласно полученной диаграмме окончание работ по проекту запланировано на 10 июня.



Далее, программой были сгенерированы отчеты по общим затратам по проекту (рисунок 6) и использованию ресурсов (рисунок 5).

Использование ресурсов от Чт 25.05.23
Проект 1

	05.06.23	12.06.23	Итого
Аналитик			80 ч
Анализ функциональных возможностей сервиса			8 ч
Анализ архитектуры и схемы работы сервиса			16 ч
Анализ структуры и жизненного цикла метаданных			16 ч
Анализ задач управления метаданными			16 ч
Составление перечня автоматизируемых функций			8 ч
Планирование и функционально-стоимостный анализ процесса разработки			16 ч
Программист	32 ч		232 ч
Разработка функциональных требований к программным средствам			8 ч
Разработка требований к архитектуре программных средств			16 ч
Постановка требований к интерфейсу программных средств			16 ч
Выбор технологий разработки			8 ч
Постановка требований к аппаратно-программному обеспечению			8 ч
Разработка пользовательского интерфейса			16 ч
Разработка программных средств взаимодействия с серверной частью сервиса			16 ч
Разработка программных средств языкового представления в режиме свободного ввода			40 ч
Разработка программных средств языкового представления в режиме конструктора			40 ч
Разработка программных средств проверки корректности метаданных			32 ч
Оформление и сбор документации	24 ч		24 ч
Ввод в эксплуатацию	8 ч		8 ч
Дизайнер			16 ч
Разработка пользовательского интерфейса			16 ч
Тестировщик	16 ч		64 ч
Планирование процесса контроля качества программных средств			24 ч
Тестирование программных средств			24 ч
Оценка качества программного кода	16 ч		16 ч
Интернет			
Разработка пользовательского интерфейса			
Разработка программных средств взаимодействия с серверной частью сервиса			
Разработка программных средств языкового представления в режиме свободного ввода			
Разработка программных средств языкового представления в режиме конструктора			
Разработка программных средств проверки корректности метаданных			
Тестирование программных средств			
Компьютер			
Итого	48 ч		392 ч

Рисунок 5 – Отчет об использовании ресурсов

По данным отчета о ресурсах можно определить, что общая занятость трудовых ресурсов составляет 392 часа с 12.04.2023 по 10.06.2023 с учетом шестидневной рабочей недели.

Отчет о бюджете от Чт 25.05.23
Проект 1

ИД.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
20	Разработка программных средств я	0,00 Р	Пропорциональное	10 450,00 Р	0,00 Р	10 450,00 Р
21	Разработка программных средств я	0,00 Р	Пропорциональное	10 400,00 Р	0,00 Р	10 400,00 Р
22	Разработка программных средств п	0,00 Р	Пропорциональное	8 200,00 Р	0,00 Р	8 200,00 Р
18	Разработка пользовательского интe	0,00 Р	Пропорциональное	6 920,00 Р	0,00 Р	6 920,00 Р
30	Оформление и сбор документации	0,00 Р	Пропорциональное	6 000,00 Р	0,00 Р	6 000,00 Р
26	Тестирование программных средств	0,00 Р	Пропорциональное	4 580,00 Р	0,00 Р	4 580,00 Р
19	Разработка программных средств в	0,00 Р	Пропорциональное	4 150,00 Р	0,00 Р	4 150,00 Р
25	Планирование процесса контроля к	0,00 Р	Пропорциональное	4 080,00 Р	0,00 Р	4 080,00 Р
12	Разработка требований к архитектуре	0,00 Р	Пропорциональное	4 000,00 Р	0,00 Р	4 000,00 Р
13	Постановка требований к интерфей	0,00 Р	Пропорциональное	4 000,00 Р	0,00 Р	4 000,00 Р
4	Анализ архитектуры и схемы работ	0,00 Р	Пропорциональное	3 200,00 Р	0,00 Р	3 200,00 Р
5	Анализ структуры и жизненного цик	0,00 Р	Пропорциональное	3 200,00 Р	0,00 Р	3 200,00 Р
6	Анализ задач управления метаданн	0,00 Р	Пропорциональное	3 200,00 Р	0,00 Р	3 200,00 Р
8	Планирование и функционально-ст	0,00 Р	Пропорциональное	3 200,00 Р	0,00 Р	3 200,00 Р
27	Оценка качества программного кода	0,00 Р	Пропорциональное	2 720,00 Р	0,00 Р	2 720,00 Р
11	Разработка функциональных требо	0,00 Р	Пропорциональное	2 000,00 Р	0,00 Р	2 000,00 Р
14	Выбор технологий разработки	0,00 Р	Пропорциональное	2 000,00 Р	0,00 Р	2 000,00 Р
15	Постановка требований к аппаратн	0,00 Р	Пропорциональное	2 000,00 Р	0,00 Р	2 000,00 Р
31	Ввод в эксплуатацию	0,00 Р	Пропорциональное	2 000,00 Р	0,00 Р	2 000,00 Р
3	Анализ функциональных возможнос	0,00 Р	Пропорциональное	1 600,00 Р	0,00 Р	1 600,00 Р
7	Составление перечня автоматизиру	0,00 Р	Пропорциональное	1 600,00 Р	0,00 Р	1 600,00 Р
1	Начало работы над проектом	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
9	Анализ предметной области и пост	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
16	Анализ требований на разработку п	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
23	Разработка программных средств з	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
28	Контроль качества программных ср	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
32	Сдача проекта окончена	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
33	Конец работы над проектом	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
				89 500,00 Р	0,00 Р	89 500,00 Р

Рисунок 6 – Отчет об общих затратах проекта

На основе отчета о бюджете проекта можно сделать вывод, что для разработки программных средств потоковой обработки событий необходимо 89500 рублей.

2 Анализ требований на разработку программных средств

2.1 Функциональные требования к программным средствам

Для управления объектами метаданных с помощью языкового представления пользователь должен обладать ролью разработчика, который, в свою очередь, может создавать и изменять соответствующие объекты. Управляемыми объектами могут быть события и активные правила.

События делятся на:

- агрегированные события;
- сложные или комплексные события;
- элементарные или простые события.

В зависимости от типа события должен выстраиваться языковой шаблон объекта метаданных, поэтому перед началом создания события у пользователя должна быть возможность выбора типа события. При модификации конкретного события тип уже известен, поэтому этап выбора необходимо опустить.

После выбора шаблона объекта метаданных пользователь должен иметь возможность набора кода объекта. Для данного действия должно быть два инструмента: поле свободного ввода и конструктор. Оба инструмента должны обрабатывать введенный пользователем код на выявление ошибок и несоответствий в синтаксической структуре объектов.

Далее, когда проверка кода системой пройдена и пользователь сохраняет введенные данные, происходит отправка данных на сервер. Программные средства должны обеспечивать обработку ответов, поступающих от REST-сервера. Кроме того, для доставки каждого запроса и получения ответа об успешной обработке, необходим токен доступа, получаемый при авторизации пользователя.

Согласно вышеописанным требованиям была разработана диаграмма вариантов использования, показанная на рисунке 7 [4].

На рисунке 8 изображена спецификация варианта использования «Выбрать элементарное событие для создания».

Выбрать элементарное событие для создания	
Id: 3	
Описание:	Разработчик создает новое элементарное событие в системе
Действующие лица:	Разработчик, REST-сервер
Предусловия:	Разработчик вошел в систему
Основной поток:	<ol style="list-style-type: none"> 1. Разработчик нажимает на кнопку «Создать событие» 2. Разработчик указывает тип «Элементарное событие» 3. Разработчик выбирает инструмент управления объектом метаданных 4. Разработчик указывает описание события 5. Разработчик указывает имя события 6. Разработчик указывает устройство, генерирующее данное событие 7. Разработчик нажимает на кнопку «Сохранить» 8. REST-сервер отправляет данные о событии в базу данных 9. REST-сервер получает данные созданного события
Постусловие:	Разработчик создал новое элементарное событие

Рисунок 8 – Спецификация для варианта использования «Выбрать элементарное событие для создания»

На рисунке 9 изображена спецификация варианта использования «Выбрать конструктор объекта метаданных».

Выбрать конструктор объекта метаданных	
Id: 4	
Описание:	Разработчик создает объект метаданных в режиме конструктора
Действующие лица:	Разработчик, Система, REST-сервер
Предусловия:	Разработчик вошел в систему и начинает создание объекта метаданных
Основной поток:	<ol style="list-style-type: none"> 1. Разработчик выбирает тип объекта метаданных 2. Система формирует шаблон для выбранного типа объекта метаданных 3. Разработчик выбирает вкладку конструктора 4. Разработчик заполняет все обязательные поля объекта метаданных 5. Разработчик нажимает кнопку «Сохранить» 6. REST-сервер отправляет данные о событии в базу данных 7. REST-сервер получает данные созданного события
Постусловие:	Разработчик создал объект метаданных в выбранном режиме конструктора

Рисунок 9 – Спецификация для варианта использования «Выбрать конструктор объекта метаданных»

В результате анализа функциональных требований была составлена диаграмма вариантов использования для языкового представления объектов метаданных портала управления, были рассмотрены спецификации некоторых

вариантов использования и определен основной поток действий, необходимый для исполнения данных функций.

2.2 Требования к архитектуре программных средств

Архитектура программных средств потоковой обработки событий и исполнения активных правил представляет собой микросервисную архитектуру, содержащую в себе несколько независимых модулей. Для построения такой архитектуры на клиентской части портала управления используется библиотека single-spa [5]. Данная библиотека помогает соединить несколько микрофронтендов в одно приложение и предоставляет следующие преимущества:

- использование множества различных фреймворков в одном приложении и даже на одной странице;
- независимое развёртывание любого из модулей приложения;
- разработка нового модуля с помощью любого фреймворка или даже без него без переписывания существующего кода;
- отложенная загрузка каждого модуля для быстрой загрузки модулей во время появления на странице.

Приложения с использованием библиотеки single-spa состоят из:

- приложение корневой конфигурации;
- микросервисные приложения (модули).

Приложение корневой конфигурации отображает HTML-страницу и Javascript, регистрирующий приложения. Каждое приложение регистрируется тремя составляющими:

- имя приложения (модуля);
- функция для загрузки кода приложения;
- функция, определяющая, когда приложение активно или неактивно.

Микросервисные приложения можно рассматривать как одностраничные приложения, упакованные в модули. Каждое приложение должно знать, как загружать, монтировать и отключать себя из DOM (Document Object Model).

Основное различие между традиционными SPA (Single Page Application) приложениями и приложениями, написанными с помощью библиотеки single-spa заключается в том, что они должны иметь возможность сосуществовать с другими приложениями, поскольку у каждого из них нет собственной HTML-страницы для отображения.

Серверная часть сервиса представлена REST-сервером, который служит промежуточным слоем между запрашиваемыми ресурсами и базой данных. Сервер принимает запросы от любого модуля клиентской части приложения, обрабатывает его, затем формирует и отправляет SQL-запрос согласно пришедшим от клиента данным. После того как от базы данных получен ответ, сервер обрабатывает его, упаковывает в объект типа JSON и направляет модулю, который запрашивал данные.

На рисунке 10 изображена диаграмма разворачивания сервиса, которая отображает архитектуру программных средств управления метаданными.

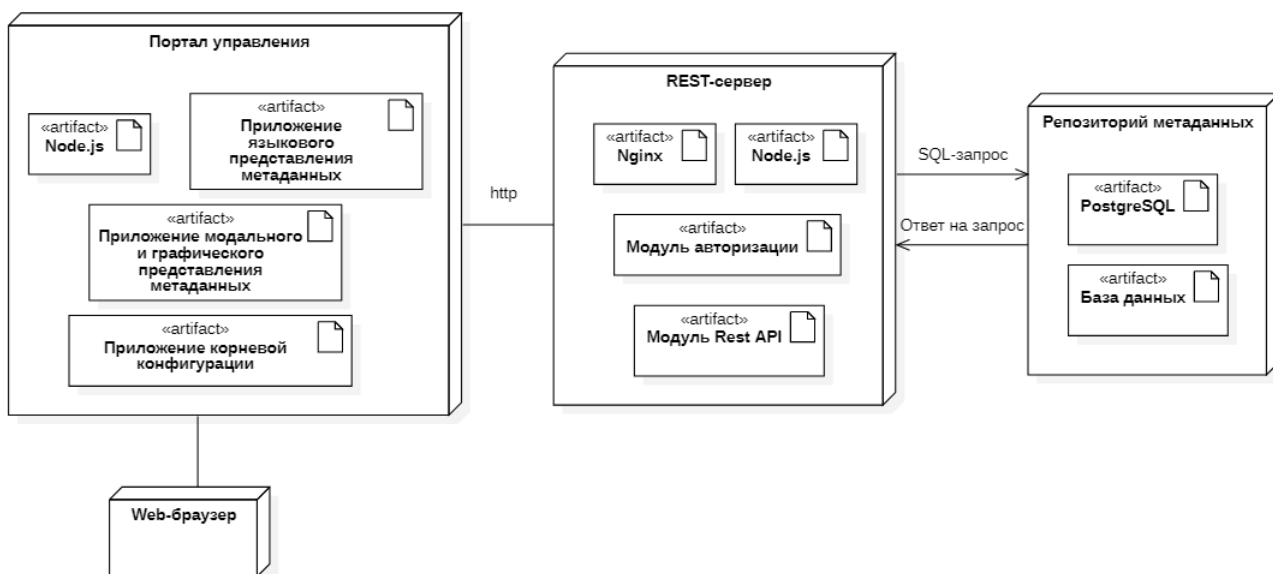


Рисунок 10 – Архитектура программных средств управления метаданными

Приложение языкового представления должно быть разработано согласно паттерну Model-View-View-Model [6]. Согласно данному шаблону проектирования архитектурно приложение делится на:

- модель (Model), которая представляет собой модели данных приложения;
- представление (View), которое представляет графический интерфейс, отображающий данные, пришедшие от модели;
- модель представления (View-Model), которая является абстракцией представления и используется для автоматического изменения состояния представления и наоборот.

Согласно представленному шаблону проектирования элементы представления должны быть связаны с событиями и свойствами представления модели. Между View-model и представлением должна быть настроена двухсторонняя коммуникация для автоматического обновления отображения данных представления. Данный механизм называется механизмом связывания данных.

2.3 Требования к пользовательскому интерфейсу программных средств

Разрабатываемые программные средства представляют собой веб-приложение, а значит должно иметь не только расширенные функциональные возможности, но и удобный пользовательский интерфейс, который обладает низким порогом входа. Кроме того, интерфейс должен соответствовать выполняемым функциям сервиса и обеспечивать высокую скорость работы с ними.

Поскольку поддержка языкового представления является частью портала управления объектами метаданными, которую необходимо внедрить как отдельное приложение в архитектуру микрофронтендов согласно описанным требованиям к архитектуре программных средств, интерфейс приложения должен представлять из себя диалоговое окно. Компоненты диалогового окна должны меняться в зависимости от типа объектов метаданных и выполняемого пользователем действия по управления метаданными.

Программные средства поддержки языкового представления включают в себя разработку двух инструментов: конструктора и инструмента свободного ввода кода объектов метаданных.

Инструмент свободного ввода языковых конструкций должен содержать редактор с возможностью подсветки основных компонентов синтаксических конструкций, например, строковых и числовых типов данных. Также при наличии в языковой конструкции параметра, требующего ввод набора каких-либо объектов, интерфейс может иметь вспомогательные выпадающие списки с информацией об этих объектах для удобства пользователя. Кроме того, одним из обязательных требований к интерфейсу являются компоненты для проверки корректности введенных языковых конструкций и сохранения объекта метаданных.

Конструктор языкового представления должен содержать поля для ввода данных об объекте метаданных. Языковые конструкции не должны отличаться от конструкций редактора свободного ввода, но должны иметь предзаполненные ключевые слова и давать пользователю возможность только изменять данные о параметрах объекта метаданных.

2.4 Обоснование выбора технологий и средств разработки

Согласно требованиям к интерфейсу, функциональным и архитектурным требованиям, были определены необходимые технологии и средства разработки языкового представления метаданных.

Для реализации компонентов приложения решено использовать фреймворк Angular [7]. Angular – это открытая платформа для разработки одностраничных веб-приложений. Данный фреймворк использует язык программирования Typescript, который в свою очередь расширяет возможности языка JavaScript для лучшей интеграции с редакторами кода. TypeScript преобразуется в JavaScript для отображения в браузере, а также понимает синтаксические конструкции данного языка. Кроме того, язык Typescript

использует строгую типизацию, что делает код более упорядоченным, а это в свою очередь упрощает и ускоряет разработку.

Для разработки графического интерфейса используются язык гипертекстовой разметки HTML и каскадная таблица стилей CSS. Данные технические инструменты поддерживаются всеми веб-браузерами и также используются в связке с выбранным фреймворком для разработки. Для более удобной стилизации приложения решено использовать препроцессор каскадной таблицы стилей Sass в синтаксисе scss [8]. Данный препроцессор расширяет возможности основной CSS и дает следующие преимущества при разработке:

- использование переменных для хранения общих стилей для всего проекта или его отдельных частей;
- вложенность и четкая иерархия стилей блоков и их элементов;
- импорт файлов – часть общих стилей может храниться в одном файле и легко импортироваться в файл частных стилей компонентов;
- использование математических операторов.

Построение диаграмм осуществляется с помощью языка UML 2.0 в среде StarUML. Данное программное обеспечение предоставляет возможность быстро и удобно создавать все диаграммы стандарта UML 2.0: диаграммы классов, последовательности, вариантов использования, развертывания и т. д.

Кроме вышеперечисленных технологий в разработке также используются следующие средства:

- single-spa-angular – библиотека для построения отдельных модулей в микросервисной архитектуре клиентской части приложения с поддержкой фреймворка Angular;
- PrimeNg – библиотека компонентов пользовательского интерфейса, позволяющая быстро разрабатывать интерфейс Angular приложения;
- CodeMirror – библиотека редактора кода в свободном формате с функциями автоматической подсветки кода [9];
- Karma – средство автоматического модульного тестирования приложений, разработанных на языке программирования JavaScript или

TypeScript; данное средство встроено в Angular и не требует дополнительных инсталляций и настроек;

- Node.js – платформа для исполнения языка JavaScript;
- npm – пакетный менеджер среды Node.js, использующийся для установки пакетов, необходимых для разработки и корректной работы приложения;
- Docker – программное обеспечение, позволяющее быстро и удобно собирать, разворачивать и эксплуатировать приложения;
- Figma – графический редактор для создания рисунков, схем и построения проекта пользовательского интерфейса приложения.

