МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Инструментальные средства управления метаданными для сервиса потоковой обработки событий

Студент _____ Тамбовцев Кирилл Андреевич
(подпись, дата) (ФИО полностью)

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

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

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

Заведующий кафедрой _____ Макарычев П.П.
(подпись) (фамилия, инициалы)

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

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

Пенза, 2022

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

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____П.П. Макарычев
«__» _____2022 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Тамбовцев Кирилл Андреевич гр. 18ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Инструментальные средства управления метаданными для сервиса потоковой обработки событий

Тема утверждена приказом ПГУ № 327/О от "18" апреля 2022 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: инструментальные средства для управления метаданными сервиса потоковой обработки событий.

2) Среда выполнения: операционная система Windows 10, web-браузер Google Chrome.

Языковые средства разработки: TypeScript, JavaScript, UML.

Инструментальные средства: Visual Studio Code, Postman, npm, Draw.io, UMLet.

Технологии разработки: ООП, сервисно-ориентированный подход.

3) Основные функции приложения:

а) регистрация, авторизация и восстановление аккаунта;

б) администрирование пользователей;

в) добавление, изменение и удаление событий;

г) добавление, изменение и удаление устройств;

д) добавление, изменение и удаление систем;

е) выстраивание иерархии систем;

ж) настройка «личного кабинета»;

з) визуализация статистики работы сервиса.

5. Перечень подлежащих разработке вопросов:

1) *Анализ предметной области и постановка задачи на разработку;*
2) *Разработка требований;*
3) *Реализация инструментальных средств;*
4) *Контроль качества инструментальных средств;*
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 постановка задачи на разработку	15%	13.04.2022 – 22.04.2022	
2	Разработка требований	15%	23.04.2022 – 02.05.2022	
3	Реализация инструментальных средств	40%	03.05.2022 – 28.05.2022	
4	Контроль качества инструментальных средств	20%	29.05.2022 – 10.06.2022	
5	Оформление пояснительной записки	10%	11.06.2022 – 17.06.2022	

Дата выдачи задания « 22 »___декабря___2021 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 167 листов, 52 рисунка, 23 таблицы, 17 использованных источников и 8 приложений.

ПОТОКОВАЯ ОБРАБОТКА СОБЫТИЙ, ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА, УПРАВЛЕНИЕ МЕТАДАННЫМИ, 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 инструментальные средства управления метаданными 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 Microsoft Visual Studio Code.

Среда исполнения – Windows 10, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы инструментальные средства управления метаданными. Все требования к разработке приложения соблюдены.

Содержание

Введение 7
1 Анализ предметной области и постановка задачи на разработку 9
1.1 Сервис потоковой обработки событий на основе активных правил 9
1.1.1 Назначение и функциональные возможности сервиса потоковой обработки событий 9
1.1.2 Микросервисная событийно-ориентированная архитектура сервиса потоковой обработки событий 13
1.1.3 Сценарий работы сервиса потоковой обработки событий 15
1.2 Управление метаданными в процессе потоковой обработки событий и исполнения активных правил 18
1.2.1 Состав метаданных сервиса потоковой обработки событий 18
1.2.2 Жизненный цикл метаданных сервиса потоковой обработки событий 19
1.2.3 Управление метаданными сервиса потоковой обработки событий 21
1.3 Постановка задачи на разработку инструментальных средств управления метаданными 23
1.3.1 Цели разработки и назначение программных средств управления метаданными 23
1.3.2 Анализ программного обеспечения аналогичного назначения 24
1.3.2.1 Платформа «AWS IoT» 24
1.3.2.2 Платформа «Azure IoT» 25
1.3.2.3 Платформа «Cisco IoT» 28
1.3.2.4 Сравнительный анализ программ-аналогов 29
1.3.3 Перечень автоматизируемых функций инструментальных программных средств 34
1.4 Планирование процесса разработки инструментальных средств управления метаданными 35

1.4.1 Выбор технологии и описание жизненного цикла разработки комплекса программных средств сервиса потоковой обработки событий 35
1.4.2 Планирование процесса разработки инструментальных средств управления метаданными 37
1.4.3 Анализ ресурсных затрат на разработку 42
2 Разработка требований к инструментальным средствам управления метаданными для сервиса потоковой обработки событий 47
2.1 Функциональные требования к инструментальным средствам управления метаданными 47
2.2 Требования к пользовательскому интерфейсу инструментальных средств управления метаданными 51
2.3 Требования к архитектуре программных средств управления метаданными 53
2.4 Обоснование выбора технологий и средств разработки инструментальных программных средств управления метаданными 56
2.5 Требования к аппаратно-программному обеспечению 58
3 Разработка инструментальных средств управления метаданными для сервиса потоковой обработки событий 60
3.1 Разработка архитектуры развертывания инструментальных программных средств 60
3.2 Разработка программных средств взаимодействия с сервером управления метаданными 65
3.2.1 Разработка программных средств уровня модели 65
3.2.2 Разработка программных средств взаимодействия модели с базой метаданных 69
3.3 Разработка пользовательского интерфейса 72
3.3.1 Разработка сценариев взаимодействия пользователя с инструментальными средствами 72
3.3.2 Разрабатка структуры пользовательского интерфейса и прототипов пользовательских страниц и окон 74

3.3.3 Разработка программных средств уровня представления 82
4 Контроль качества инструментальных программных средств управления метаданными для сервиса потоковой обработки событий 85
4.1 Планирование процесса контроля качества серверных программных средств 85
4.1.1 Мероприятия по контролю качества программных средств 85
4.1.2 Обоснование выбора моделей тестирования 86
4.1.3 Обоснование выбора метрик программного кода 87
4.1.4 Обоснование выбора методов оценки и метрик качества пользовательского интерфейса 88
4.2 Тестирование серверных программных средств 89
4.2.1 Модульное тестирование 89
4.2.2 Интеграционное тестирование 90
4.2.3 Конфигурационное тестирование 92
4.3 Оценка качества программного кода 93
4.3.1 Метрики Холстеда 93
4.3.2 Метрики устойчивости и абстрактности 99
4.4 Оценка качества пользовательского интерфейса 103
Заключение 106
Список использованных источников 108
Приложение А. Глоссарий 110
Приложение Б. Графики трудовых ресурсов 112
Приложение В. Листинг моделей инструментальных средств 115
Приложение Г. Листинг сервисов инструментальных средств 131
Приложение Д. Диаграммы функциональных блоков пользовательского интерфейса 142
Приложение Е. Листинг модульных тестов 146
Приложение Ж. Листинг интеграционных тестов 153
Приложение И. Результаты конфигурационного тестирования 159

Введение

На сегодняшний день разработано большое количество систем, которые для различных задач работают в масштабе реального времени. Они генерируют огромные объемы данных, требующие обработки в кратчайшие сроки, чтобы полученная информация оставалась актуальной.

В большинстве случаев компании используют системы с более монолитной структурой, которая состоит в основном из приложения (уровня обработки данных) и базы данных (уровня хранения данных). Приложения непрерывно обрабатывают входящие потоки данных, приходящие от внешних служб или пользовательских устройств, а также постоянно обращаются к базам данных через их системы управления (СУБД) для обновления состояния полученной информации. Данные системы обладают высокой скоростью разработки, а также низкой себестоимостью. Однако при развитии и подобных приложений раскрываются их недостатки, заключающиеся в сложности масштабирования, трудности развертывания новых или обновления старых частей программы, усложнения тестирования и выявления ошибок и т.д.

Одним из вариантов решения указанных проблем монолитных и тесно связанных приложений является построение системы на основе микросервисов. Микросервисы – это небольшие, автономные и независимые приложения, отвечающие за конкретный элемент логики в предметной области. Системы обработки данных создаются из нескольких микросервисов, которые взаимодействуют друг с другом через стандартизованные интерфейсы (например, HTTP-соединения REST). При этом улучшается отказоустойчивость системы, упрощается поддержка отдельных ее компонент, а также повышаются в разы возможности в масштабировании комплекса связанных программных средств.

Практически все данные, приходящие в данные системы, создаются в виде непрерывных потоков событий. Именно для хранения и обработки этих потоков событий, для вычленения из потоков отдельных событий, нахождения

между ними связей и закономерностей, а также для реагирования на появление определенных событийных шаблонов разрабатываются приложения на событийно-ориентированной архитектуре. Приложения, созданные на основе событийно-ориентированной парадигмы, являются развитием микросервисов. Они обмениваются информацией через очереди событий и хранят данные приложения в виде локального состояния вместо обращения к внешним хранилищам данных.

Данные приложения, как и любые другие сервисы, нуждаются в управлении администратором извне посредством специальных средств, ориентированных на человека. Именно поэтому имеется необходимость в разработке инструментального средства, позволяющего управлять метаданными событий и исполняемых правил.

Целью выпускной квалификационной работы является реализация инструментальных средств управления метаданными для сервиса потоковой обработки событий в виде клиентского приложения. Разработка включает в себя анализ требований, проектирование, реализацию и тестирование программного обеспечения.

1 Анализ предметной области и постановка задачи на разработку

1.1 Сервис потоковой обработки событий на основе активных правил

1.1.1 Назначение и функциональные возможности сервиса потоковой обработки событий

Сервис потоковой обработки событий и исполнения активных правил предназначен для реализации событийно-ориентированной обработки потоков событий, поступающих от наблюдаемых систем, а также выявляемых в соответствии с заданными событийными шаблонами. Кроме того, рассматриваемый сервис предполагает реализацию механизмов реагирования на события в виде исполнения активных правил, воздействующих на управляемые системы.

Данная технология может быть использована в приложениях управления и контроля техническими объектами, в медицинских приложениях для предупреждения врачей о состоянии пациента, в системах умного дома и контроля производства, в транспортных приложениях для прогноза пробок, а также в контроле воздушных сообщений для обнаружения потенциально опасных маршрутов и предупреждении диспетчера о сближении самолетов и многих других сферах человеческой деятельности.

Во всех указанных случаях сервис потоковой обработки событий и исполнения активных правил можно описать как систему, на вход которой поступает событие и его контекст, а на выходе мы получаем определенную реакцию сервиса.

Кроме того, нельзя обойтись без специальных средств администрирования, аудита и журнализации работы сервиса.

На рисунке 1 представлена диаграмма вариантов использования сервиса потоковой обработки событий и исполнения активных правил.