Для разработки выбрана среда разработки WebStorm от компании JetBrains, которая обеспечивает быстрый и точный анализ кода, рефакторинг кода, его простую отладку и автоматическое исправление форматирования по заданному шаблону. Разработка осуществляется под управлением операционной системы Windows 10, запуск и просмотр разрабатываемого приложения – в браузере Google Chrome.

2.5 Требования к аппаратно-программному обеспечению

Исходя из выбранных технологий и средств разработки были выявлены основные требования к аппаратно-программному обеспечению.

Прежде всего для развертывания сервиса необходимо наличие Docker. Это программное обеспечение позволяет запускать приложение в виртуальной среде. Основные системные требования для установки данной платформы для операционной системы Windows [10]:

- Windows 10 64-bit: Pro, Enterprise, Education (Build 16299 или выше);
- наличие функций Hyper-V и Containers Windows.

Требования к оборудованию для успешного запуска Hyper-V на Windows 10:

- 64-битный процессор с поддержкой функции Second Level Address Translation (SLAT);

- 4 GB памяти системы;
- на уровне BIOS должна быть включена поддержка аппаратной виртуализации.

Поскольку доступ к программным средствам будет обеспечиваться посредством выхода в интернет и просмотра в веб-браузере, необходимо выяснить, какие браузеры поддерживают технологии разрабатываемого сервиса. Согласно документациям используемых технологий разработки, корректное отображение обеспечивают следующие браузеры:

- Google Chrome;
- Mozilla Firefox;
- Safari;
- Microsoft Edge;
- Internet Explorer.

Согласно вышеописанным требованиям веб-браузеров, работа разрабатываемых программных средств гарантирована на большинстве современных компьютерных систем с доступом в интернет.

3 Разработка программных средств

3.1 Разработка архитектуры развертывания программных средств языкового представления

Проанализировав требования к разрабатываемому приложению, были определены основные компоненты, из которых состоят программные средства. Так как разработка ведется с помощью фреймворка Angular, приложение должно иметь модульную архитектуру, то есть каждый обособленный компонент приложения должен быть оформлен как модуль. Структура программных средств языкового представления изображена на рисунке 11.

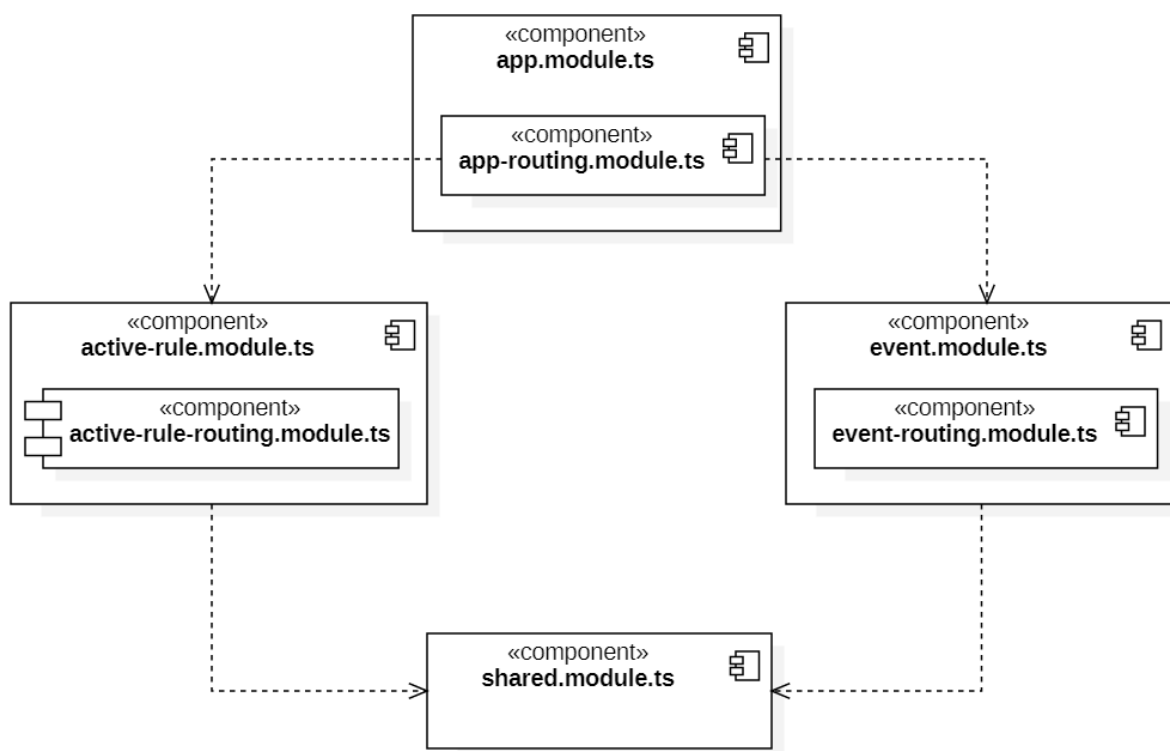


Рисунок 11 – Диаграмма основных компонентов приложения

Описание основных действий каждого компонента диаграммы представлено в таблице 4.

Таблица 4 – Описание действий компонентов приложения

№	Наименование компонента	Описание
1	app.module.ts	Центральный модуль приложения, являющийся точкой входа
2	app-routing.module.ts	Модуль маршрутизации приложения, загружающий остальные модули приложения
3	event.module.ts	Модуль компонентов событий
4	event-routing.module.ts	Модуль маршрутизации компонентов модуля событий
5	active-rule.module.ts	Модуль компонентов активных правил
6	active-rule-routing.module.ts	Модуль маршрутизации компонентов модуля активных правил
7	shared.module.ts	Модуль компонентов, используемых в нескольких модулях приложения

Модули `app.module.ts`, `event.module.ts` и `active-rule.module.ts` содержат в себе модули маршрутизации. Данные модули содержат в себе конфигурации маршрутов текущего модуля. В качестве примера таких конфигураций в листинге 1 представлен код модуля маршрутизации `app-routing.module.ts`.

Листинг 1 – Код модуля маршрутизации `app-routing.module.ts`

```
const routes: Routes = [
  {
    loadChildren: () => import('./active-rule/active-rule.module').then((m) => m.ActiveRuleModule),
    path: AppRoutesEnum.ACTIVE_RULE_SCRIPT
  },
  {
    loadChildren: () =>
import('./event/event.module').then((m) => m.EventModule),
    path: AppRoutesEnum.EVENT_SCRIPT
  }
];
```

Каждый из описанных модулей содержит в себе один или несколько компонентов, выполняющих более узконаправленные функции. Детальное содержание компонентов приложения показано на диаграмме пакетов, изображенной на рисунке 12.

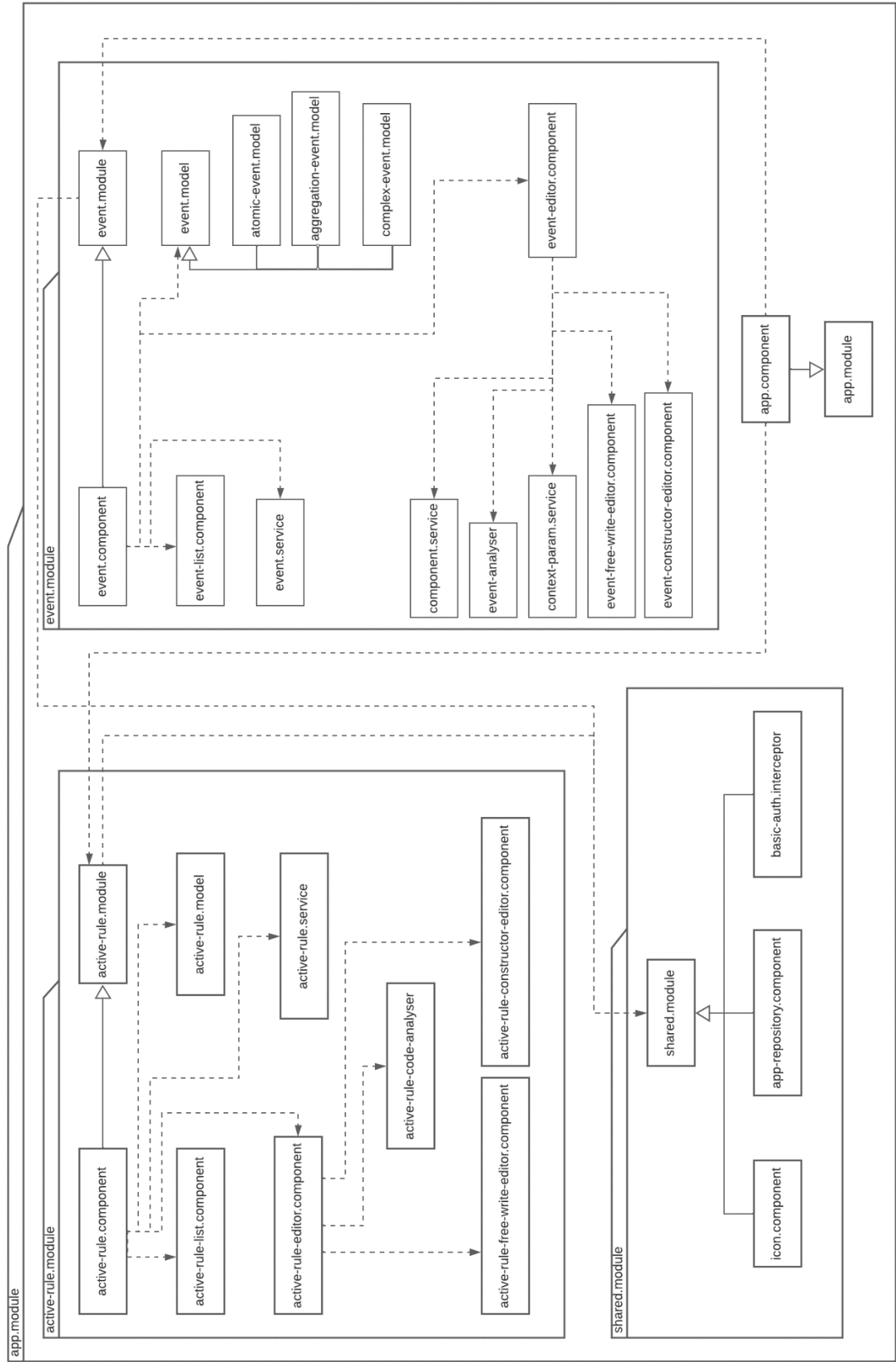


Рисунок 12 – Диаграмма пакетов приложения

Описание каждой сущности диаграммы представлено в таблице 5.

Таблица 5 – Описание сущностей диаграммы пакетов приложения

№	Сущность	Описание
1	app.module	Основной модуль приложения, внедряющий в себя зависимости всех остальных модулей и корневых сервисов
2	app.component	Основной компонент приложения, представляющий web-страницу сервиса
3	active-rule.module	Модуль активных правил
4	active-rule.component	Компонент диалогового окна активных правил
5	active-rule-list.component	Компонент списка всех активных правил
6	active-rule.service	Сервис, предоставляющий HTTP запросы для модуля активных правил
7	active-rule.model	Модель данных активных правил
8	active-rule-editor.component	Компонент редактора активных правил
9	active-rule-free-write-editor.component	Компонент редактора свободного ввода языковых конструкций активных правил
10	active-rule-constructor-editor.component	Компонент редактора языкового конструктора активных правил
11	active-rule-analyser	Анализатор языковых конструкций активных правил
12	event.module	Модуль событий
13	event.component	Компонент диалогового окна событий
14	event-list.component	Компонент списка всех событий
15	event.service	Сервис, предоставляющий HTTP запросы для модуля событий
16	event.model	Обобщенная модель данных событий
17	atomic-event.model	Модель данных атомарного события
18	aggregation-event.model	Модель данных агрегирующего события
19	complex-event.model	Модель данных сложного события
20	event-editor.component	Компонент редактора событий
21	event-free-write-editor.component	Компонент редактора свободного ввода языковых конструкций событий

Продолжение таблицы 5

№	Сущность	Описание
22	event-creator-editor.component	Компонент редактора языкового конструктора событий
23	event-analyser	Анализатор языковых конструкций событий
24	context-param.service	Сервис, предоставляющий HTTP запросы для параметров контекста
25	component.service	Сервис, предоставляющий HTTP запросы для компонентов системы
26	shared.module	Модуль общих компонентов системы
27	icon.component	Компонент иконок приложения
28	app-repository	Компонент хранилищ приложения
29	basic-auth.interceptor	Механизм для автоматической подстановки токена авторизации к каждому HTTP запросу

Каждая из сущностей программных средств относится к одному из следующих типов:

- компонент;
- модуль;
- сервис;
- модель данных;
- интерсептор.

Компоненты представляют собой единицу пользовательского интерфейса со своей логикой, шаблоном и стилями. Поэтому каждый компонент содержит файлы с расширениями html, ts и scss.

Модули служат изолирующей логической структурой программных средств, помогают разделить приложение на логически обособленные части.

Сервисы являются поставщиками данных и никак не отображаются на пользовательском интерфейсе. В основном они необходимы для хранения дополнительной логики и данных, в случае программных средств языкового представления сервисы служат прослойкой между клиентской и серверной частями приложения.

Модели необходимы для локального хранения данных, которые были получены от сервера или сгенерированы программой.

Интерсепторы представляют собой специальный механизм обработки HTTP запросов, например, это может быть автоматическая подставка параметров запроса или обработка ошибок общего типа.

3.2 Разработка языковых конструкций объектов метаданных

Перед началом разработки программных средств обработки событий и исполнения активных правил необходимо определить языковые конструкции каждого из типов объектов. Согласно описанным выше требованиям существуют следующие типы объектов метаданных:

- активные правила;
- события.

События в свою очередь делятся на атомарные, агрегированные и сложные.

В зависимости от типа событие имеет различные параметры. Аналогично, активные правила имеют параметры, задаваемые при создании или редактировании объекта. Каждому параметру должно задаваться значение, что говорит о необходимости выбора единой синтаксической конструкции для определения параметров объекта метаданных. Для подобного рода выражений была выбрана конструкция вида:

$$KW = V,$$

где KW – ключевое слово определяемого параметра;

V – значение определяемого параметра.

В качестве знака присваивания был выбран знак «=». Тип значения зависит от определяемого параметра.

Ниже представлены параметры всех типов событий, определенных в приложении.

Параметры атомарного (простого) события:

- описание события;

- устройство (параметр контекста) события;
- компонент, к которому принадлежит событие.

Параметры агрегированного события:

- описание события;
- устройство (параметр контекста) события;
- временные рамки выполнения события.

Параметры сложного (составного) события:

- описание события;
- устройство (параметр контекста) события;
- набор составных событий.

Для объявления события при создании и редактировании выбрано ключевое слово `EVENT`. Ниже представлены языковые конструкции каждого из типов событий согласно описанным параметрам (листинги 2-4).

Листинг 2 – Языковая конструкция атомарного события

```
EVENT <код события>
DESCRIPTION = "<описание события>"
COMPONENT = "<код устройства>"
CONTEXT = "<код параметра контекста>"
```

Данная языковая конструкция содержит описание параметров атомарного события посредством ключевых слов `DESCRIPTION`, `COMPONENT` и `CONTEXT`. Присвоенное значение полю `DESCRIPTION` описывает основное действие событие, `COMPONENT` – компонент, к которому принадлежит событие, `CONTEXT` – параметр контекста (устройство), которое способствует выполнению события.

Листинг 3 – Языковая конструкция агрегирующего события

```
EVENT <код события>
DESCRIPTION = "<описание события>"
TIMESTAMP_BEGIN = "<время начала события>"
TIMESTAMP_END = "<время окончания события>"
CONTEXT = "<код параметра контекста>"
```

Конструкции, содержащие ключевые слова DESCRIPTION и CONTEXT несут ту же смысловую нагрузку, что и в случае с атомарным событием. Дополнительно, агрегирующее событие содержит конструкции с ключевыми словами TIMESTAMP_BEGIN и TIMESTAMP_END, которые содержат значения временного промежутка события (начала и окончания соответственно).

Листинг 4 – Языковая конструкция сложного события

```
EVENT <код события>
DESCRIPTION = "<описание события>"
EVENTS = "<набор событий в составе>"
CONTEXT = "<код параметра контекста>"
```

Также, как и в случае с атомарным и агрегирующим событиями, в языковом шаблоне присутствуют конструкции с ключевыми словами DESCRIPTION и CONTEXT. Сложное событие состоит из нескольких других событий. События состава описаны конструкцией, начинающейся с ключевого слова EVENTS.

Определение активных правил начинается с ключевого слова RULE. Параметры активного правила следующие:

- описание правила;
- набор событий правила;
- условие выполнения правила;
- действие правила.

Согласно определенным параметрам активного правила и составным языковым конструкциям, составлен языковой шаблон активного правила, представленный на листинге 5.

Листинг 5 – Языковая конструкция активного правила

```
RULE <код правила>
DESCRIPTION = "<описание правила>"
EVENT = "<набор событий правила>"
CONDITION = "<условие выполнения события>"
ACTION = "<действие правила>"
```

В данной конструкции представлены ключевые слова DESCRIPTION, EVENT, CONDITION, ACTION, которые задают описание, набор событий, условие и действие активного правила соответственно.

Во всех языковых конструкциях объектов метаданных, описанных выше, присутствует включение уникального кода объекта. Данный параметр определяется базой данных при создании объекта, поэтому он необходим только при редактировании для обновления данных. Также, все значения параметров, обрамленные в кавычки, имеют строковый тип данных, остальные – целочисленный.

3.3 Разработка пользовательского интерфейса

Поскольку программные средства разрабатываются под систему, управляемой пользователем, он должен быть ориентирован на человека, иметь логичный и интуитивно понятный интерфейс, позволяющий пользователям взаимодействовать без затруднений.

На основе описанных требований был разработан пользовательский интерфейс, содержащий следующие окна:

- страница списка всех активных правил;
- страница создания активного правила;
- страница редактирования правила;
- страница списка всех событий;
- страница создания события;
- страница редактирования события.

Каждая описанная страница отображается в специальном диалоговом окне.

При переходе на языковое представление правил пользователь попадает на страницу, содержащую список всех правил (рисунок 13).