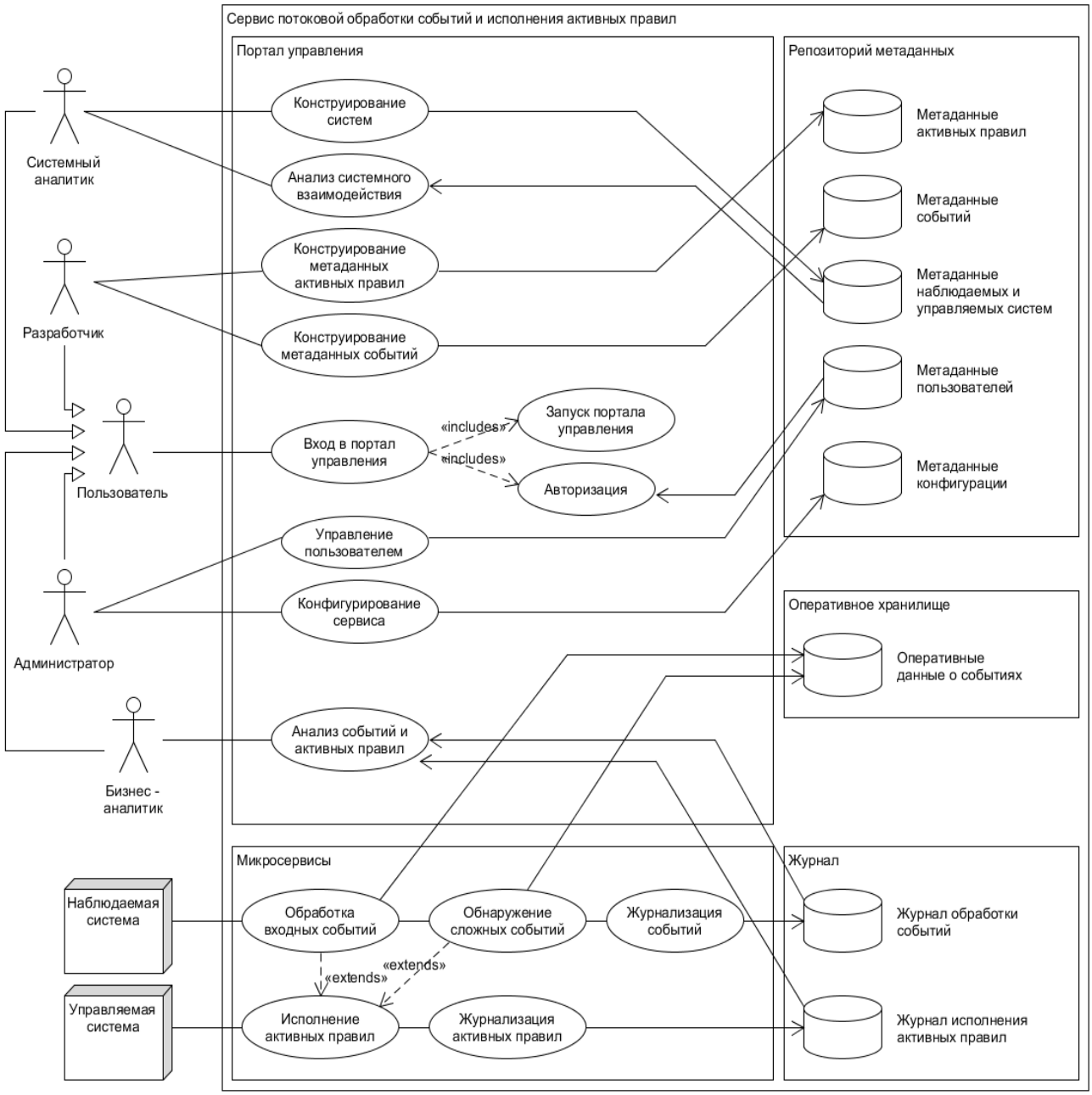


Рисунок 1 – Диаграмма вариантов использования сервиса потоковой обработки событий и исполнения активных правил

Основными ролями пользователей в системе являются:

– разработчик активных правил и метаданных событий;
– системный аналитик;
– бизнес-аналитик;
– администратор.

Основными группами функций сервиса потоковой обработки событий и исполнения активных правил являются:

– конструирование и управление активными правилами – создание и редактирование правил и их компонент, проверка корректности правила, подключение и отключение правила;

– конструирование и управление метаданными событий – создание и редактирование объектов событий, привязка созданных типов к компонентам наблюдаемой системы;

– анализ активных правил – выявление и устранение коллизий исполнения активных правил (статический анализ на этапе конструирования и динамический анализ на этапе исполнения);

– обработка событий – обработка потока событий от наблюдаемых систем, первичная обработка входных событий и формирование событий на основе агрегации входных данных, выявление сложных событий на основе заданных шаблонов, доставка событий к соответствующим компонентам сервиса для их обработки;

– исполнение активных правил – выбор экземпляра активного правила (группы правил) для исполнения, вычисление условия экземпляра активного правила, принятие решение о выполнении действия экземпляра активного правила, применение действия к исполняемому объекту;

– журнализация и анализ данных журналов событий - журнализация обрабатываемых событий, исполнения активных правил, обработки компонент активных правил, выявление закономерностей в журналах событий, моделирование новых событий и активных правил на основе выявленных закономерностей, выявление «проблемных» событий;

– настройка и масштабирование сервиса – изменение параметров функционирования отдельных компонент с учетом результатов аудита, масштабирование отдельных компонент (изменение параметров и распределение очередей и потоков обработки) и сервиса в целом

(подключение/отключение компонентов), синхронизация глобальных и локальных метаданных, описывающих конфигурацию системы и отдельных компонент.

Перечень функций в зависимости от роли субъекта в сервисе, представлен в таблице 1.

Таблица 1 – Функции сервиса

№	Роль	Группа функций
1	Разработчик	Конструирование метаданных активных правил Конструирование метаданных событий
2	Администратор сервиса	Настройка и масштабирование сервиса Работа с метаданными компонент сервиса Администрирование пользователей
3	Системный аналитик	Конструирование метаданных систем Анализ взаимодействия систем и их компонентов
4	Аналитик	Аудит обрабатываемых событий Аудит исполнения активных правил
5	Система	Анализ коллизий взаимодействия активных правил Обработка событий Исполнение активных правил Журнализация и анализ данных журналов событий

Большинство функций, используемых пользователями, реализуются при помощи инструментальных средств управления метаданными. Все метаданные, а также их изменения сохраняются в специальных репозиториях.

1.1.2 Микросервисная событийно-ориентированная архитектура сервиса потоковой обработки событий

Архитектура сервиса потоковой обработки событий и исполнения активных правил построена на принципах интеграции микросервисного и событийно-ориентированных подходов.

При построении самой системы потоковой обработки событий была использована микросервисная архитектура, которая позволяет разделить приложение на отдельные сервисы, тем самым поддерживая его модульность. Модули независимы друг от друга, из-за чего их легко масштабировать, заменять, разворачивать и обновлять [1]. Модули могут быть реализованы с использованием различных языков программирования, фреймворков, связующего программного обеспечения, выполняться в различных средах контейнеризации, виртуализации, под управлением различных операционных систем на различных аппаратных платформах [2].

Для доставки событий, поступающих от наблюдаемых систем, используется событийно-ориентированный подход на основе механизма очередей и платформы потоковой обработки данных Kafka с применением фреймворка Kafka Stream [3]. Для обращения к компонентам сервиса в процессе их настройки, запуск и останова, синхронизации метаданных используется REST-сервер.

Архитектура сервиса потоковой обработки событий и исполнения активных правил представлена на рисунке 2. Входные события поступают в сервис от наблюдаемых систем (баз данных и систем управления базами данных, систем управления техническими объектами – умный дом, интернет вещей и пр.), обрабатываются сервисом и, при наличии активных правил, выполняются соответствующие действия по отношению к управляемым системам.

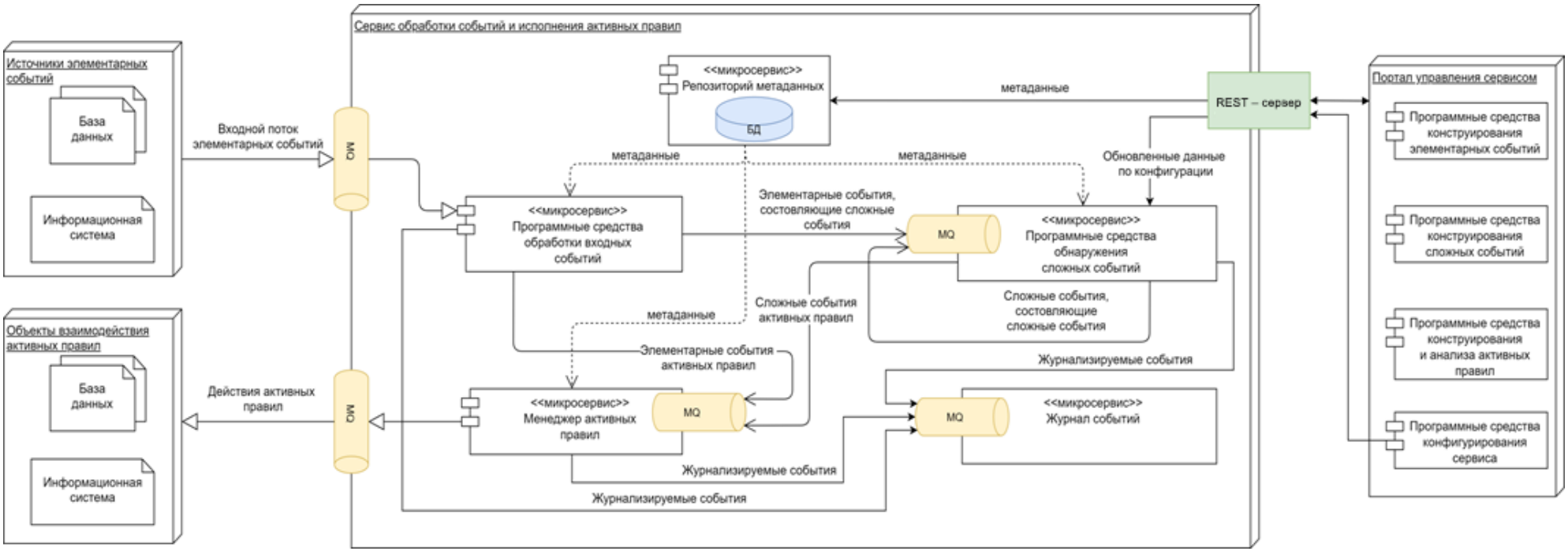


Рисунок 2 – Архитектура сервиса потоковой обработки событий и исполнения активных правил

Для хранения метаданных в процессе функционирования сервиса потоковой обработки событий и исполнения активных правил спроектирован централизованный репозиторий метаданных.

Для повышения эффективности обработки событий, исполнения активных правил и функционирования сервиса в целом, каждый из компонент сервиса имеет локальный репозиторий метаданных в виде in-memory хранилища под управлением Redis. Синхронизация централизованного и локального репозиториев метаданных осуществляется через REST-сервер по специальному сценарию, призванному снизить время ожидания при запросе к метаданным без ущерба актуальности метаданных.

1.1.3 Сценарий работы сервиса потоковой обработки событий

Работа сервиса потоковой обработки событий и исполнения активных правил представляет собой длительную или же вовсе непрерывную обработку потока событий в масштабе реального времени. Это означает, что определение событий и реакция на них должна занимать определенное небольшое количество времени для поддержания постоянного и своевременного взаимодействия с созданным окружением. Поддержание в системе рассмотренных характеристик реализуется механизмами активных правил, а также промежуточными сервисами пересылки данных.

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

Микросервисы, размещенные в системе, используют созданные пользователями метаданные для работы с потоками событий, приходящими от наблюдаемых систем: обрабатывают входящие события, определяют появление выстроенные событийные шаблоны, а также идентифицируют и активируют активные правила. Результаты работы микросервисов журнализируются, после чего предоставляются аналитикам.