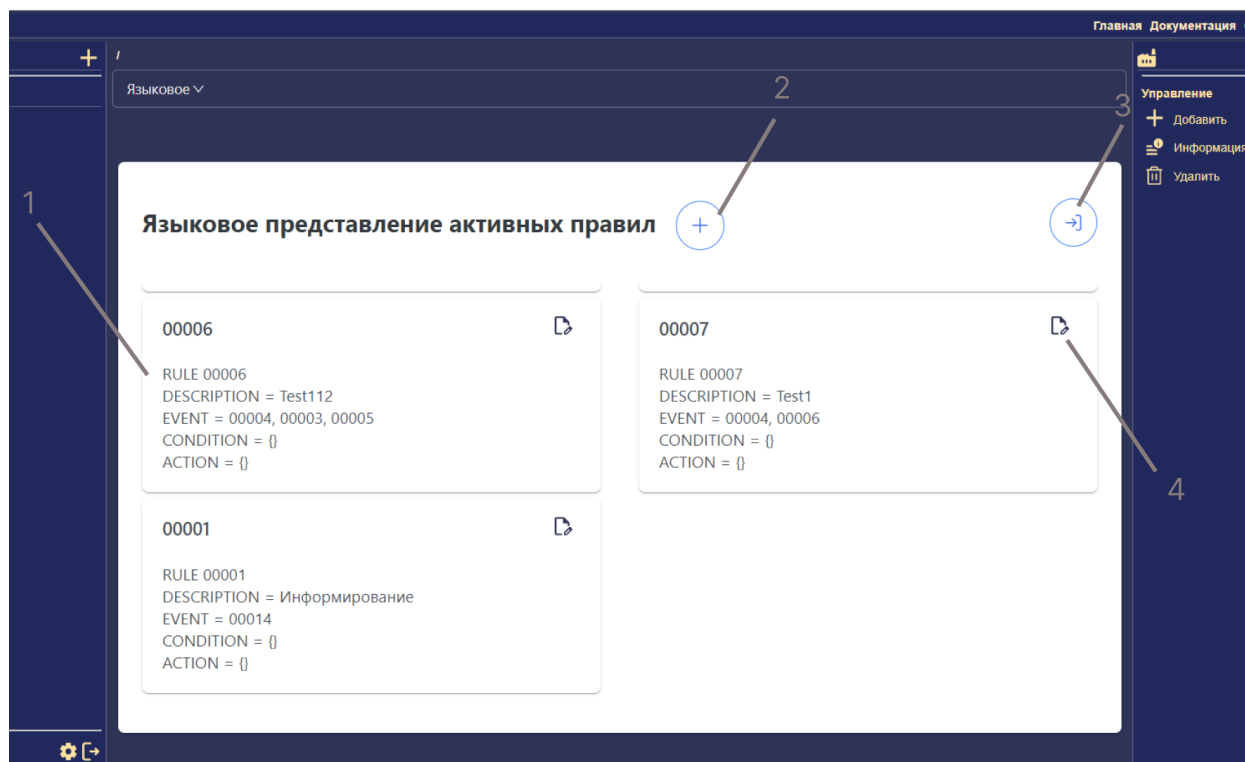


Рисунок 13 – Страница списка всех активных правил

Основные элементы страницы:

- 1 – карточка объекта активного правила;
- 2 – кнопка добавления активного правила;
- 3 – кнопка перехода к остальным представлениям активных правил;
- 4 – кнопка редактирования активного правила.

Каждая карточка на данной странице содержит в себе языковую конструкцию активного правила. Также, для наглядности, в заголовке карточки дополнительно указан уникальный код правила.

При нажатии на кнопку добавления активного правила пользователь попадает на страницу с инструментами добавления правила, один из которых изображен на рисунке 14.

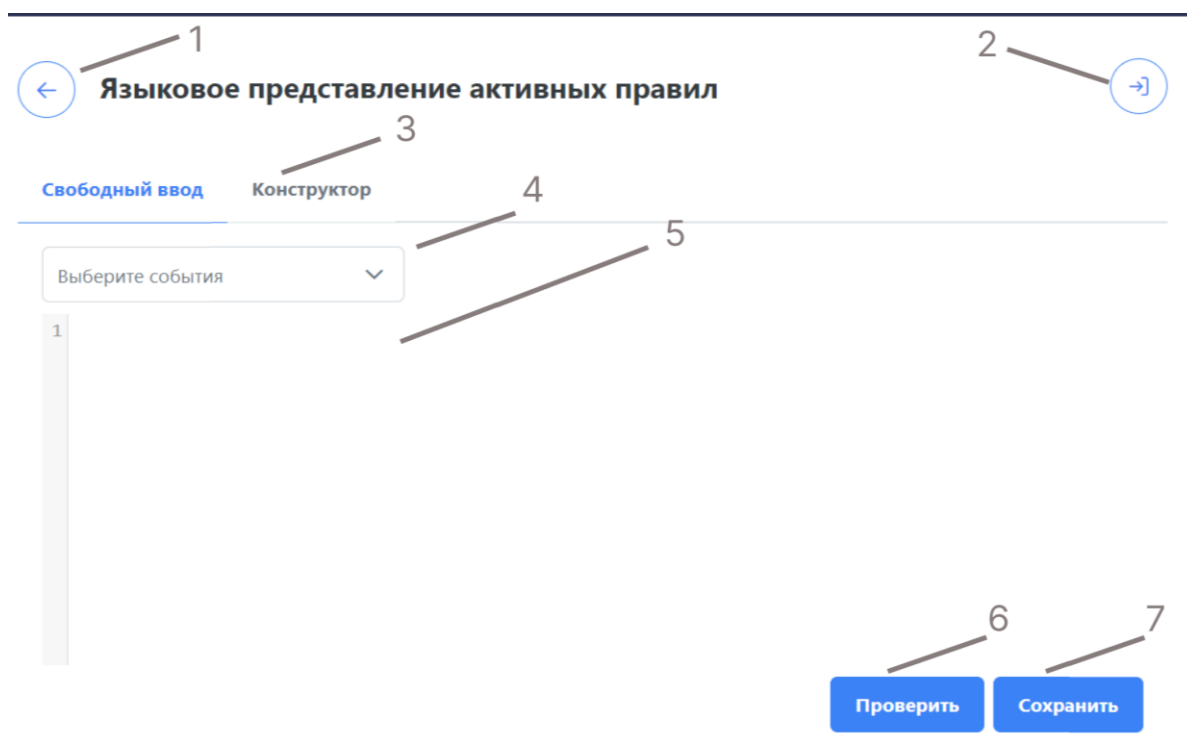


Рисунок 14 – Страница добавления активного правила в режиме свободного ввода

Данная страница содержит следующие компоненты:

- 1 – кнопка возврата к списку активных правил;
- 2 – кнопка возврата к остальным представлениям;
- 3 – переключение между инструментами управления активным правилом;
- 4 – вспомогательный список выбора событий активного правила;
- 5 – редактор языковой конструкции активного правила;
- 6 – кнопка проверки корректности синтаксической конструкции активного правила;
- 7 – кнопка сохранения введенных изменений.

При переключении инструмента управления активным правилом пользователь может выбрать конструктор, графическое представление которого изображено на рисунке 15.

Рисунок 15 – Страница добавления активного правила в режиме конструктора

В отличие от режима свободного ввода конструктор содержит заранее определенные ключевые слова правила (1) и поля для ввода значений заданных параметров (2).

При попытке выхода из редактора языкового представления появляется окно подтверждения действия пользователя (рисунок 16).

Рисунок 16 – Окно подтверждения выхода из редактора активного правила

Данное окно подтверждения содержит следующие элементы:

- 1 – текст предупреждения о последствии выхода из редактора;
- 2 – кнопка отмены действия;

3 – кнопка подтверждения действия.

При нажатии на кнопку редактирования активного правила пользователь попадает в тот же редактор, что и при создании, но с уже имеющейся языковой конструкцией правила (рисунок 17) и заполненными полями данных в режиме конструктора (рисунок 18).

The screenshot shows a web interface titled "Языковое представление активных правил". At the top, there are navigation buttons: a back arrow on the left and a forward arrow on the right. Below the title, there are two tabs: "Свободный ввод" (Free input) and "Конструктор" (Constructor). The "Свободный ввод" tab is active. Below the tabs, there is a dropdown menu showing "00003". Below the dropdown, there is a text area containing the following text:

```
1 RULE 00003
2 DESCRIPTION = "Включение увлажнителя воздуха"
3 CONDITION = "{}"
4 ACTION = "{}"
5 EVENT = "00003"
```

At the bottom right, there are two buttons: "Проверить" (Check) and "Сохранить" (Save).

Рисунок 17 – Страница редактирования активного правила в режиме свободного ввода

The screenshot shows the same web interface as Figure 17, but the "Конструктор" (Constructor) tab is active. The "Свободный ввод" (Free input) tab is inactive. Below the tabs, there are five rows of input fields:

- RULE 00003
- DESCRIPTION = Включение увлажнителя воздуха
- CONDITION = {}
- ACTION = {}
- EVENT = 00003

At the bottom right, there is a single button: "Сохранить" (Save).

Рисунок 18 – Страница редактирования активного правила в режиме конструктора

Страница языкового представления событий содержит аналогичные компоненты странице активных правил и представлена на рисунке 19.

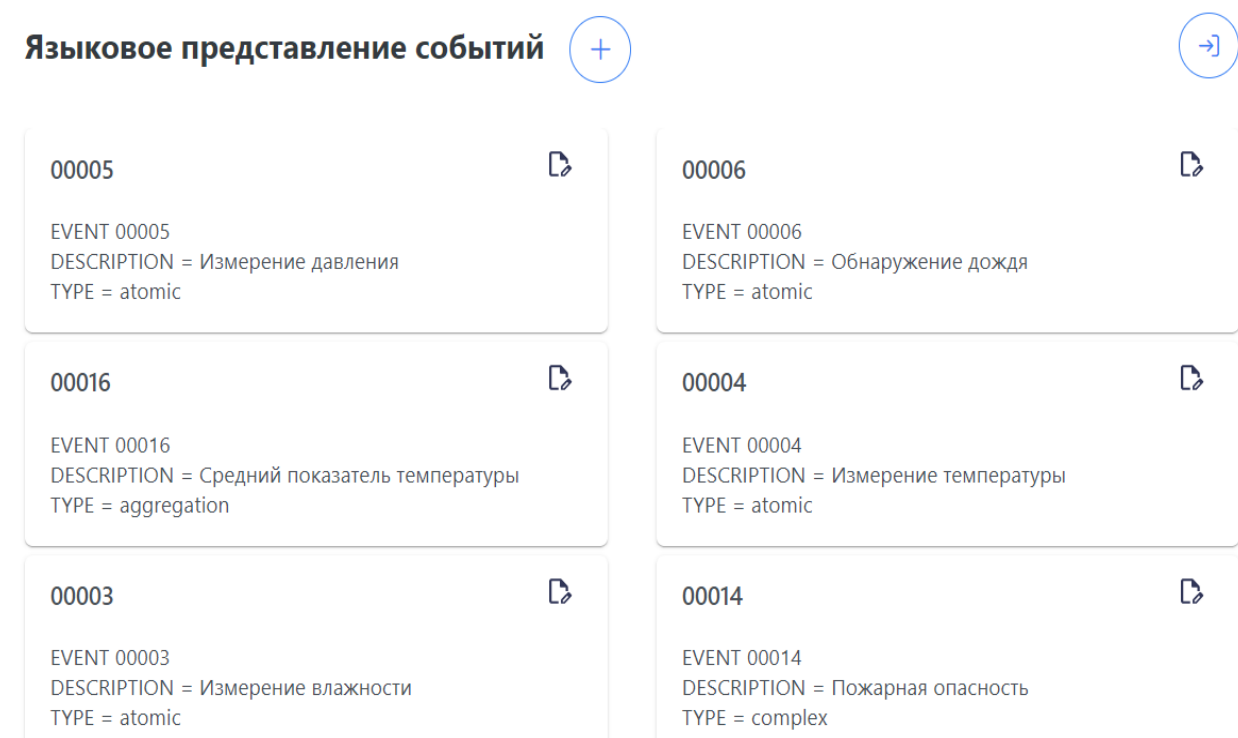


Рисунок 19 – Страница списка событий

Аналогично странице активных правил каждое событие на странице представлено карточкой. Поскольку в зависимости от типа событие имеет различные параметры, в карточках событий отображены только объединяющие их параметры: код, описание и тип события.

Страницы добавления и редактирования также содержат аналогичные элементы (рисунок 20). В отличие от редактора активных правил данная страница содержит выпадающий список, предлагающий выбрать тип события при создании. В режиме редактирования выбор типа недоступен. Кроме того, до тех пор, пока пользователь не выберет тип события, кнопки проверки синтаксической конструкции и сохранения события ему недоступны.

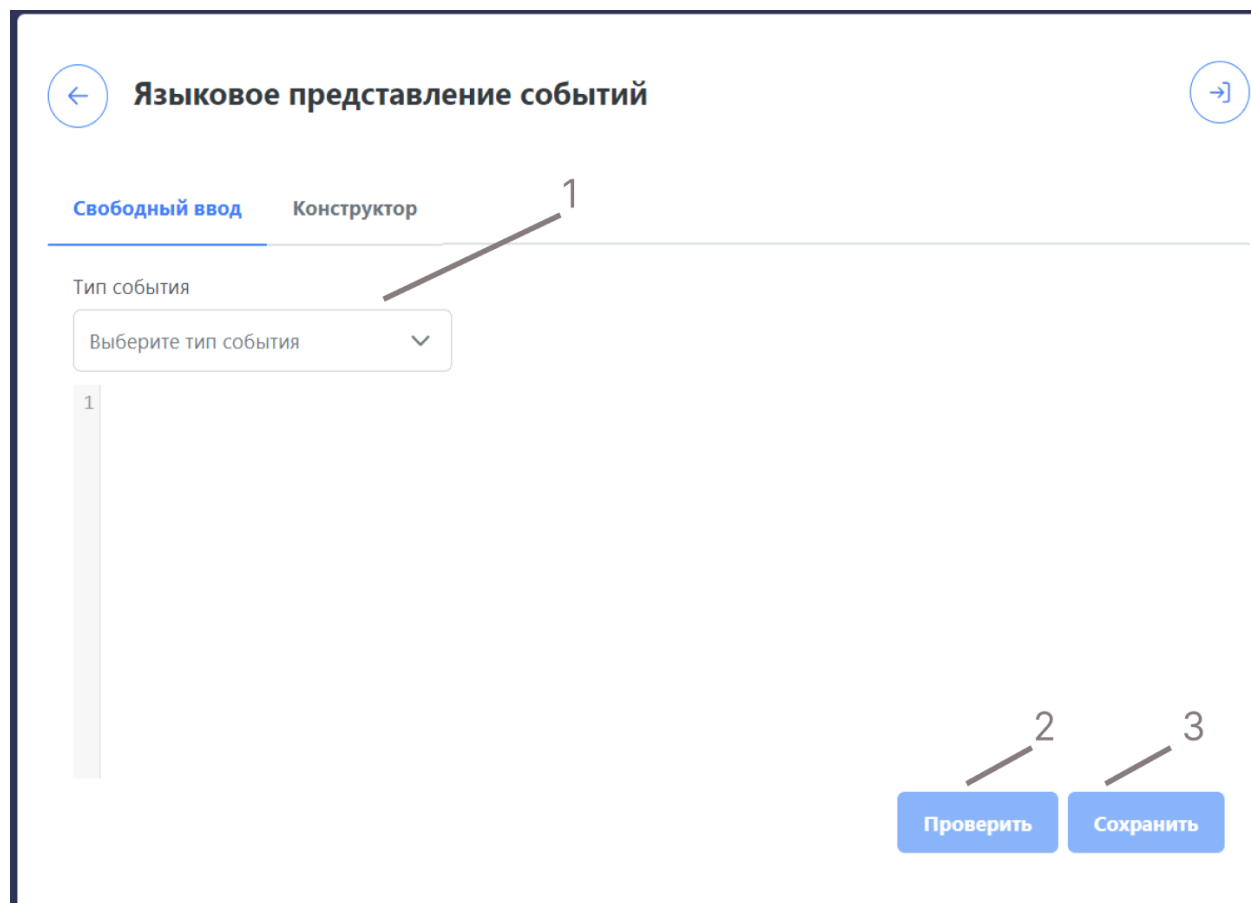


Рисунок 20 – Страница редактора события

Данная страница содержит следующие элементы:

- 1 – выпадающий список выбора типа события;
- 2 – недоступная кнопка проверки синтаксической конструкции события;
- 3 – недоступная кнопка сохранения события.

Также, как и на страницах редактора активных правил, на странице редактора событий есть две кнопки выхода: возврат к списку всех событий и возврат к другим представлениям событий и активных правил.

Конструктор события содержит все predetermined языковые конструкции, необходимые для задания события конкретного типа, с полями для ввода данных. В случае если тип события не задан, конструктор будет пуст (рисунок 21).

← Языковое представление событий →

Свободный ввод **Конструктор**

Тип события

Выберите тип события

Сохранить

Рисунок 21 – Страница редактора события в режиме конструктора при незаданном типе события

Редактор свободного ввода каждого из объектов метаданных оснащен синтаксическим анализатором введенных конструкций. В случае ошибки анализатор выдаст пользователю на экран сообщение, пример которого показан на рисунке 22. Сообщение выводится при нажатии на кнопку предварительной проверки синтаксической конструкции или при попытке сохранить объект метаданных. В случае если языковая конструкция верна, анализатор выдаст сообщение о том, что ошибки не найдены (рисунок 23).

```
1 EVENT 00005
2 DESCRIPTION "Измерение давления"
3 COMPONENT = "00003"
4 CONTEXT = "00012"
```

⊗ Ошибка в строке 2: пропущен или неверно указан знак присваивания "=" после ключевого слова "DESCRIPTION"

Проверить Сохранить

Рисунок 22 – Сообщение о синтаксической ошибке

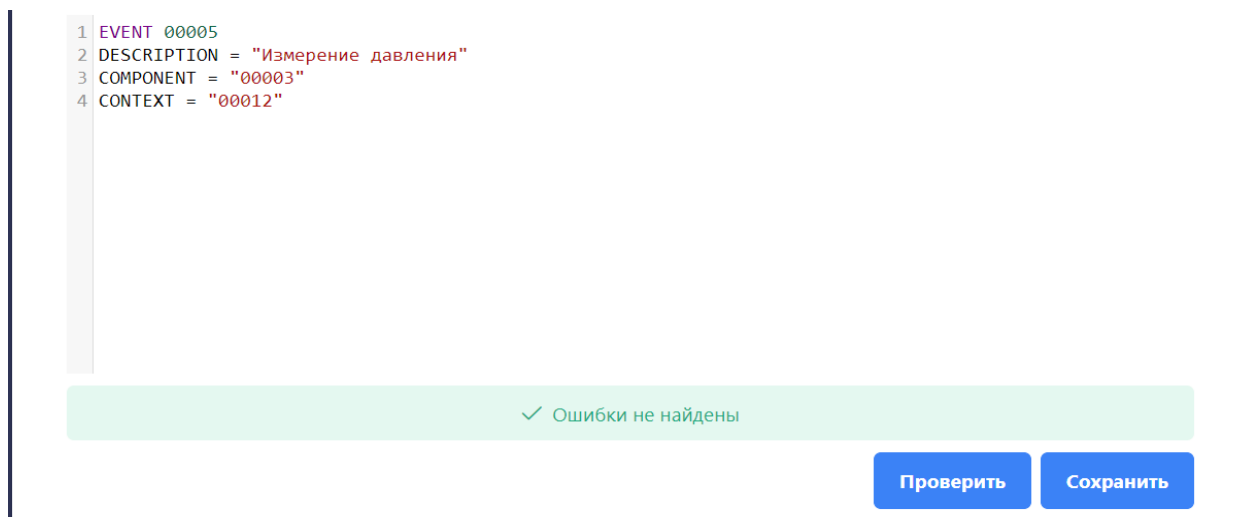


Рисунок 23 – Сообщение об успешной проверке языковой конструкции

Разработка графического интерфейса проводилась с использованием языка гипертекстовой разметки HTML5 и каскадной таблицы стилей CSS3. Для внедрения редактора кода использовалась библиотека CodeMirror с возможностью подсветки синтаксиса. Компоненты кнопок, выпадающих списков и диалогового окна реализованы с помощью библиотеки компонентов PrimeNG.

3.4 Разработка программных средств управления метаданными с помощью языкового представления

3.4.1 Разработка программных средств редактора языковых конструкций объектов метаданных

Согласно разработанным языковым конструкциям объектов метаданных был разработан специализированный редактор, реализация которого включает в себя множество классов, представленных на диаграмме, изображенной на рисунке 24.

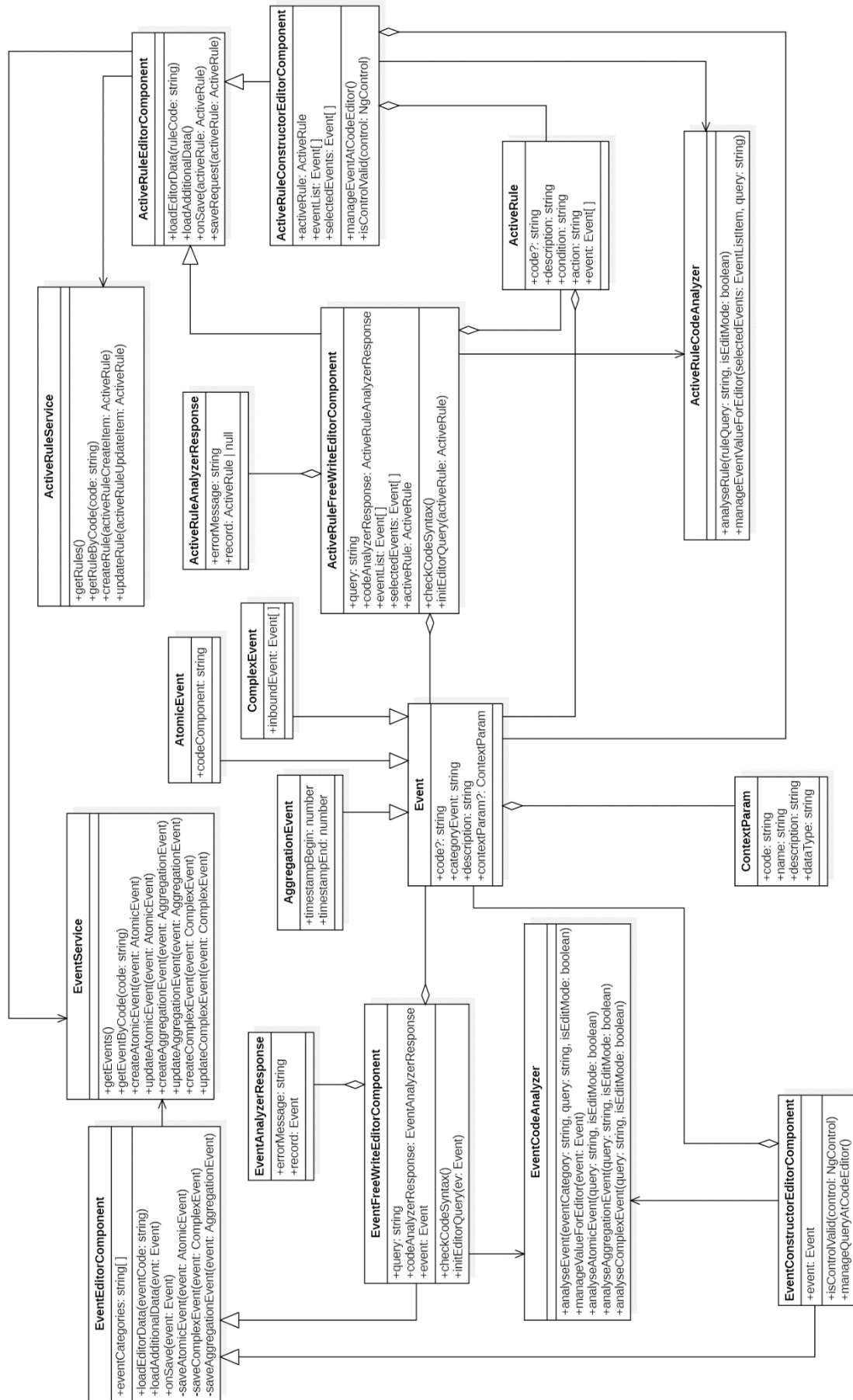


Рисунок 24 – Диаграмма классов редактора языковых конструкций

Редакторы языковых конструкций включают в себя два инструмента: конструктор и инструмент свободного ввода. Оба инструмента представлены отдельными классами, например, `EventFreeWriteEditorComponent` и `EventConstructorEditorComponent` являются классами редактора свободного ввода языковых конструкций событий и редактора режима конструктора соответственно. Данные классы являются дочерними для класса `EventEditorComponent`, от которого получают данные для визуализации.

Ниже представлены спецификации некоторых классов диаграммы, изображенные на рисунках 25, 26 и 27.

Наименование класса: <code>EventFreeWriteEditorComponent</code>
Назначение класса: управление языковыми конструкциями событий в режиме свободного ввода
Поля класса: <ul style="list-style-type: none"> – <code>query: string</code> – значение редактора свободного ввода; – <code>codeAnalyzerResponse: EventAnalyzerResponse</code> – ответ анализатора языковых конструкций события; – <code>event: Event</code> – объект события.
Методы класса: <ul style="list-style-type: none"> – <code>checkCodeSyntax()</code> – проверка корректности языковых конструкций; – <code>initEditorQuery(ev:Event)</code> – инициализация значения редактора свободного ввода.

Рисунок 25 – Спецификация класса «`EventFreeWriteEditorComponent`»

Наименование класса: <code>EventConstructorEditorComponent</code>
Назначение класса: управление языковыми конструкциями событий в режиме конструктора
Поля класса: <ul style="list-style-type: none"> – <code>event: Event</code> – объект события.
Методы класса: <ul style="list-style-type: none"> – <code>isControlValid(control:NgControl)</code> – проверка введенного значения параметра языковой конструкции; – <code>manageQueryAtCodeEditor()</code> – синхронизация введенных данных с редактором свободного ввода.

Рисунок 26 – Спецификация класса «`EventConstructorEditorComponent`»

Наименование класса: EventEditorComponent
Назначение класса: получение информации в редактор для управления событием
Поля класса: <ul style="list-style-type: none"> – eventCategories: string[] – список типов событий.
Методы класса: <ul style="list-style-type: none"> – loadEditorData(eventCode:string) – получение информации о событии; – loadAdditionalData(evnt:Event) – получение дополнительной информации о событии; – onSave(event:Event) – сохранение события; – saveAtomicEvent(event:AtomicEvent) – запрос на сохранение атомарного события; – saveComplexEvent(event:ComplexEvent) – запрос на сохранение сложного события; – saveAggregationEvent(event:AggregationEvent) – запрос на сохранение агрегирующего события.

Рисунок 27 – Спецификация класса «EventEditorComponent»

Каждый редактор свободного ввода оснащен функцией инициализации языковых конструкций `initEditorQuery` на основе пришедшего объекта метаданных. Данный метод запускается в момент открытия редактора или в момент переключения с режима конструктора на свободный ввод. Код данной функции на примере активных правил представлен в листинге 6.

Листинг 6 – Код функции `initEditorQuery()`

```
initEditorQuery(activeRule: ActiveRule){
    this.query = `RULE ${activeRule.code}
        DESCRIPTION = "${activeRule.description}"
        CONDITION = "${activeRule.condition}"
        ACTION = "${activeRule.action}"
        EVENT = "${this.selectedEvents?.map(ev =>
ev.code).toString()}`;
}
```

В свою очередь, конструктор оснащен функцией `manageQueryAtCodeEditor`, которая изменяет значение языковой конструкции в режиме свободного ввода при любом изменении в конструкторе. Код данной функции представлен в листинге 7.

Листинг 7 – Код функции manageQueryAtCodeEditor()

```

manageQueryAtCodeEditor(selectedEvents: EventListItem[], query:
string): string {
    const queryRows = query.split('\n');
    if (query.includes(RuleKeywords.EVENT)) {
        for (let i = 0; i < queryRows.length; i++) {
            const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
            if (rowLexems.includes(RuleKeywords.EVENT)) {
                queryRows.splice(i, 1);
            }
        }
    }
    if (selectedEvents?.length) {
        queryRows.push(`EVENT = "${selectedEvents.map(ev =>
ev.code).join(', ')}"`)
    }
    return queryRows.join('\n');
}

```

Таким образом, редактор языкового представления событий и активных правил оснащен двусторонней синхронизацией между двумя инструментами управления, что позволяет пользователю свободно переключаться между ними, не теряя введенных изменений.

Полный листинг классов редактора представлен в приложении Б.

3.4.2 Разработка анализатора языковых конструкций объектов метаданных

Для сохранения объекта метаданных необходимо убедиться в корректности языковых конструкций, введенных пользователем. Для таких целей был разработан специализированный анализатор языковых конструкций объектов метаданных для редактора свободного ввода.

Спецификации классов EventCodeAnalyzer и ActiveRuleCodeAnalyzer анализаторов языковых шаблонов событий и активных правил приведены на рисунках 28 и 29.

Наименование класса: EventCodeAnalyzer
Назначение класса: анализатор языковых конструкций событий
Поля класса: отсутствуют.
Методы класса: <ul style="list-style-type: none"> – analyseEvent(eventCategory:string, query:string, isEditMode:boolean) – запуск анализа языковых конструкций события; – analyseAtomicEvent(query:string, isEditMode:boolean) – анализ языковых конструкций атомарного события; – analyseAggregationEvent(query:string, isEditMode:boolean) – анализ языковых конструкций агрегирующего события; – analyseComplexEvent(query:string, isEditMode:boolean) – анализ языковых конструкций сложного события; – manageValueForEditor(event:Event) – синхронизация значения конструктора и редактора свободного ввода.

Рисунок 28 – Спецификация класса «EventCodeAnalyzer»

Наименование класса: ActiveRuleCodeAnalyzer
Назначение класса: анализатор языковых конструкций активных правил
Поля класса: отсутствуют.
Методы класса: <ul style="list-style-type: none"> – analyseRule(ruleQuery:string, isEditMode:boolean) – анализ языковых конструкций активных правил; – manageValueForEditor(selectedEvents:Event[], query:string) – синхронизация значения конструктора и редактора свободного ввода.

Рисунок 29 – Спецификация класса «ActiveRuleCodeAnalyzer»

Методы анализа языковых конструкций возвращают объект типа EventAnalyzerResponse в случае анализатора событий и ActiveRuleAnalyzerResponse после проверки корректности конструкций активных правил. Структуры данных классов представлены в листинге 8.

Листинг 8 – Код структуры классов EventAnalyzerResponse и ActiveRuleAnalyzerResponse

```
export class ActiveRuleAnalyzerResponse {
  errorMessage: string;
```

```

    record: ActiveRule | null;
}

export interface EventAnalyzerResponse {
    errorMessage: string;
    record: Event | null;
}

```

Согласно данной структуре объекты представленных типов возвращают либо сообщение об ошибке и нулевое значение объекта метаданных, при наличии некорректно введенных конструкций, либо пустое сообщение об ошибке и сформированный объект метаданных согласно верно введенным пользователем языковым конструкциям.

Типы обрабатываемых ошибок и примеры специальных сообщений о них пользователю на примере анализатора языковых конструкций активных правил представлены в таблице 6.

Таблица 6 – Обрабатываемые ошибки анализатором ActiveRuleCodeAnalyzer

№	Тип ошибки	Текст сообщения пользователю
1	Отсутствие или ошибка в объявлении правила	Ошибка: объявление правила должно начинаться с ключевого слова RULE
2	Отсутствие или ошибка в ключевых словах конструкций параметра правила	Не определены следующие обязательные параметры правила: DESCRIPTION, ACTION
3	Конструкция объявления параметра правила начинается не с ключевого слова, но содержит его	Ошибка в строке 3: строка должна начинаться с ключевого слова ACTION
4	Ошибка в знаке присваивания значения параметра правила	Ошибка в строке 3: пропущен или неверно указан знак присваивания «=» после ключевого слова ACTION
5	Неверно указан тип данных значения параметра	Ошибка в строке 3: неверно указано значение поля ACTION
6	Неверно указан набор событий или присутствует несуществующее событие	Ошибка в строке 4: событие с кодом 0023 не существует

Кроме того, для анализа передается значение `isEditMode`, которое сообщает анализатору находится ли пользователь в режиме редактирования

объекта метаданных или в режиме создания. В случае, если пользователь изменяет уже существующий объект, анализатор добавляет в ответ код объекта.

Полный листинг анализаторов активных правил и событий представлен в приложении В.

4 Контроль качества программных средств

4.1 Планирование процесса контроля качества программных средств

4.1.1 Мероприятия по контролю качества программных средств

Контроль качества – это процесс, осуществляемый на заключительном этапе разработки с целью контроля качества готового программного обеспечения, выявления дефектов, определения их причин и устранения дальнейших дефектов. Процесс направлен на контроль уровня качества конечного продукта и работы команды разработки. [11]

Основная задача контроля качества программного обеспечения – контроль качества, в первую очередь путем тестирования продукции. Для этого создается и проводится серия тестов для комплексной проверки продукции.

Контроль качества также включает в себя проверку и обеспечение достаточного тестового покрытия, чтобы ни одна часть программного обеспечения не осталась непроверенной. В противном случае существует риск, что будут пропущены ошибки, критически важные для долговечности и функциональности программного решения.

Неправильные процедуры при построении программного кода, которые приводят к разрастанию программного кода и задерживают или предотвращают дальнейшее написание, а значит и разработку программных инструментов, можно заранее предотвратить, рассчитав метрики программного кода.

Ошибки, связанные с производительностью разрабатываемого программного обеспечения, можно отслеживать с помощью различных методов тестирования программного обеспечения.

Для контроля качества разработанного программного необходимо провести несколько видов тестирования и оценить качество продукта с помощью специальных метрик.

4.1.2 Обоснование выбора модели тестирования

Методы тестирования можно разделить на ручное и автоматическое тестирование. При ручном тестировании человек проверяет функционал приложения вручную и с помощью специального инструмента для взаимодействия с API [12].

Различают следующие методы тестирования.

Модульное тестирование – метод тестирования программного обеспечения, который заключается в тестировании отдельных методов и функций классов, компонентов и модулей, используемых в программном обеспечении. Автоматизация тестирования модулей обычно не требует больших затрат и может быть выполнена очень быстро серверами непрерывной интеграции.

Интеграционное тестирование – метод тестирования программного обеспечения, в ходе которого проверяется взаимодействие различных модулей и сервисов приложения. В основном, данный метод используется для проверки взаимодействия клиентской и серверной частей посредством API.

Функциональное тестирование – метод тестирования программного обеспечения, направленный на проверку соответствия продукта функциональным требованиям посредством сравнения ожидаемого и фактического результата теста.

Тестирование производительности – метод тестирования, который можно использовать для оценки надежности, скорости и отзывчивости программного обеспечения. Этот метод тестирования можно использовать для определения соответствия приложения требованиям производительности, поиска узких мест, оценки стабильности в периоды пиковой нагрузки и т.д.

Дымчатое тестирование – метод тестирования, который рассматривается как минимальный цикл тестов, выполняемый с целью подтверждения, что сборка установленного приложения запускается и выполняет базовые функции.

Для тестирования программного обеспечения в качестве ручного вида тестирования было выбрано функциональное тестирование, поскольку оно:

- рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом;
- позволяет сфокусироваться и не упустить наиболее важный функционал;
- имитирует фактическое использование системы.

Кроме того, был выбран вид автоматического тестирования – модульное тестирование, поскольку оно позволяет проверить работу сервисов и компонентов приложения отдельно от других. Также, данный метод тестирования сокращает время на создание и отладку продукта.

Тестирование разработанного приложения осуществляется на примере системы, описанной в приложении Г.

4.1.3 Обоснование выбора метрик программного кода

Используя метрики программного кода, разработчики и менеджеры проектов могут охарактеризовать создаваемое или уже существующее программное обеспечение, предсказать объем работ, количественно оценить качество тех или иных проектных решений, разрабатываемых систем и их компонентов, описать сложность и надежность. Метрики также могут использоваться для оценки продуктивности разработки программного обеспечения, а результаты измерений программного кода могут использоваться для корректировки рабочего процесса программистов.

Различают несколько метрик, которые можно использовать для оценки программного кода с разных точек зрения, с разным уровнем усилий и времени, и используя различные методы для расчета результатов.

Примеры метрик программного кода [13]:

- метрики устойчивости и абстрактности компонентов приложения;
- комплексный показатель качества кода
- метрика Холстеда – анализ строк и синтаксических параметров кода программы.

Двумя основными метриками являются метрика Холстеда и метрики устойчивости и абстрактности. Они и были взяты для оценки качества программного кода.

Метрика Холстеда позволяет рассчитать показатели программы на основе операндов, операторов и количестве строк программного кода. С помощью данной метрики можно определить объем работ и возможность расширения функционала программы.

Метрики устойчивости и абстрактности компонентов приложения позволяют определить назначение модулей и компонентов программы и выявить их необходимость.

4.2 Тестирование программных средств

4.2.1 Функциональное тестирование

После того как программные средства были реализованы, проведено функциональное тестирование. Рассмотрим тестирование страницы представления активных правил. Результаты тестирования приведены в приложении Д. План тестирования представлен в таблице 7.