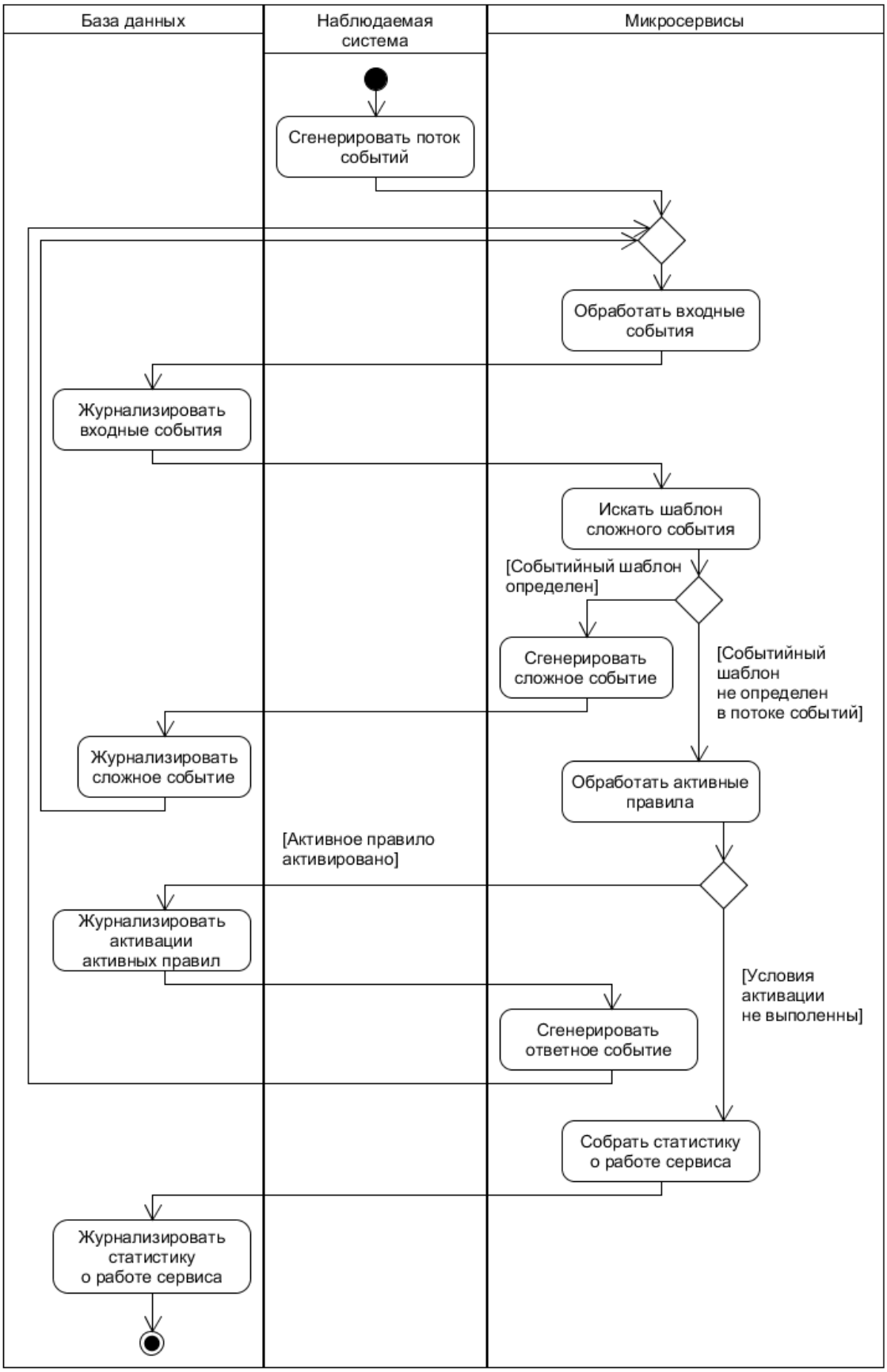


Рисунок 3 – Диаграмма деятельности сервиса потоковой обработки событий

На рисунке 4 представлен сценарий работы сервиса, реализованный в виде диаграммы последовательности.