Таблица 7 – План функционального тестирования

Вариант использования	Тест	Результаты
Создать событие	Пользователь создает атомарное событие изменения температуры	Событие создано, выведено соответствующее сообщение, присвоен уникальный код 00021 (Рисунок Д.1)
	Пользователь создает атомарное событие изменения температуры за период времени	Событие создано, выведено соответствующее сообщение, присвоен уникальный код 00022 (Рисунок Д.2)

Продолжение таблицы 7

Вариант использования	Тест	Результаты
Создать событие	Пользователь создает сложное событие пожарной опасности	Событие создано, выведено соответствующее сообщение, присвоен уникальный код 00023 (Рисунок Д.3)
	Пользователь пытается создать атомарное событие изменения силы тока, начиная языковую конструкцию с ключевого слова RULE	Событие не создано, выведено сообщение о том, что конструкция должна начинаться с ключевого слова EVENT (Рисунок Д.4)
Изменить событие	Пользователь изменяет параметр контекста на «Дым» события о пожарной опасности	Событие успешно сохранено, выведено соответствующее сообщение (Рисунок Д.5)
	Пользователь пытается изменить параметр контекста события на несуществующий	Событие не сохранено, выведено сообщение о несуществующем параметре контекста (Рисунок Д.6)
Проверить корректность языковых конструкций	Пользователь проверяет корректную языковую конструкцию активного правила	Выведено сообщение о том, что ошибки не найдены (Рисунок Д.7)
	Пользователь проверяет корректную языковую конструкцию события	Выведено сообщение о том, что ошибки не найдены (Рисунок Д.8)
	Пользователь проверяет некорректную языковую конструкцию активного правила	Выведено сообщение об ошибке (Рисунок Д.9)
	Пользователь проверяет некорректную языковую конструкцию события	Выведено сообщение об ошибке (Рисунок Д.10)
Создать активное правило	Пользователь пытается создать активное правило приведения в норму показателя температуры в помещении без действия	Правило не создано, выведено сообщение о том, что параметр действия правила не задан (Рисунок Д.11)

Окончание таблицы 7

Вариант использования	Тест	Результаты
Создать активное правило	Создать активное правило	Пользователь создает активное правило приведения в норму показателя температуры в помещении
Изменить активное правило	Пользователь изменяет условие правила приведения температуры помещения в норму	Правило сохранено, выведено сообщение об успешном сохранении изменений (Рисунок Д.13)
	Пользователь пытается изменить событие правила на несуществующее	Правило не сохранено, выведено сообщение о несуществующем событии (Рисунок Д.14)

Программа успешно прошла все функциональные тесты.

4.2.2 Модульное тестирование

Для обеспечения корректности работы разработанного приложения и проверки ожидаемого поведения программы проведено модульное тестирование. Тестирование модуля `ActiveRule`, разработанного на языке `Angular`, осуществляется с помощью встроенного инструмента `Jasmine` [14]. Полный код модульных тестов приведен в приложении Е. План модульного тестирования представлен в таблице 8.

Таблица 8 – План модульного тестирования

Тестируемый класс	Тест	Ожидаемый результат	Актуальный результат
<code>ActiveRuleComponent</code>	Должен создаваться	Пройден	Пройден (рисунок 30)
	Должно отобразиться диалоговое окно	Пройден	Пройден (рисунок 30)
<code>ActiveRuleListComponent</code>	Должен создаваться	Пройден	Пройден (рисунок 30)

Продолжение таблицы 8

Тестируемый класс	Тест	Ожидаемый результат	Актуальный результат
ActiveRuleListComponent	Должны загрузиться активные правила	Пройден	Пройден (рисунок 30)
ActiveRuleEditorComponent	Должен создаться	Пройден	Пройден (рисунок 30)
	Должно загрузиться активное правило	Пройден	Пройден (рисунок 30)
	Должен переключиться на режим конструктора	Пройден	Пройден (рисунок 30)
	Должно загрузиться активное правило	Пройден	Пройден (рисунок 30)
	Должны загрузиться поля ввода данных	Пройден	Пройден (рисунок 30)
ActiveRuleFreeWriteEditorComponent	Должен переключиться на режим свободного ввода	Пройден	Пройден (рисунок 30)
	Должно загрузиться активное правило	Пройден	Пройден (рисунок 30)
	Должен загрузиться языковой шаблон активного правила	Пройден	Пройден (рисунок 30)

Окончание таблицы 8

Тестируемый класс	Тест	Ожидаемый результат	Актуальный результат
ActiveRuleFreeWriteEditorComponent	Должно выводиться сообщение о корректной языковой конструкции правила при проверке	Пройден	Пройден (рисунок 30)
	Должно выводиться сообщение о некорректной языковой конструкции правила при ошибке по время проверки	Пройден	Пройден (рисунок 30)

На рисунке 30 представлен результат тестирования модуля ActiveRule.



Рисунок 30 – Результат модульного тестирования

Программа успешно прошла все этапы модульного тестирования.

4.3 Оценка качества программного кода

4.3.1 Метрика Холстеда

Метрики Холстеда – это метрики, которые рассчитываются на основе анализа строк исходного кода программы и элементов синтаксиса.

В качестве примера расчета метрик Холстеда рассматривается класс редактора свободного ввода языковых конструкций активных правил. Код данного класса представлен в приложении Б листинге Б.2.

Операторы приведенного класса представлены в таблице 9, операнды представлены в таблице 10.

Таблица 9 – Операторы класса `ActiveRuleEditorComponent`

Оператор	Число вхождений
:	44
?	9
< >	1
=	24
()	39
.	43
=>	7
{ }	38
,	27
;	28
	2
this	36
[]	9
===	1
!	1
\$	8

Продолжение таблицы 9

Оператор	Число вхождений
if	5
subscribe	4
navigate	1
map	2
else	3
setTimeout	1
new	2
return	3
@Component	1
n1 = 25	N1 = 339

Таблица 10 – Операнды ActiveRuleEditorComponent

Операнд	Число вхождений
router	6
codeAnalyzer	3
activeRule	21
activeRuleService	5
null	2
RULE	1
DESCRIPTION	1
CONDITION	1
ACTION	1
EVENT	1
eventService	2
eventList	2
"	3
'Правило успешно сохранено'	1
'Ошибки не найдены'	1

Продолжение таблицы 10

Операнд	Число вхождений
successMessage	3
query	5
selectedEvents	4
isEditMode	7
n2 = 19	N2 = 70

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 339 + 70 = 409.$$

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 25 + 19 = 44.$$

Объем программы рассчитывается по формуле:

$$H_V = N * \log_2(n) = 409 * 5,46 = 2233,14.$$

Потенциальный объем программы:

$$V^* = (2 + n') * \log_2(2 + n') = 8 * \log_2(8) = 24.$$

Теоретический уровень программы рассчитывается по формуле:

$$L' = V^* / H_V = 24 / 2233,14 = 0.011.$$

Сложность программы рассчитывается по формуле:

$$HD = (n1 / 2) * (N2 / n2) = (25 / 2) * (70 / 19) = 46.05.$$

Все полученные показатели программы находятся в пределах нормы и удовлетворяют требованиям по метрикам программного кода.

4.3.2 Устойчивость и абстрактность

Соотношение между абстракцией и устойчивостью было рассчитано для разработанных программных средств. Соответствующие показатели рассчитывались на основе модулей, в которых организованы программные

классы приложения. В приложении организовано 4 программных модуля: App, Shared, Event и ActiveRule.

Формула расчета устойчивости для каждого модуля:

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

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

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

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

Устойчивость каждого модуля системы:

$$I_1 = \frac{3}{3+0} = 1;$$

$$I_2 = \frac{0}{2+0} = 0;$$

$$I_3 = \frac{1}{1+1} = 0.5;$$

$$I_4 = \frac{2}{1+2} = 0.67.$$

Формула расчета абстрактности каждого модуля системы:

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

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

Nc – количество классов в компоненте;

Na – количество абстрактных классов в компоненте;

Устойчивость рассчитана для каждого модуля сервиса:

$$A_1 = \frac{0}{2} = 0;$$

$$A_2 = \frac{1}{3} = 0.33;$$

$$A_3 = \frac{1}{9} = 0.11;$$

$$A_4 = \frac{1}{9} = 0.11.$$

Согласно рассчитанным метрикам устойчивости и абстрактности была найдена величина нормированного расстояния от главной последовательности.

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

где A – абстрактность;

I – устойчивость.

Нормированное расстояние для каждого модуля:

$$D'_1 = |0 + 1 - 1| = 0;$$

$$D'_2 = |0.33 + 0 - 1| = 0.67;$$

$$D'_3 = |0.11 + 0.5 - 1| = 0.39;$$

$$D'_4 = |0.11 + 0.67 - 1| = 0.22.$$

Все рассчитанные метрики можно изобразить на диаграмме, представленной на рисунке 31. На данном графике располагаются зоны неприятия, где модули системы максимально устойчивы и не абстрактны, и бесполезности, где компоненты по большей мере неустойчивы, но абстрактны. Кроме того, изображена главная последовательность, к которой должны стремиться все модули системы.

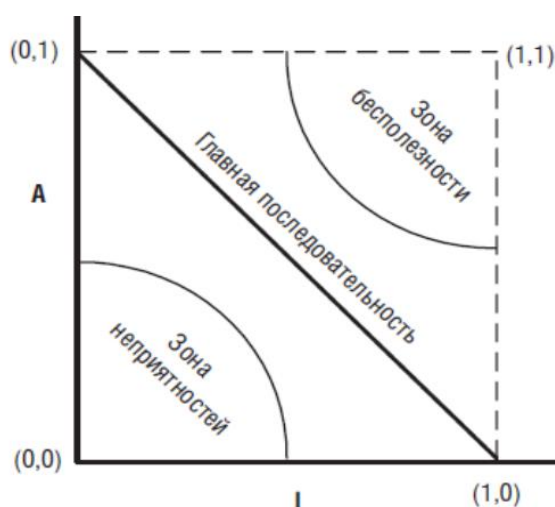


Рисунок 31 – Пример графика значений главной последовательности

Согласно вычисленным значениям метрик, был построен аналогичный график с указанными на нем значениями модулей системы, изображенный на рисунке 32.

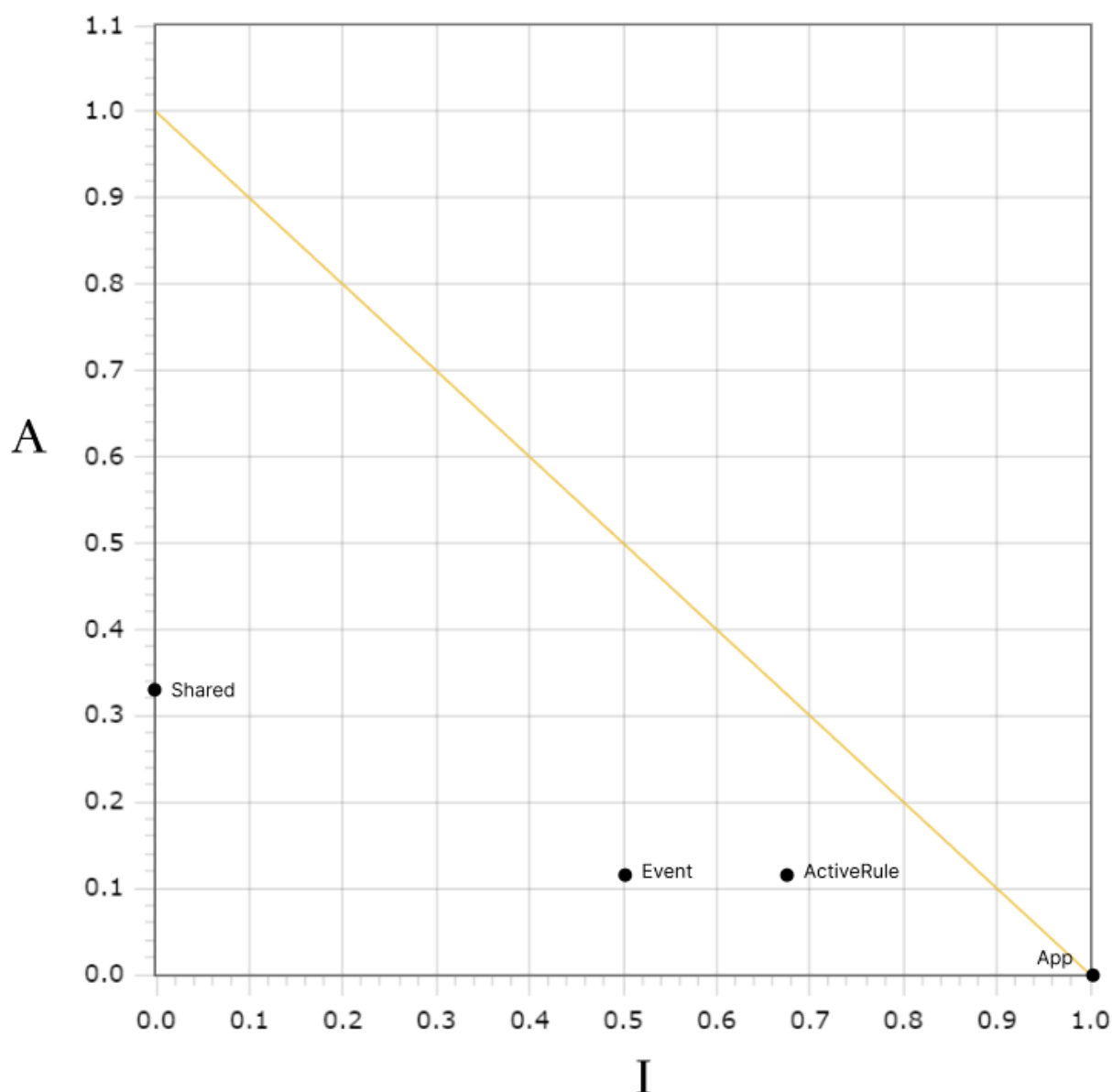


Рисунок 32 – Метрики кода на графике значений главной последовательности

Главный модуль приложения App характеризуется как не устойчивый и не абстрактный, поэтому располагается на главной последовательности. Модуль Shared располагается далеко от главной последовательности и характеризуется как слишком устойчивый. Это вызвано тем, что данный модуль реализует общие функции сервиса и не зависит от других модулей и классов приложения.

Модули Event и ActiveRule расположены достаточно близко к главной последовательности. Это означает, что соотношение абстрактности и устойчивости модулей является верным.

Заключение

Во время выполнения выпускной квалификационной работы были разработаны программные средства поддержки языкового представления метаданных для сервиса потоковой обработки событий.

Был подробно проанализирован сервис потоковой обработки событий и исполнения активных правил, для которого разрабатывались средства поддержки языкового представления. Были рассмотрены назначение и функциональные возможности системы, ее микросервисная событийно-ориентированная архитектура и схема ее работы. Также, были описаны структура, жизненный цикл метаданных сервиса, а также задачи управления ими. Кроме того, были определены цели разработки и назначение программных средств, был описан перечень основных автоматизируемых функций. После было произведено планирование разработки, а также функционально-стоимостной анализ разработки программных средств. Затраты на проект составили 89500 рублей. Начало работ было определено как 12 апреля, а окончание работ по проекту запланировано на 10 июня текущего года. Затраты трудовых ресурсов составили 392 часа.

На следующем этапе были определены функциональные требования, требования к архитектуре и пользовательскому интерфейсу программных средств. В качестве языка программирования для реализации программных средств был выбран язык TypeScript под управлением фреймворка Angular, а также другие технологии разработки, интегрированные в рабочий процесс. Помимо этого, были описаны требования к аппаратно-программному обеспечению пользователя для работы с программными средствами языкового представления метаданными.

В процессе этапа разработки была построена архитектура развертывания приложения, разработаны языковые конструкции объектов метаданных. Кроме того, были реализованы два инструмента управления объектами метаданных: редактор и конструктор языковых шаблонов. Для проверки корректности языковых конструкций был разработан специализированный анализатор.

Разработка программных средств проводилась в среде разработки WebStorm под управлением операционной систем Windows 10. Программные средства создавались в виде web-приложения, что позволяет запускать их в любом современном браузере.

Разработанное программное средство было протестировано при помощи функционального и модульного тестирования. Все тесты приложение прошло успешно. Кроме того, были вычислены метрики программного кода: устойчивость, абстрактность и метрика Холстеда. Данные метрики показали достаточный уровень разработанных модулей системы.

Список использованных источников

1. Adaikkalavan R. Active Authorization Rules for Enforcing Role-Based Access Control and its Extensions / R. Adaikkalavan, S. Chakravarthy // 21st International Conference on Data Engineering Workshops – Tokyo : Изд-во IEEE, 2005. – С. 220 – 222.

2. Шибанов С.В., Шлепнев Я.С. Сервис потоковой обработки событий и исполнения активных правил // Математическое моделирование и суперкомпьютерные технологии. Труды XXI Международной конференции (Н. Новгород, 22–26 ноября 2021 г.) / Под ред. проф. Д.В. Баландина. – Нижний Новгород: Изд-во Нижегородского госуниверситета, 2021. - С. 403-407.

3. Шибанов С. В., Скоробогатько А. А., Э. В. Лысенко. Интегрированная модель активных правил / С. В. Шибанов, А. А. Скоробогатько, Э. В. Лысенко // Математическое и программное обеспечение систем в промышленной и социальной сферах: междунар. сб. науч. трудов – Магнитогорск : Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2011. – С. 41 – 46.

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

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

6. Паттерны для новичков: MVC vs MVP vs MVVM [Электронный ресурс]. URL: <https://habr.com/ru/post/215605/> (дата обращения: 18.05.2023).

7. Introduction to the Angular docs // Angular [Электронный ресурс]. URL: <https://angular.io/docs> (дата обращения: 18.05.2023).

8. Основы Sass // Sass [Электронный ресурс]. URL: <https://sass-scss.ru/guide/> (дата обращения: 18.05.2023).

9. CodeMirror 5 [Электронный ресурс]. URL: <https://codemirror.net/5/index.html> (дата обращения: 20.05.2023).

10. Как установить Docker на Windows 10 [Электронный ресурс]. URL: <https://efsol.ru/manuals/docker-win-10.html> (дата обращения: 21.05.2023).

11. Различия между контролем качества и обеспечением качества в IT // Wezom [Электронный ресурс]. URL: <https://wezom.com.ua/blog/razlichija-mezhdu-kontrolem-kachestva-i-obespecheniem-kachestva-v-it> (дата обращения: 12.06.2023).
12. Различные виды тестирования ПО // Atlassian [Электронный ресурс]. URL: <https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing> (дата обращения: 12.06.2023).
13. Выпускная квалификационная работа: стандарты и программные технологии. Учебно-методическое пособие для бакалавров по направлениям 09.03.04 "Программная инженерия", 09.03.02 «Информационные системы и технологии» / под ред. П. П. Макарычева. – Пенза: Изд-во ПГУ, 2020. – 182с.
14. Тестирование Angular компонентов с использованием Jasmine: часть 1 [Электронный ресурс]. URL: <https://code.tutsplus.com/ru/tutorials/testing-components-in-angular-using-jasmine--cms-29165> (дата обращения: 14.06.2023).

Приложение А
ГЛОССАРИЙ
(справочное)

ECA (Event condition action) – структура активного правила в событийно-ориентированной архитектуре, которое состоит из события, условия и действия;

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

API (Application Programming Interface) – механизмы, помогающие программным модулям взаимодействовать друг с другом посредством набора определений и протоколов;

DOM (Document Object Model) – программный интерфейс, представляющий древовидную структуру программных документов типа HTML и XML.

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

Фреймворк – программная платформа, которая задает структуру и определяет правила написания кода.

Рефакторинг – процесс изменения кода программы, не влияющий на изменение внешнего поведения системы, но направленный на облегчение понимания ее работы.

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

Интерсептор – вид классов в программировании, позволяющий перехватывать HTTP-запросы и HTTP-ответы и вносить в них определенные изменения.

Приложение Б
ЛИСТИНГ РЕДАКТОРОВ ЯЗЫКОВЫХ КОНСТРУКЦИЙ
(рекомендуемое)

Листинг Б.1 – Файл active-rule-editor.component.ts

```

import { Component, OnInit } from "@angular/core";
import { AnalyzerResponse, ActiveRuleCodeAnalyzer } from
"./active-rule-code-analyzer";
import { AppRoutesEnum } from "../app-routes.enum";
import { Router } from "@angular/router";
import { ActiveRuleService } from "../active-rule.service";
import { EventService } from "../event/event.service";
import { EventListItem } from "../event/event";
import { Observable } from "rxjs";
import { ActiveRule, ActiveRuleEventTypeBind } from "../active-
rule";
import { NgControl, NgForm } from "@angular/forms";

@Component({
  selector: 'app-active-rule-editor',
  templateUrl: 'active-rule-editor.component.html',
  styleUrls: ['active-rule-editor.component.scss']
})
export class ActiveRuleEditorComponent implements OnInit {
  isEditMode: boolean = false;
  eventList: EventListItem[] = [];
  selectedEvents: EventListItem[] | undefined = [];
  initialEventsList: EventListItem[] | undefined = [];
  activeRule: ActiveRule = new ActiveRule();

  constructor(private codeAnalyzer: ActiveRuleCodeAnalyzer,
    private router: Router,
    private activeRuleService: ActiveRuleService,
    private eventService: EventService) {

  }

  ngOnInit(): void {
    const currentRouteParts = this.router.url.split('/');

    this.isEditMode = currentRouteParts[2] ===
AppRoutesEnum.EDIT;

    this.loadAdditionalData();
    if (this.isEditMode) {
      this.loadEditorData(currentRouteParts[3]);
    } else {
      this.query = '';
    }
  }

  loadEditorData(ruleCode: string) {

this.activeRuleService.getRuleByCode(ruleCode).subscribe(
  rule => {
    this.activeRule = rule;
    this.selectedEvents = rule.event;
  }
);

```

```

        this.initialEventsList = rule.event;
        this.initEditorQuery(rule);
    }
}

loadAdditionalData() {
    this.eventService.getEvents().subscribe(
        events => {
            this.eventList = events.map(e => {
                return {
                    code: e.code,
                    description: e.description,
                    categoryEvent: e.categoryEvent
                }
            });
        }
    )
}

onSave() {
    this.codeAnalyzerResponse =
this.codeAnalyzer.analyseRule(this.query, this.isEditMode);

    if (!this.codeAnalyzerResponse.errorMessage) {

this.saveRequest(this.codeAnalyzerResponse.record).subscribe(
    activeRule => {
        this.selectedEvents?.forEach(ev => {
            if
(!this.initialEventsList?.includes(ev)) {

this.activeRuleService.addEventsToRule({
                    codeRule: activeRule.code,
                    codeEvent: ev.code,
                    typeBind:
ActiveRuleEventTypeBind.EVENT_TO_RULE
                }).subscribe();
            }
        });
        this.initialEventsList?.forEach(ev => {
            if (!this.selectedEvents?.includes(ev))
{

this.activeRuleService.deleteEventsToRule({
                    codeRule: activeRule.code,
                    codeEvent: ev.code,
                    typeBind:
ActiveRuleEventTypeBind.EVENT_TO_RULE
                }).subscribe();
            }
        });
        this.successMessage = 'Правило успешно
сохранено';
        if (this.isEditMode) {

```

```

        setTimeout(() => {
            // @ts-ignore

this.loadEditorData(activeRule.code);
            }, 3000)

        } else {

this.router.navigate([`/${AppRoutesEnum.ACTIVE_RULE_SCRIPT}/${AppRoutesEnum.EDIT}/${activeRule.code}`]);
        }
    }
)
}

private saveRequest(activeRule: ActiveRule | null |
undefined): Observable<ActiveRule> {
    if (this.isEditMode) {
        return this.activeRuleService.updateRule({
            code: activeRule?.code,
            condition: activeRule?.condition,
            action: activeRule?.action,
            description: activeRule?.description
        });
    } else {
        return this.activeRuleService.createRule({
            condition: activeRule?.condition,
            action: activeRule?.action,
            description: activeRule?.description
        });
    }
}
}

```

Листинг Б.2 – Файл active-rule-free-write-editor.component.ts

```

import {ChangeDetectionStrategy, Component, Input, OnInit} from
"@angular/core";
import {AnalyzerResponse, ActiveRuleCodeAnalyzer} from
"../active-rule-code-analyzer";
import {EventListItem} from "../../event/event";
import {ActiveRule, ActiveRuleEventTypeBind} from "../../active-
rule";
import {Router} from "@angular/router";
import {ActiveRuleService} from "../../active-rule.service";
import {AppRoutesEnum} from "../../app-routes.enum";
import {Observable} from "rxjs";

@Component({
    templateUrl: 'free-write.component.html',
    selector: 'app-active-rule-free-write-editor',
    changeDetection: ChangeDetectionStrategy.OnPush
})

```

```

export class ActiveRuleFreeWriteComponent implements OnInit {
  codeMirrorOptions: any = {
    mode: "text/x-mysql",
    indentWithTabs: true,
    smartIndent: true,
    lineNumbers: true,
    lineWrapping: false,
    extraKeys: {"Ctrl-Space": "autocomplete"},
    gutters: ["CodeMirror-linenumbers", "CodeMirror-
foldgutter"],
    autoCloseBrackets: true,
    matchBrackets: true,
    lint: true
  };

  query: string = ``;

  codeAnalyzerResponse: AnalyzerResponse | null | undefined;
  successMessage: string = '';
  isEditMode: boolean = false;
  @Input() eventList: EventListItem[] = [];
  selectedEvents: EventListItem[] | undefined = [];
  @Input() initialEventsList: EventListItem[] | undefined =
[];
  @Input() activeRule!: ActiveRule;

  constructor(private codeAnalyzer: ActiveRuleCodeAnalyzer,
               private router: Router,
               private activeRuleService: ActiveRuleService) {
  }

  ngOnInit(): void {
    const currentRouteParts = this.router.url.split('/');

    this.isEditMode = currentRouteParts[2] ===
AppRoutesEnum.EDIT;
    if (this.isEditMode) {
      this.initEditorData();
    } else {
      this.query = '';
    }
  }

  initEditorData() {
    this.selectedEvents = this.activeRule?.event;
    this.initialEventsList = this.activeRule?.event;
    this.query = `RULE ${this.activeRule?.code}
DESCRIPTION = "${this.activeRule?.description}"
EVENT = "${this.activeRule?.event?.map(ev =>
ev.code).toString()}"
CONDITION = "${this.activeRule?.condition}"
ACTION = "${this.activeRule?.action}"`;
  }

  setEditorContent() {

```

```

        this.codeAnalyzerResponse = null;
    }

    checkCodeSyntax() {
        this.codeAnalyzerResponse =
this.codeAnalyzer.analyseRule(this.query, this.isEditMode);
        if (this.codeAnalyzerResponse.record) {
            this.successMessage = 'Ошибки не найдены';
        }
    }
}

```

Листинг Б.3 – Файл active-rule-constructor-editor.component.ts

```

import { Component, OnInit } from "@angular/core";
import { AnalyzerResponse, ActiveRuleCodeAnalyzer } from
"./active-rule-code-analyzer";
import { AppRoutesEnum } from "../app-routes.enum";
import { Router } from "@angular/router";
import { ActiveRuleService } from "../active-rule.service";
import { EventService } from "../event/event.service";
import { EventListItem } from "../event/event";
import { Observable } from "rxjs";
import { ActiveRule, ActiveRuleEventTypeBind } from "../active-
rule";
import { NgControl, NgForm } from "@angular/forms";

@Component({
    selector: 'app-active-rule-editor',
    templateUrl: 'active-rule-editor.component.html',
    styleUrls: ['active-rule-editor.component.scss']
})
export class ActiveRuleEditorComponent implements OnInit {
    eventList: EventListItem[] = [];
    selectedEvents: EventListItem[] | undefined = [];
    activeRule: ActiveRule = new ActiveRule();

    constructor(private codeAnalyzer: ActiveRuleCodeAnalyzer,
                private router: Router,
                private activeRuleService: ActiveRuleService,
                private eventService: EventService) {

    }

    manageEventAtCodeEditor() {
        this.query =
this.codeAnalyzer.manageEventValueForEditor(this.selectedEvents,
this.query);
    }

    isControlValid(control: NgControl, form: NgForm): boolean {
        if (form) {
            return <boolean>control.valid;
        }
        return true;
    }
}

```

Листинг Б.4 – Файл event-editor.component.ts

```

import { Component, OnInit } from "@angular/core";
import { EventAnalyzerResponse, EventCodeAnalyzer } from "../event-
code-analyzer";
import { AppRoutesEnum } from "../../app-routes.enum";
import { Router } from "@angular/router";
import {
    AggregationEvent,
    AggregationEventUpdateItem,
    AtomicEvent,
    AtomicEventUpdateItem,
    ComplexEvent,
    ComplexEventUpdateItem,
    ContextParam,
    EventCategory,
    EventListItem,
    EventPageItem
} from "../event";
import { forkJoin } from "rxjs";
import { NgControl, NgForm } from "@angular/forms";
import { EventService } from "../event.service";
import { ContextParamService } from "../context-param.service";
import { ComponentItem, ComponentService } from
"../component.service";

@Component({
    selector: 'app-event-editor',
    templateUrl: 'event-editor.component.html',
    styleUrls: ['event-editor.component.scss']
})
export class EventEditorComponent implements OnInit {
    isEditMode: boolean = false;
    event!: EventPageItem;
    EventCategory = EventCategory;
    eventCategories = [EventCategory.ATOMIC,
EventCategory.AGGREGATION, EventCategory.COMPLEX];

    contextParams: ContextParam[] = [];
    events: EventListItem[] = [];
    initialEvents: EventListItem[] = [];
    initialContext: string = '';
    components: ComponentItem[] = [];
    selectedComponent: string = '';

    constructor(private codeAnalyzer: EventCodeAnalyzer, private
componentService: ComponentService,
                private router: Router, private
contextParamService: ContextParamService,
                private eventService: EventService) {
    }

    ngOnInit(): void {
        const currentRouteParts = this.router.url.split('/');

```

```

        this.isEditMode = currentRouteParts[2] ===
AppRoutesEnum.EDIT;

        if (this.isEditMode) {
            this.loadEditorData(currentRouteParts[3]);
        } else {
            this.query = '';
            this.event = {} as EventPageItem;
            this.loadAdditionalData(this.event);
        }
    }

loadEditorData(eventCode: string) {
    this.eventService.getEventByCode(eventCode).subscribe(
        evnt => {
            this.event = evnt;
            this.initialContext = evnt.contextParam.code;
            if (evnt.inboundEvent) {
                this.initialEvents = evnt.inboundEvent;
            }
            this.loadAdditionalData(evnt);
        }
    )
}

loadAdditionalData(evnt: EventPageItem) {
    forkJoin([
        this.eventService.getEvents(),
        this.contextParamService.getParams(),
        this.componentService.getComponents()
    ]).subscribe(
        ([events, params, components]) => {
            this.contextParams = params;
            this.events = events.map(ev => {
                return {
                    code: ev.code,
                    description: ev.description,
                    categoryEvent: ev.categoryEvent
                }
            });
            this.components = components;
            components.forEach(comp => {
                if (comp.event.find(ev => ev.code ===
this.event.code)) {
                    this.selectedComponent = comp.code;
                }
            });
            this.initEditorQuery(evnt);
        }
    );
}

onSave() {
    this.codeAnalyzerResponse =

```

```

this.codeAnalyzer.analyseEvent(this.event.categoryEvent,
this.query, this.isEditMode);

    if (!this.codeAnalyzerResponse.errorMessage) {
        switch (this.event.categoryEvent) {
            case EventCategory.ATOMIC:

this.saveAtomicEvent(this.codeAnalyzerResponse.record as
AtomicEvent);

                break;
            case EventCategory.AGGREGATION:

this.saveAggregationEvent(this.codeAnalyzerResponse.record as
AggregationEvent);

                break;
            case EventCategory.COMPLEX:

this.saveComplexEvent(this.codeAnalyzerResponse.record as
ComplexEvent);

                break;
        }
    }

    }

private saveAtomicEvent(event: AtomicEvent) {
    if (this.isEditMode) {
        let updateItem!: AtomicEventUpdateItem;
        if (this.initialContext !== event.contextParamCode)
{
            updateItem = {
                // @ts-ignore
                code: event.code,
                description: event.description,
                contextParamCode: event.contextParamCode
            }
        } else {
            updateItem = {
                // @ts-ignore
                code: event.code,
                description: event.description
            }
        }
    }

this.eventService.updateAtomicEvent(updateItem).subscribe((event
) => {
    this.successMessage = 'Событие успешно
сохранено';
    this.loadEditorData(event.code);
    });
} else {
    this.eventService.createAtomicEvent({
        description: event.description,
        contextParamCode: event.contextParamCode,
        codeComponent: event.codeComponent

```

```

        }).subscribe((event) => {
            this.successMessage = 'Событие успешно
сохранено';

this.router.navigate(['`/${AppRoutesEnum.EVENT_SCRIPT}/${AppRoute
sEnum.EDIT}/${event.code}`]);
        });
    }

    private saveComplexEvent(event: ComplexEvent) {
        if (this.isEditMode) {
            let updateItem!: ComplexEventUpdateItem;
            if (this.initialContext !== event.contextParamCode)
{
                updateItem = {
                    // @ts-ignore
                    code: event.code,
                    description: event.description,
                    contextParamCode: event.contextParamCode,
                    templateEvent: event.templateEvent
                }
            } else {
                updateItem = {
                    // @ts-ignore
                    code: event.code,
                    description: event.description,
                    templateEvent: event.templateEvent
                }
            }
        }

        this.eventService.updateComplexEvent(updateItem).subscribe(
            (ev) => {
                this.afterSaveComplexEvent(ev.code);
            }
        )
    } else {
        this.eventService.createComplexEvent({
            description: event.description,
            contextParamCode: event.contextParamCode,
            templateEvent: event.templateEvent
        }).subscribe(
            (ev) => {
                this.afterSaveComplexEvent(ev.code);
            }
        );
    }
}

    private afterSaveComplexEvent(codeComplex: string) {
        this.event.inboundEvent?.forEach(ev => {
            if (!this.initialEvents?.includes(ev)) {
                this.eventService.bindEventToComplex({
                    codeComplex: codeComplex,
                    codeAnother: ev.code

```

```

        }).subscribe();
    }
});
this.initialEvents?.forEach(ev => {
    if (!this.event.inboundEvent?.includes(ev)) {
        this.eventService.unbindEventToComplex({
            codeComplex: codeComplex,
            codeAnother: ev.code
        }).subscribe();
    }
});
this.successMessage = 'Событие успешно сохранено';
setTimeout(() => {
    this.loadEditorData(codeComplex);

this.router.navigate([`/${AppRoutesEnum.EVENT_SCRIPT}/${AppRoute
sEnum.EDIT}/${codeComplex}`]);
    }, 1500);
}

private saveAggregationEvent(event: AggregationEvent) {
    if (this.isEditMode) {
        let updateItem!: AggregationEventUpdateItem;
        if (this.initialContext !== event.contextParamCode)
{
            updateItem = {
                // @ts-ignore
                code: event.code,
                description: event.description,
                contextParamCode: event.contextParamCode,
                timestampEnd: event.timestampEnd,
                timestampBegin: event.timestampBegin,
                aggregationQuery: event.aggregationQuery
            }
        } else {
            updateItem = {
                // @ts-ignore
                code: event.code,
                description: event.description,
                timestampEnd: event.timestampEnd,
                timestampBegin: event.timestampBegin,
                aggregationQuery: event.aggregationQuery
            }
        }
    }

this.eventService.updateAggregationEvent(updateItem).subscribe((
event) => {
    this.successMessage = 'Событие успешно
сохранено';
    this.loadEditorData(event.code);
});
} else {
    this.eventService.createAggregationEvent({
        description: event.description,
        contextParamCode: event.contextParamCode,

```

```

        timestampEnd: event.timestampEnd,
        timestampBegin: event.timestampBegin,
        aggregationQuery: event.aggregationQuery
    }).subscribe((event) => {
        this.successMessage = 'Событие успешно
сохранено';

this.router.navigate(['`/${AppRoutesEnum.EVENT_SCRIPT}/${AppRoute
sEnum.EDIT}/${event.code}`]);
    });
}
}
}
}

```

Листинг Б.5 – Файл event-free-write-editor.component.ts

```

import { Component, OnInit } from "@angular/core";
import { EventAnalyzerResponse, EventCodeAnalyzer } from "../event-
code-analyzer";
import { AppRoutesEnum } from "../../app-routes.enum";
import { Router } from "@angular/router";
import {
    AggregationEvent,
    AggregationEventUpdateItem,
    AtomicEvent,
    AtomicEventUpdateItem,
    ComplexEvent,
    ComplexEventUpdateItem,
    ContextParam,
    EventCategory,
    EventListItem,
    EventPageItem
} from '../event';
import { forkJoin } from "rxjs";
import { NgControl, NgForm } from "@angular/forms";
import { EventService } from "../event.service";
import { ContextParamService } from "../context-param.service";
import { ComponentItem, ComponentService } from
"../component.service";

@Component({
    selector: 'app-event-editor',
    templateUrl: 'event-editor.component.html',
    styleUrls: ['event-editor.component.scss']
})
export class EventFreeWriteEditorComponent implements OnInit {
    codeMirrorOptions: any = {
        mode: "text/x-mysql",
        indentWithTabs: true,
        smartIndent: true,
        lineNumbers: true,
        lineWrapping: false,
        extraKeys: {"Ctrl-Space": "autocomplete"},
        gutters: ["CodeMirror-linenumbers", "CodeMirror-