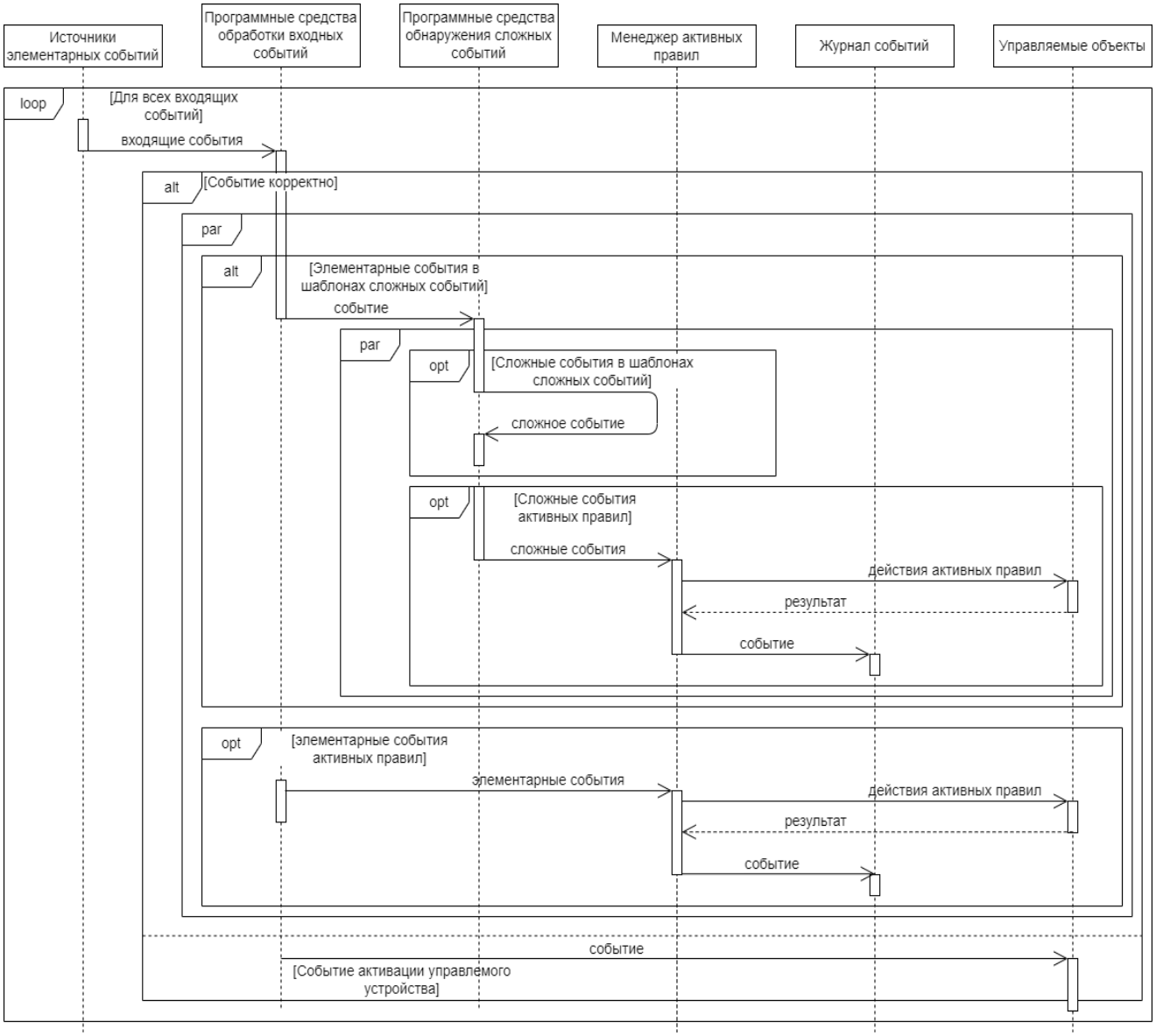


Рисунок 4 – Диаграмма последовательности сервиса потоковой обработки событий

Данная диаграмма более детально показывает связи и обмен сообщениями между микросервисами, а также моменты их активации и результаты обработки.

Сгенерированный поток событий от компонентов-источников подается на вход сервису. Микросервис по обработке входных событий проверяет корректность пришедших данных, после чего средствами микросервиса по обнаружению сложных событий отыскиваются и обрабатываются совпадения

в потоке событий с заранее заданными событийными шаблонами. При нахождении шаблона, в системе генерируется сложное событие, которое также переходит в поток на обработку. Далее все события подаются на вход менеджера активных правил, который активизирует выполнение активных правил при срабатывании условий. При активации активного правила идет реакция на появление события в потоке, что выражается в посыле управляющих сигналов управляемым объектам.

1.2 Управление метаданными в процессе потоковой обработки событий и исполнения активных правил

1.2.1 Состав метаданных сервиса потоковой обработки событий

Важную роль в процессе функционирования сервиса потоковой обработки событий и исполнения активных правил играют метаданные, описывающие все хранимые и обрабатываемые объекты.

В системе выделяются следующие типы метаданных:

- метаданные событий;
- метаданные компонентов (устройств);
- метаданные систем.

Метаданные событий описывают разные виды событий, приходящих извне от отслеживаемых устройств или генерирующихся внутри пользовательского окружения. Они также подразделяются на типы в зависимости от назначения:

- метаданные элементарных (простых, атомарных) событий;
- метаданные агрегирующих событий;
- метаданные сложных событий.

Метаданные элементарных событий являются самыми базовыми и описывают атомарные события, приходящие от определенного устройства и несущие в себе измерение одного конкретного параметра.

Метаданные агрегирующих событий представляют информацию об агрегирующем событии, которое объединяет в себе несколько элементарных

событий и получает агрегат – обобщение совокупности показателей. Данный объект используется для обнаружения, вычисления или вычленения из информационной последовательности полезных данных.

Метаданные сложных событий отображают в базу данных информацию о сложных событиях, которые генерируются в системе соответствующим микросервисом при обнаружении в потоке событий определенного заданного событийного шаблона. Данный шаблон представляет собой совокупность событий указанных типов, приходящих в систему в четкой последовательности с указанными интервалами времени.

Метаданные компонентов описывают интегрированные в пользовательское окружение устройства, измерительные датчики и другие приборы, посылающие в систему потоки событий.

Метаданные систем же хранят информацию о системах, в которых располагаются отслеживаемые устройства. Система может представлять собой отдельно помещение, здание или же подсистему компонентов в большой группе устройств.

Кроме того, в базах данных сервиса потоковой обработки событий хранятся данные о пользователях, работающих над окружением, а также вспомогательные метаданные (параметры и контекст событий, уровни и роли компонентов и т.д.).

Предполагается, что большинство видов метаданных могут создавать пользователи с советующими ролями в системе при помощи специальных инструментальных средств и сохранять их в централизованном репозитории метаданных.

1.2.2 Жизненный цикл метаданных сервиса потоковой обработки событий

Жизненный цикл (ЖЦ) метаданных сервиса потоковой обработки событий начинается с момента принятия решения о подключении к сервису внешних систем мониторинга с целью обработки поступающих от них и

событий и систем управления с целью воздействия на них с помощью активных правил.

ЖЦ метаданных включает в себя несколько этапов (рисунок 5):

1) формирование метаданных и размещение их в центральном репозитории;

2) комплексный анализ метаданных;

3) преобразование метаданных;

4) актуализация метаданных;

5) деактуализация метаданных.

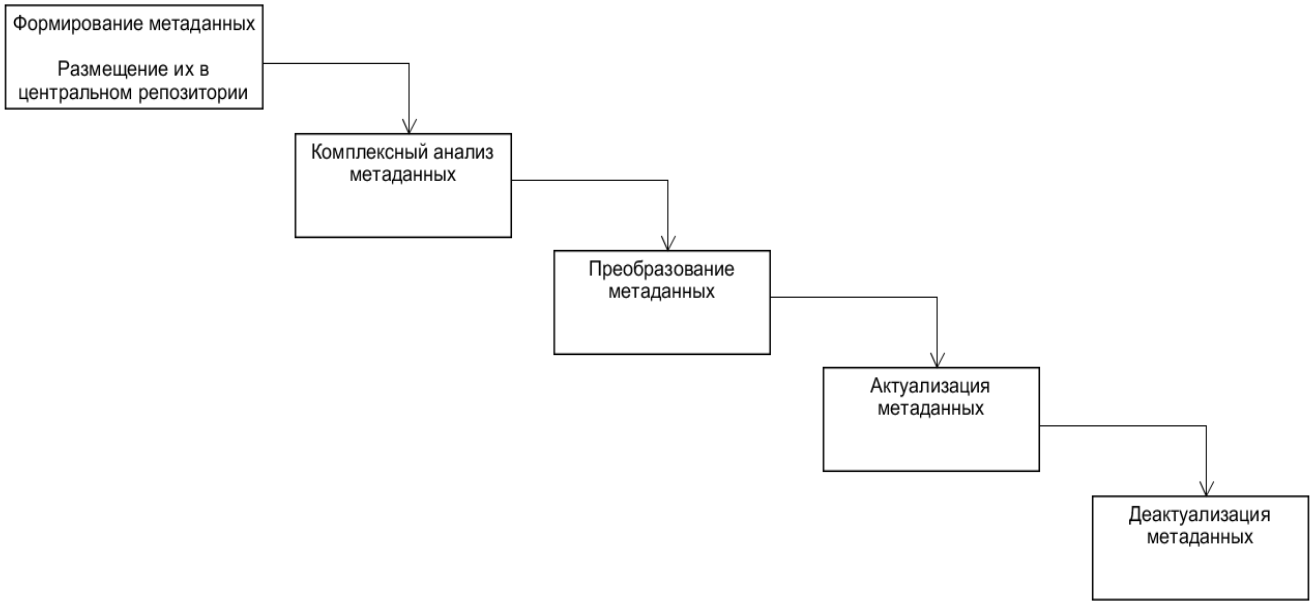


Рисунок 5 – Жизненный цикл метаданных сервиса потоковой обработки событий

Этап формирования метаданных включает следующие операции:

– создание/модификация объектов метаданных;

– автономная проверка объектов метаданных (проверка ограничений, независящих от других объектов метаданных);

– сохранение объектов метаданных в центральном репозитории.

Этап комплексного анализа метаданных включает следующие операции:

– проверка согласованности с другими объектами метаданных;

– выявление коллизий в метаданных, в том числе анализ коллизий взаимодействия активных правил в процессе их исполнения.

Этап преобразования метаданных включает следующие операции:

– устранение коллизий в метаданных;

– планирование мероприятий для детального контроля возможных коллизий на этапе динамического анализа в процессе исполнения активных правил;

– преобразование метаданных к виду, удобному для практического применения.

Этап актуализации метаданных включает следующие операции:

– включение в использование объектов метаданных;

– доставка метаданных в локальные репозитории компонент сервиса.

Этап деактуализации метаданных включает следующие операции:

– отключение от использования объектов метаданных;

– удаление объектов метаданных из локальных репозиториев компонент сервиса;

– удаление объектов метаданных из центрального репозитория (полное удаление).

В пределах каждого из этапов учитывается семантическая связанность объектов метаданных. Так, например, добавление метаданных типов элементарных событий в репозиторий возможно только после добавления метаданных о системах мониторинга. В свою очередь, удаление метаданных о системах мониторинга возможно только после удаления метаданных типов элементарных событий или невозможно вовсе в зависимости от настроек.

1.2.3 Управление метаданными сервиса потоковой обработки событий

Управление метаданными в основном происходит через пользовательские инструментальные средства управления метаданными. На

рисунке 6 показан процесс управления метаданными в виде диаграммы деятельности.