```

```

foldgutter"],
    autoCloseBrackets: true,
    matchBrackets: true,
    lint: true
  });

  query: string = ``;

  codeAnalyzerResponse: EventAnalyzerResponse | null |
undefined;
  successMessage: string = '';
  event!: EventPageItem;
  EventCategory = EventCategory;

  contextParams: ContextParam[] = [];
  events: EventListItem[] = [];
  initialEvents: EventListItem[] = [];
  initialContext: string = '';
  components: ComponentItem[] = [];
  selectedComponent: string = '';

  constructor(private codeAnalyzer: EventCodeAnalyzer, private
componentService: ComponentService,
               private router: Router, private
contextParamService: ContextParamService,
               private eventService: EventService) {

  }

  ngOnInit(): void {
    const currentRouteParts = this.router.url.split('/');

    this.isEditMode = currentRouteParts[2] ===
AppRoutesEnum.EDIT;
  }

  setEditorContent() {
    this.codeAnalyzerResponse = null;
  }

  checkCodeSyntax() {
    this.codeAnalyzerResponse =
this.codeAnalyzer.analyseEvent(this.event.categoryEvent,
this.query, this.isEditMode);
    if (this.codeAnalyzerResponse.record) {
      this.successMessage = 'Ошибки не найдены';
    }
  }

  initEditorQuery(ev: EventPageItem) {
    switch (ev.categoryEvent) {
      case EventCategory.ATOMIC:
        this.query = `EVENT ${ev.code}
DESCRIPTION = "${ev.description}"
COMPONENT = "${this.selectedComponent}"

```

```

CONTEXT = "${ev.contextParam?.code}"`;
        break;
        case EventCategory.AGGREGATION:
            this.query = `EVENT ${ev.code}`
DESCRIPTION = "${ev.description}"
CONTEXT = "${ev.contextParam?.code}"
AGGREGATION_QUERY = "${ev.aggregationQuery}"
TIMESTAMP_BEGIN = ${ev.timestampBegin}
TIMESTAMP_END = ${ev.timestampEnd}`;
        break;
        case EventCategory.COMPLEX:
            this.query = `EVENT ${ev.code}`
DESCRIPTION = "${ev.description}"
CONTEXT = "${ev.contextParam?.code}"
TEMPLATE_EVENT = "${ev.templateEvent}"
EVENTS = "${ev.inboundEvent?.map(ev => ev.code).join(',')}"`;
        break;
    }

}

}

```

Листинг Б.6 – Файл event-creator-editor.component.ts

```

import {Component, OnInit} from "@angular/core";
import {EventAnalyzerResponse, EventCodeAnalyzer} from "../event-
code-analyzer";
import {AppRoutesEnum} from "../../app-routes.enum";
import {Router} from "@angular/router";
import {
    AggregationEvent,
    AggregationEventUpdateItem,
    AtomicEvent,
    AtomicEventUpdateItem,
    ComplexEvent,
    ComplexEventUpdateItem,
    ContextParam,
    EventCategory,
    EventListItem,
    EventPageItem
} from '../event';
import {forkJoin} from "rxjs";
import {NgControl, NgForm} from "@angular/forms";
import {EventService} from "../event.service";
import {ContextParamService} from "../context-param.service";
import {ComponentItem, ComponentService} from
"../component.service";

@Component({
    selector: 'app-event-editor',
    templateUrl: 'event-editor.component.html',
    styleUrls: ['event-editor.component.scss']
})
export class EventConstructorEditorComponent implements OnInit {

```

```

    successMessage: string = '';
    isEditMode: boolean = false;
    event!: EventPageItem;
    EventCategory = EventCategory;
    eventCategories = [EventCategory.ATOMIC,
EventCategory.AGGREGATION, EventCategory.COMPLEX];

    contextParams: ContextParam[] = [];
    events: EventListItem[] = [];
    initialEvents: EventListItem[] = [];
    initialContext: string = '';
    components: ComponentItem[] = [];
    selectedComponent: string = '';

    constructor(private codeAnalyzer: EventCodeAnalyzer, private
componentService: ComponentService,
                private router: Router, private
contextParamService: ContextParamService,
                private eventService: EventService) {
    }

    ngOnInit(): void {
        const currentRouteParts = this.router.url.split('/');

        this.isEditMode = currentRouteParts[2] ===
AppRoutesEnum.EDIT;
    }

    manageParamAtCodeEditor() {
        this.query =
this.codeAnalyzer.manageParamValueForEditor(this.event.contextPa
ram, this.query);
    }

    manageEventsAtCodeEditor() {
        this.query =
this.codeAnalyzer.manageEventsValueForEditor(this.event.inboundE
vent, this.query);
    }

    manageComponentAtCodeEditor() {
        this.query =
this.codeAnalyzer.manageComponentValueForEditor(this.selectedCom
ponent, this.query);
    }

    isControlValid(control: NgControl, form: NgForm): boolean {
        if (form) {
            return <boolean>control.valid;
        }
        return true;
    }
}

```

Приложение В
ЛИСТИНГ АНАЛИЗАТОРОВ ЯЗЫКОВЫХ КОНСТРУКЦИЙ
(рекомендуемое)

Листинг В.1 – Файл event-code-analyzer.ts

```

import {Injectable} from "@angular/core";
import {AggregationEvent, AtomicEvent, ComplexEvent,
ContextParam, EventCategory, EventListItem} from '../event';

@Injectable({providedIn: 'root'})
export class EventCodeAnalyzer {

    analyseEvent(eventCategory: EventCategory, query: string,
isEditMode: boolean): EventAnalyzerResponse {
        switch (eventCategory) {
            case EventCategory.ATOMIC:
                return this.analyseAtomicEvent(query,
isEditMode);
            case EventCategory.AGGREGATION:
                return this.analyseAggregationEvent(query,
isEditMode);
            case EventCategory.COMPLEX:
                return this.analyseComplexEvent(query,
isEditMode);
        }
    }

    manageParamValueForEditor(contextParam: ContextParam |
undefined, query: string): string {
        const queryRows = query.split('\n');
        if (query.includes(EventKeywords.CONTEXT)) {
            for (let i = 0; i < queryRows.length; i++) {
                const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
                if (rowLexems.includes(EventKeywords.CONTEXT)) {
                    queryRows.splice(i, 1);
                }
            }
            if (contextParam) {
                queryRows.push(`CONTEXT = "${contextParam.code}"`);
            }
            return queryRows.join('\n');
        }

        manageComponentValueForEditor(component: string | undefined,
query: string): string {
            const queryRows = query.split('\n');
            if (query.includes(EventKeywords.COMPONENT)) {
                for (let i = 0; i < queryRows.length; i++) {
                    const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
                    if (rowLexems.includes(EventKeywords.COMPONENT))
                    {
                        queryRows.splice(i, 1);
                    }
                }
            }
        }
    }
}

```

```

    }
  }
  if (component) {
    queryRows.push(`COMPONENT = "${component}"`)
  }
  return queryRows.join('\n');
}

manageEventsValueForEditor(selectedEvents: EventListItem[] |
undefined, query: string): string {
  const queryRows = query.split('\n');
  if (query.includes(EventKeywords.EVENTS)) {
    for (let i = 0; i < queryRows.length; i++) {
      const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
      if (rowLexems.includes(EventKeywords.EVENTS)) {
        queryRows.splice(i, 1);
      }
    }
  }
  if (selectedEvents?.length) {
    queryRows.push(`EVENTS = "${selectedEvents.map(ev =>
ev.code).join(', ')}"`)
  }
  return queryRows.join('\n');
}

private analyseAtomicEvent(query: string, isEditMode:
boolean): EventAnalyzerResponse {
  let requiredFields = [EventKeywords.DESCRPTION,
EventKeywords.CONTEXT, EventKeywords.COMPONENT];
  let event = {} as AtomicEvent;
  const queryRows = query.split('\n');
  const eventInitLexems = queryRows[0].trim().split('
').filter(lex => lex !== '');
  if (eventInitLexems[0] !== EventKeywords.EVENT) {
    return {
      errorMessage: `Ошибка: объявление события должно
начинаться с ключевого слова "${EventKeywords.EVENT}"`,
      record: null
    }
  }
  if (isEditMode) {
    event.code = eventInitLexems[1];
  }
  for (let i = 0; i < queryRows.length; i++) {
    const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
    for (let field of requiredFields) {
      if (rowLexems.includes(field)) {
        if (rowLexems[0] !== field) {
          return {
            errorMessage: `Ошибка в строке ${i +
1}: строка должна начинаться с ключевого слова "${field}"`,
            record: null
          }
        }
      }
    }
  }
}

```

```

        }
    }
    if (rowLexems[1] !== '=') {
        return {
            errorMessage: `Ошибка в строке ${i +
1}: пропущен или неверно указан знак присваивания "=" после
ключевого слова "${field}"`,
            record: null
        }
    }
    if (rowLexems.length > 3) {
        const valueLexem =
rowLexems.slice(2).join(' ');
        rowLexems.splice(2, rowLexems.length -
2, valueLexem);
        if (!valueLexem.endsWith('"') ||
!valueLexem.startsWith('"')
            || valueLexem.split('"').length !==
3) {
            return {
                errorMessage: `Ошибка в строке
${i + 1}: неверно указано значение поля "${field}"`,
                record: null
            }
        }
    }
    event = this.setAtomicEventFieldValue(event,
field, rowLexems[2]);
    requiredFields =
requiredFields.filter(requiredField => requiredField !== field);
}
}

if (requiredFields.length) {
    return {
        errorMessage: `Не определены следующие
обязательные параметры события: ${requiredFields.join(', ')}`,
        record: null
    }
} else {
    return {
        errorMessage: '',
        record: event
    }
}
}

private analyseAggregationEvent(query: string, isEditMode:
boolean): EventAnalyzerResponse {
    let requiredFields = [
        EventKeywords.DESRIPTION,
        EventKeywords.CONTEXT,
        EventKeywords.AGGREGATION_QUERY,

```

```

        EventKeywords.TIMESTAMP_END,
        EventKeywords.TIMESTAMP_BEGIN
    ];
    let event = {} as AggregationEvent;
    const queryRows = query.split('\n');
    const eventInitLexems = queryRows[0].trim().split('
').filter(lex => lex !== '');
    if (eventInitLexems[0] !== EventKeywords.EVENT) {
        return {
            errorMessage: `Ошибка: объявление события должно
начинаться с ключевого слова "${EventKeywords.EVENT}"`,
            record: null
        }
    }
    if (isEditMode) {
        event.code = eventInitLexems[1];
    }
    for (let i = 0; i < queryRows.length; i++) {
        const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
        for (let field of requiredFields) {
            if (rowLexems.includes(field)) {
                if (rowLexems[0] !== field) {
                    return {
                        errorMessage: `Ошибка в строке ${i +
1}: строка должна начинаться с ключевого слова "${field}"`,
                        record: null
                    }
                }
            }
            if (rowLexems[1] !== '=') {
                return {
                    errorMessage: `Ошибка в строке ${i +
1}: пропущен или неверно указан знак присваивания "=" после
ключевого слова "${field}"`,
                    record: null
                }
            }
            if (rowLexems.length > 3) {
                const valueLexem =
rowLexems.slice(2).join(' ');
                rowLexems.splice(2, rowLexems.length -
2, valueLexem);
                if (!valueLexem.endsWith('"') ||
!valueLexem.startsWith('"')
|| valueLexem.split('"').length !==
3) {
                    return {
                        errorMessage: `Ошибка в строке
${i + 1}: неверно указано значение поля "${field}"`,
                        record: null
                    }
                }
            }
        }
    }
}

```

```

        event =
this.setAggregationEventFieldValue(event, field, rowLexems[2]);
        requiredFields =
requiredFields.filter(requiredField => requiredField !== field);
    }
}

if (requiredFields.length) {
    return {
        errorMessage: `Не определены следующие
обязательные параметры события: ${requiredFields.join(', ')}\`,
        record: null
    }
} else {
    return {
        errorMessage: '',
        record: event
    }
}
}

private analyseComplexEvent(query: string, isEditMode:
boolean): EventAnalyzerResponse {
    let requiredFields = [EventKeywords.DESCRPTION,
EventKeywords.CONTEXT, EventKeywords.TEMPLATE_EVENT];
    let event = {} as ComplexEvent;
    const queryRows = query.split('\n');
    const eventInitLexems = queryRows[0].trim().split('
').filter(lex => lex !== '');
    if (eventInitLexems[0] !== EventKeywords.EVENT) {
        return {
            errorMessage: `Ошибка: объявление события должно
начинаться с ключевого слова "${EventKeywords.EVENT}"`,
            record: null
        }
    }
    if (isEditMode) {
        event.code = eventInitLexems[1];
    }
    for (let i = 0; i < queryRows.length; i++) {
        const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
        for (let field of requiredFields) {
            if (rowLexems.includes(field)) {
                if (rowLexems[0] !== field) {
                    return {
                        errorMessage: `Ошибка в строке ${i +
1}: строка должна начинаться с ключевого слова "${field}"`,
                        record: null
                    }
                }
            }
        }
        if (rowLexems[1] !== '=') {
            return {

```

```

        errorMessage: `Ошибка в строке ${i +
1}: пропущен или неверно указан знак присваивания "=" после
ключевого слова "${field}"`,
        record: null
    }
}
    if (rowLexems.length > 3) {
        const valueLexem =
rowLexems.slice(2).join(' ');
        rowLexems.splice(2, rowLexems.length -
2, valueLexem);
        if (!valueLexem.endsWith('"') ||
!valueLexem.startsWith('"')
            || valueLexem.split('"').length % 2
!== 1) {
            return {
                errorMessage: `Ошибка в строке
${i + 1}: неверно указано значение поля "${field}"`,
                record: null
            }
        }
    }
    event =
this.setComplexEventFieldValue(event, field, rowLexems[2]);
    requiredFields =
requiredFields.filter(requiredField => requiredField !== field);
}
}

if (requiredFields.length) {
    return {
        errorMessage: `Не определены следующие
обязательные параметры события: ${requiredFields.join(', ')}`,
        record: null
    }
} else {
    return {
        errorMessage: '',
        record: event
    }
}
}

private setAtomicEventFieldValue(atomicEvent: AtomicEvent,
field: EventKeywords, value: string) {
    switch (field) {
        case EventKeywords.DESRIPTION:
            atomicEvent.description = value.replace(/"/g,
""");
            return atomicEvent;
        case EventKeywords.COMPONENT:
            atomicEvent.codeComponent = value.replace(/"/g,
""");

```

```

        return atomicEvent;
    case EventKeywords.CONTEXT:
        atomicEvent.contextParamCode =
value.replace("/g, "");
        return atomicEvent;
    default:
        return atomicEvent;
    }
}

private setAggregationEventFieldValue(aggregationEvent:
AggregationEvent, field: EventKeywords, value: string) {
    switch (field) {
        case EventKeywords.DESRIPTION:
            aggregationEvent.description =
value.replace("/g, "");
            return aggregationEvent;
        case EventKeywords.COMPONENT:
            aggregationEvent.codeComponent =
value.replace("/g, "");
            return aggregationEvent;
        case EventKeywords.CONTEXT:
            aggregationEvent.contextParamCode =
value.replace("/g, "");
            return aggregationEvent;
        case EventKeywords.AGGREGATION_QUERY:
            aggregationEvent.aggregationQuery =
value.replace("/g, "");
            return aggregationEvent;
        case EventKeywords.TIMESTAMP_BEGIN:
            aggregationEvent.timestampBegin = +value;
            return aggregationEvent;
        case EventKeywords.TIMESTAMP_END:
            aggregationEvent.timestampEnd = +value;
            return aggregationEvent;
        default:
            return aggregationEvent;
    }
}

private setComplexEventFieldValue(complexEvent:
ComplexEvent, field: EventKeywords, value: string) {
    switch (field) {
        case EventKeywords.DESRIPTION:
            complexEvent.description = value.replace("/g,
""");
            return complexEvent;
        case EventKeywords.TEMPLATE_EVENT:
            complexEvent.templateEvent =
JSON.stringify(JSON.parse(value.replace(/^.|.$/g, "")));
            return complexEvent;
        case EventKeywords.CONTEXT:
            complexEvent.contextParamCode =
value.replace("/g, "");
            return complexEvent;
    }
}

```

```

        default:
            return complexEvent;
    }
}

export enum EventKeywords {
    AGGREGATION_QUERY = 'AGGREGATION_QUERY',
    TYPE = 'TYPE',
    CODE = 'CODE',
    DESCRIPTION = 'DESCRIPTION',
    EVENT = 'EVENT',
    EVENTS = 'EVENTS',
    COMPONENT = 'COMPONENT',
    TEMPLATE_EVENT = 'TEMPLATE_EVENT',
    TIMESTAMP_BEGIN = 'TIMESTAMP_BEGIN',
    TIMESTAMP_END = 'TIMESTAMP_END',
    CONTEXT = 'CONTEXT'
}

export enum ChangeEventMode {
    DROPDOWN = 'dropdown',
    EDITOR = 'editor'
}

export interface EventAnalyzerResponse {
    errorMessage: string;
    record: AtomicEvent | AggregationEvent | ComplexEvent |
    null;
}

```

Листинг В.2 – Файл active-rule-code-analyzer.ts

```

import {Injectable} from "@angular/core";
import {ActiveRule} from "../active-rule";
import {EventListItem} from '../../event/event';

@Injectable({providedIn: 'root'})
export class ActiveRuleCodeAnalyzer {

    analyseRule(ruleQuery: string, isEditMode: boolean):
    AnalyzerResponse {
        let requiredFields = [RuleKeywords.DESCRPTION,
        RuleKeywords.CONDITION, RuleKeywords.ACTION];
        let activeRule = new ActiveRule();
        const queryRows = ruleQuery.split('\n');
        const ruleInitLexems = queryRows[0].trim().split('
').filter(lex => lex !== '');
        if (ruleInitLexems[0] !== RuleKeywords.RULE) {
            return {
                errorMessage: `Ошибка: объявление правила должно
                начинаться с ключевого слова "${RuleKeywords.RULE}"`,
                record: null
            }
        }
    }
}

```

```

    }
    if (isEditMode) {
        activeRule.code = ruleInitLexems[1];
    }
    for (let i = 0; i < queryRows.length; i++) {
        const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
        for (let field of requiredFields) {
            if (rowLexems.includes(field)) {
                if (rowLexems[0] !== field) {
                    return {
                        errorMessage: `Ошибка в строке
${i+1}: строка должна начинаться с ключевого слова "${field}"`,
                        record: null
                    }
                }
                if (rowLexems[1] !== '=') {
                    return {
                        errorMessage: `Ошибка в строке
${i+1}: пропущен или неверно указан знак присваивания "=" после
ключевого слова "${field}"`,
                        record: null
                    }
                }
                if (rowLexems.length > 3) {
                    const valueLexem =
rowLexems.slice(2).join(' ');
                    rowLexems.splice(2, rowLexems.length -
2, valueLexem);
                    if (!valueLexem.endsWith('"') ||
!valueLexem.startsWith('"')
|| valueLexem.split('"').length !==
3) {
                        return {
                            errorMessage: `Ошибка в строке
${i+1}: неверно указано значение поля "${field}"`,
                            record: null
                        }
                    }
                }
            }
            activeRule =
this.setActiveRuleFieldValue(activeRule, field, rowLexems[2]);
            requiredFields =
requiredFields.filter(requiredField => requiredField !== field);
        }
    }

    if (requiredFields.length) {
        return {
            errorMessage: `Не определены следующие
обязательные параметры правила: ${requiredFields.join(', ')}`,
            record: null
        }
    }

```

```

    } else {
      return {
        errorMessage: '',
        record: activeRule
      }
    }
  }

  manageEventValueForEditor(selectedEvents: EventListItem[] |
undefined, query: string): string {
    const queryRows = query.split('\n');
    if (query.includes(RuleKeywords.EVENT)) {
      for (let i = 0; i < queryRows.length; i++) {
        const rowLexems = queryRows[i].trim().split('
').filter(lex => lex !== '');
        if (rowLexems.includes(RuleKeywords.EVENT)) {
          queryRows.splice(i, 1);
        }
      }
    }
    if (selectedEvents?.length) {
      queryRows.push(`EVENT = "${selectedEvents.map(ev =>
ev.code).join(', ')}"`)
    }
    return queryRows.join('\n');
  }

  private setActiveRuleFieldValue(activeRule: ActiveRule,
field: RuleKeywords, value: string) {
    switch (field) {
      case RuleKeywords.ACTION:
        activeRule.action = value.replace(/"/g, "");
        return activeRule;
      case RuleKeywords.CONDITION:
        activeRule.condition = value.replace(/"/g, "");
        return activeRule;
      case RuleKeywords.DESCRPTION:
        activeRule.description = value.replace(/"/g, "");
        return activeRule;
      default:
        return activeRule;
    }
  }
}

export enum RuleKeywords {
  RULE = 'RULE',
  TYPE = 'TYPE',
  CODE = 'CODE',
  DESCRIPTION = 'DESCRIPTION',
  EVENT = 'EVENT',
  CONDITION = 'CONDITION',
  ACTION = 'ACTION'
}

```

```
export enum ChangeEventMode {  
    DROPDOWN = 'dropdown',  
    EDITOR = 'editor'  
}  
  
export interface AnalyzerResponse {  
    errorMessage: string;  
    record: ActiveRule | null;  
}
```

Приложение Г
ПРИМЕР СИСТЕМЫ ДЛЯ ТЕСТИРОВАНИЯ ПРИЛОЖЕНИЯ
(рекомендуемое)

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

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

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

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

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

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

Функциональные требования: Осуществляется *мониторинг и контроль пропускного режима*:

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

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

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

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

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

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

Осуществляется *мониторинг и контроль жизнедеятельности предприятия:*

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

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

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

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

Осуществляется *мониторинг и контроль электропотребления предприятия:*

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

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

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

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

В системе определяются набор типов событий, представленных в таблице 1. Все типы событий делятся на четыре категории:

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

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

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

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

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

Для реализации управляющих воздействий в правилах используются команды управления для устройств (проходных, входных дверей, обогревателей, кондиционеров).

Приложение Д
РЕЗУЛЬТАТЫ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

The screenshot shows a web interface titled "Языковое представление событий" (Event Language Representation). It has two tabs: "Свободный ввод" (Free input) and "Конструктор" (Constructor). The "Конструктор" tab is active. Under "Тип события" (Event type), the dropdown is set to "atomic". Under "Параметр контекста" (Context parameter), the dropdown is set to "00003". Under "Компонент" (Component), the dropdown is set to "00001". Below these fields, a code editor displays the following JSON-like structure:

```
1 EVENT 00021
2 DESCRIPTION = "Изменение температуры"
3 COMPONENT = "00001"
4 CONTEXT = "00003"
```

At the bottom, a green status bar shows a checkmark and the text "✓ Событие успешно создано" (Event successfully created). Below this are two blue buttons: "Проверить" (Check) and "Сохранить" (Save).

Рисунок Д.1 – Создание атомарного события изменения температуры

The screenshot shows the same web interface as Figure D.1, but with the "Свободный ввод" (Free input) tab active. Under "Тип события" (Event type), the dropdown is set to "aggregation". Under "Параметр контекста" (Context parameter), the dropdown is set to "00003". Below these fields, a code editor displays the following JSON-like structure:

```
1 EVENT 00022
2 DESCRIPTION = "Изменение температуры за период времени"
3 AGGREGATION_QUERY = "{}"
4 TIMESTAMP_BEGIN = 1652950484
5 TIMESTAMP_END = 1653950484
6 CONTEXT = "00003"
```

At the bottom, a green status bar shows a checkmark and the text "✓ Событие успешно создано" (Event successfully created). Below this are two blue buttons: "Проверить" (Check) and "Сохранить" (Save).

Рисунок Д.2 – Создание агрегирующего события изменения температуры за период времени

← Языковое представление событий →

Свободный ввод Конструктор

Тип события
complex

Параметр контекста
00007

События
00021, 00007

```

1 EVENT 00023
2 DESCRIPTION = "Пожарная опасность"
3 CONTEXT = "00007"
4 TEMPLATE_EVENT = "{\"test\":\"Test\"}"
5 EVENTS = "00021,00007"

```

✓ Событие успешно сохранено

Проверить Сохранить

Рисунок Д.3 – Создание сложного события пожарной опасности

← Языковое представление событий →

Свободный ввод Конструктор

Тип события
atomic

Параметр контекста
00001

Компонент
00001

```

1 RULE
2 DESCRIPTION = "Изменение силы тока"
3 CONTEXT = "00001"
4 COMPONENT = "00001"

```

❌ Ошибка: объявление события должно начинаться с ключевого слова "EVENT"

Проверить Сохранить

Рисунок Д.4 – Попытка создать событие с ошибкой в языковой конструкции

←
Языковое представление событий
→]

Свободный ввод

Конструктор

Тип события

complex
▼

Параметр контекста

00007
▼

00004 - Температура

00005 - Дождь

00006 - Движение

00007 - Дым

События

00021, 00007
▼

✓ Событие успешно сохранено

Проверить

Сохранить

Рисунок Д.5 – Изменение параметра контекста события пожарной опасности

←
Языковое представление событий
→]

Свободный ввод

Конструктор

Тип события

complex
▼

Параметр контекста

00007
▼

События

00021, 00007
▼

```

1 EVENT 00023
2 DESCRIPTION = "Пожарная опасность"
3 CONTEXT = "00067"
4 EVENTS = "00021,00007"

```

⊗ Параметр контекста с кодом 00067 не существует

Проверить

Сохранить

Рисунок Д.6 – Попытка изменить параметр контекста на несуществующий события пожарной опасности

←

Языковое представление активных правил

→

Свободный ввод

Конструктор

00021

```
1 RULE 00008
2 DESCRIPTION = "Приведение в норму температуры"
3 CONDITION = "{"cond": "t<23"}"
4 ACTION = "{"1": "Включить обогрев", "2": "Выключить кондиционер"}"
5 EVENT = "00021"
```

✓ Ошибки не найдены

Проверить

Сохранить

Рисунок Д.7 – Результат проверки корректной языковой конструкции активного правила

←

Языковое представление событий

→

Свободный ввод

Конструктор

Тип события

complex

Параметр контекста

00007

События

00021, 00007

```
1 EVENT 00023
2 DESCRIPTION = "Пожарная опасность"
3 TEMPLATE_EVENT = "{"test": "Test"}"
4 EVENTS = "00021, 00007"
5 CONTEXT = "00007"
```

✓ Ошибки не найдены

Проверить

Сохранить

Рисунок Д.8 – Результат проверки корректной языковой конструкции события

←

Языковое представление активных правил

→]

Свободный ввод

Конструктор

00021

▼

```
1 RULE 00008
2 DESCRIPTION = "Приведение в норму температуры"
3 CONDITION "{cond": "t<23"}"
4 ACTION = "{1": "Включить обогрев", 2: "Выключить кондиционер"}"
5 EVENT = "00021"
```

⊗ Ошибка в строке 3: пропущен или неверно указан знак присваивания "=" после ключевого слова "CONDITION"

Проверить

Сохранить

Рисунок Д.9 – Результат проверки некорректной языковой конструкции активного правила

←

Языковое представление событий

→]

Свободный ввод

Конструктор

Тип события

atomic

▼

Параметр контекста

00003

▼

Компонент

00001

▼

```
1 EVENT 00021
2 "Изменение температуры"
3 COMPONENT = "00001"
4 CONTEXT = "00003"
```

⊗ Не определены следующие обязательные параметры события: DESCRIPTION

Проверить

Сохранить

Рисунок Д.10 – Результат проверки некорректной языковой конструкции события

←

Языковое представление активных правил

→]

Свободный ввод

Конструктор

00021

▼

1

RULE

2

DESCRIPTION = "Приведение в норму температуры"

3

CONDITION = "{cond: "t<23"}"

4

EVENT = "00021"

⊗

Не определены следующие обязательные параметры правила: ACTION

Проверить

Сохранить

Рисунок Д.11 – Попытка создать правило без указания действия

←

Языковое представление активных правил

→]

Свободный ввод

Конструктор

00021

▼

1

RULE 00008

2

DESCRIPTION = "Приведение в норму температуры"

3

CONDITION = "{cond: "t<23"}"

4

ACTION = "{1: "Включить обогрев", 2: "Выключить кондиционер"}"

5

EVENT = "00021"

Проверить

Сохранить

Рисунок Д.12 – Создание активного правила приведения в норму температуры

←

Языковое представление активных правил

→]

Свободный ввод

Конструктор

00021

1 RULE 00008
2 DESCRIPTION = "Приведение в норму температуры"
3 CONDITION = "{cond": "t<25"}"
4 ACTION = "{1": "Включить обогрев", "2": "Выключить кондиционер"}"
5 EVENT = "00021"

✓ Правило успешно сохранено

Проверить

Сохранить

Рисунок Д.13 – Изменение условия правила о приведении в норму температуры помещения

←

Языковое представление активных правил

→]

Свободный ввод

Конструктор

00021

1 RULE 00008
2 DESCRIPTION = "Приведение в норму температуры"
3 CONDITION = "{cond": "t<25"}"
4 ACTION = "{1": "Включить обогрев", "2": "Выключить кондиционер"}"
5 EVENT = "00045"

⊗ События с кодом 00045 не существует

Проверить

Сохранить

Рисунок Д.14 – Попытка изменить код события на несуществующий

Приложение Е
ЛИСТИНГ МОДУЛЬНЫХ ТЕСТОВ
(рекомендуемое)

Листинг Е.1 – Файл active-rule.component.spec.ts

```
import { TestBed } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { ActiveRuleComponent } from "../active-rule.component";
import { CUSTOM_ELEMENTS_SCHEMA } from "@angular/core";

fdescribe('ActiveRuleComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        ActiveRuleComponent
      ],
      schemas: [
        CUSTOM_ELEMENTS_SCHEMA
      ]
    }).compileComponents();
  });

  it('должен создаваться', () => {
    const fixture =
TestBed.createComponent(ActiveRuleComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`должно отобразиться диалоговое окно`, () => {
    const fixture =
TestBed.createComponent(ActiveRuleComponent);
    const app = fixture.componentInstance;
    expect(app.isDialogVisible).toBeTruthy();
  });
});
```

Листинг Е.2 – Файл active-rule-list.component.spec.ts

```
import {TestBed} from '@angular/core/testing';
import {RouterTestingModule} from '@angular/router/testing';
import {CUSTOM_ELEMENTS_SCHEMA} from "@angular/core";
import {ActiveRuleListComponent} from "../active-rule-
list.component";
import {ActiveRuleService} from "../active-rule.service";
import {HttpClient, HttpClientModule} from
"@angular/common/http";

fdescribe('ActiveRuleListComponent', () => {
  beforeEach(async () => {
```

```

    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule,
        HttpClientModule
      ],
      declarations: [
        ActiveRuleListComponent,
      ],
      schemas: [
        CUSTOM_ELEMENTS_SCHEMA
      ],
      providers: [
        ActiveRuleService,
        HttpClient
      ]
    }).compileComponents();
  });

  it('должен создаваться', () => {
    const fixture =
TestBed.createComponent(ActiveRuleListComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`должны загрузиться активные правила`, async () => {
    const fixture =
TestBed.createComponent(ActiveRuleListComponent);
    const app = fixture.componentInstance;
    await app.getRules();
    fixture.detectChanges();
    expect(app.activeRules.length).toBeGreaterThan(0);
  });
});

```

Листинг Е.3 – Файл active-rule-editor.component.spec.ts

```

import {TestBed} from '@angular/core/testing';
import {RouterTestingModule} from '@angular/router/testing';
import {CUSTOM_ELEMENTS_SCHEMA} from "@angular/core";
import {ActiveRuleService} from "../active-rule.service";
import {HttpClient, HttpClientModule} from
"@angular/common/http";
import {ActiveRuleEditorComponent} from "../active-rule-
editor.component";
import {EventService} from "../../event/event.service";
import {ActiveRule} from "../active-rule";
import {EventCategory} from "../../event/event";

fdescribe('ActiveRuleEditorComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({

```

```

        imports: [
            RouterTestingModule,
            HttpClientModule
        ],
        declarations: [
            ActiveRuleEditorComponent,
        ],
        schemas: [
            CUSTOM_ELEMENTS_SCHEMA
        ],
        providers: [
            ActiveRuleService,
            EventService,
            HttpClient
        ]
    })).compileComponents();
});

it('должен создаваться', () => {
    const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
});

it(`должно загрузиться активное правило`, () => {
    const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
    const app = fixture.componentInstance;
    await app.loadEditorData('00001');
    fixture.detectChanges();
    expect(app.activeRule).toBeTruthy();
});
});

```

Листинг Е.4 – Файл active-rule-free-write-editor.component.spec.ts

```

import {TestBed} from '@angular/core/testing';
import {RouterTestingModule} from '@angular/router/testing';
import {CUSTOM_ELEMENTS_SCHEMA} from "@angular/core";
import {ActiveRuleService} from "../active-rule.service";
import {HttpClient, HttpClientModule} from
"@angular/common/http";
import {ActiveRuleEditorComponent} from "../active-rule-
editor.component";
import {EventService} from "../../event/event.service";
import {ActiveRule} from "../active-rule";
import {EventCategory} from "../../event/event";

fdescribe('ActiveRuleFreeWriteEditorComponent', () => {
    beforeEach(async () => {
        await TestBed.configureTestingModule({

```

```

        imports: [
            RouterTestingModule,
            HttpClientModule
        ],
        declarations: [
            ActiveRuleEditorComponent,
        ],
        schemas: [
            CUSTOM_ELEMENTS_SCHEMA
        ],
        providers: [
            ActiveRuleService,
            EventService,
            HttpClient
        ]
    }).compileComponents();
});

it('должен переключиться на режим свободного ввода', () => {
    const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
});

it(`должно загрузиться активное правило`, () => {
    const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
    const app = fixture.componentInstance;
    expect(app.activeRule).toBeTruthy();
});

it(`должен загрузиться языковой шаблон активного правила`,
() => {
    const rule: ActiveRule = {
        code: '00009',
        action: '{}',
        condition: '{}',
        description: 'Test rule',
        event: [
            {
                categoryEvent: EventCategory.ATOMIC,
                description: 'Test event',
                code: '00002'
            }
        ]
    };
    const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
    const app = fixture.componentInstance;
    app.initEditorQuery(rule);
    expect(app.query).toBeTruthy();
});

```

```

    });

    it('должно выводиться сообщение о корректной языковой
    конструкции правила при проверке', () => {
        const rule: ActiveRule = {
            code: '00009',
            action: '{}',
            condition: '{}',
            description: 'Rule',
            event: [
                {
                    categoryEvent: EventCategory.ATOMIC,
                    description: 'Event',
                    code: '00002'
                }
            ]
        };
        const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
        const app = fixture.componentInstance;
        app.initEditorQuery(rule);
        app.checkCodeSyntax();
        expect(app.successMessage).toBeTruthy();

        expect(fixture.nativeElement.querySelector('#successMsg')).toBeD
efined();
    });

    it('должно выводиться сообщение о некорректной языковой
    конструкции правила при ошибке по время проверки', () => {
        const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
        const app = fixture.componentInstance;
        app.query = '';
        app.checkCodeSyntax();

        expect(app.codeAnalyzerResponse?.errorMessage).toBeTruthy();

        expect(fixture.nativeElement.querySelector('#errorMsg')).toBeDef
ined();
    });
});

```

Листинг Е.5 – Файл active-rule-constructor-editor.component.spec.ts

```

import {TestBed} from '@angular/core/testing';
import {RouterTestingModule} from '@angular/router/testing';
import {CUSTOM_ELEMENTS_SCHEMA} from "@angular/core";
import {ActiveRuleService} from "../active-rule.service";
import {HttpClient, HttpClientModule} from
"@angular/common/http";

```

```

import {ActiveRuleEditorComponent} from "../active-rule-
editor.component";
import {EventService} from "../../event/event.service";
import {ActiveRule} from "../active-rule";
import {EventCategory} from "../../event/event";

fdescribe('ActiveRuleConstructorEditorComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule,
        HttpClientModule
      ],
      declarations: [
        ActiveRuleEditorComponent,
      ],
      schemas: [
        CUSTOM_ELEMENTS_SCHEMA
      ],
      providers: [
        ActiveRuleService,
        EventService,
        HttpClient
      ]
    }).compileComponents();
  });

  it('должен переключиться на режим конструктора', () => {
    const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it('должно загрузиться активное правило', () => {
    const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
    const app = fixture.componentInstance;
    expect(app.activeRule).toBeTruthy();
  });

  it('должны загрузиться поля ввода данных', () => {
    const fixture =
TestBed.createComponent(ActiveRuleEditorComponent);
    const description =
fixture.debugElement.nativeElement.querySelector('#description')
;
    const action =
fixture.debugElement.nativeElement.querySelector('#action');
    const condition =
fixture.debugElement.nativeElement.querySelector('#condition');
  });

```

```
        const events =
fixture.debugElement.nativeElement.querySelector('#events');
        expect(description).toBeTruthy();
        expect(action).toBeTruthy();
        expect(events).toBeTruthy();
        expect(condition).toBeTruthy();
    });
});
```