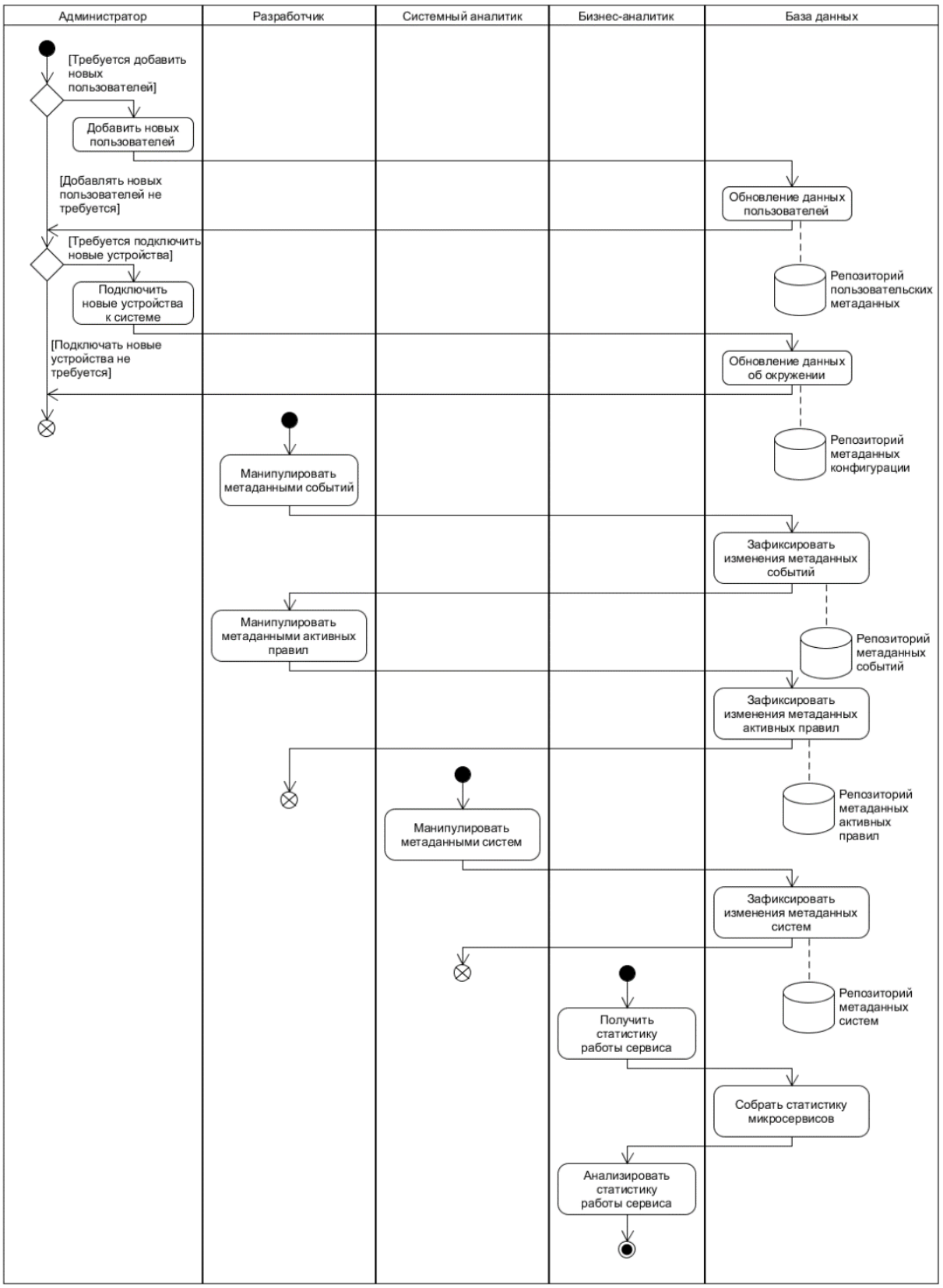


Рисунок 6 – Диаграмма деятельности процесса управления метаданными

Администратор сервиса инициирует создание окружения, добавляет в него пользователей иных ролей, указывает общую информацию о системе и подключает отслеживаемые устройства.

Разработчик создает метаданные об отслеживаемых событиях и событийные шаблоны сложных событий, а также определяет структуру активных правил.

Системный аналитик группирует подключенные устройства, созданные события и активные правила по системам, указывает отслеживаемые системы и анализирует их взаимодействие.

Бизнес-аналитик исследуют и изучают статистику работы сервиса в целом и отдельных микросервисов или групп событий в частности.

1.3 Постановка задачи на разработку инструментальных средств управления метаданными

1.3.1 Цели разработки и назначение программных средств управления метаданными

Перед непосредственным использованием сервиса, физическим подключением к нему устройств и создания потока событий необходима его конфигурация и настройка. При этом необходима всесторонняя подготовка его составляющих: отдельных микросервисов, связывающих программных средств, репозиториев метаданных и т.д.

Также развертывание сервиса под определенную систему требует и наполнение множества баз данных управляющими и информационными данными, над которыми, как следствие, работают различные люди со знаниями в конкретных областях.

Кроме того, после разработки и запуска сервиса в работу с определенным окружением, необходимо расширять его новыми элементами при необходимости, проводить мониторинг и анализ состояния и метаданных системы для выявления неисправностей или ошибок в проектировании, интеграции или использовании сервиса.

Данными работами также занимаются определенные специалисты, которых не должны отвлекать технические подробности реализации тех или иных функций в системе.

Внедрение отдельного инструментального средства предоставляет сервису следующие преимущества:

– верификация доступа пользователей к инструментарию сервиса;
– автоматизация большого количества функций сервиса;
– предоставление пользователям упрощенной модели взаимодействия с сервисом по средством пользовательского интерфейса;
– упрощение контроля и мониторинга создаваемого окружения.

Именно поэтому целью разработки программных средств управления метаданными является создание и предоставление пользователям удобного инструмента для манипулирования метаданными системы, а также комплекса программных средств с обширными возможностями мониторинга, контроля и управления созданным окружением.

1.3.2 Анализ программного обеспечения аналогичного назначения

1.3.2.1 Платформа «AWS IoT»

Одним из самых популярных решений для мониторинга и проведения манипуляций над событиями на сегодняшний день является платформа от компании Amazon на сервисе облачных вычислений AWS – «AWS IoT». Платформа предоставляет как коммерческие, так и бесплатные тарифы для подключения устройств к системе.

Для анализа и управления компания предоставляет как web-приложение, доступное с помощью любого современного браузера, так и отдельно инсталлируемое приложение AWS IoT Device [4]. Причем, устанавливаемое локальное приложение может быть развернуто как в виде полноценного программного обеспечения, так и в виде дополнения ко встроенной командной строке операционной системы.

Также разработчики представленной системы разработали специальный SDK, облачную среду выполнения программного кода и развертываемую операционную систему, используемую в микроконтроллерах, для упрощения внедрения сторонних устройств в конструируемую пользователем среду интернета вещей.

Интерфейс пользователя web-версии AWS IoT представлен на рисунке 7.

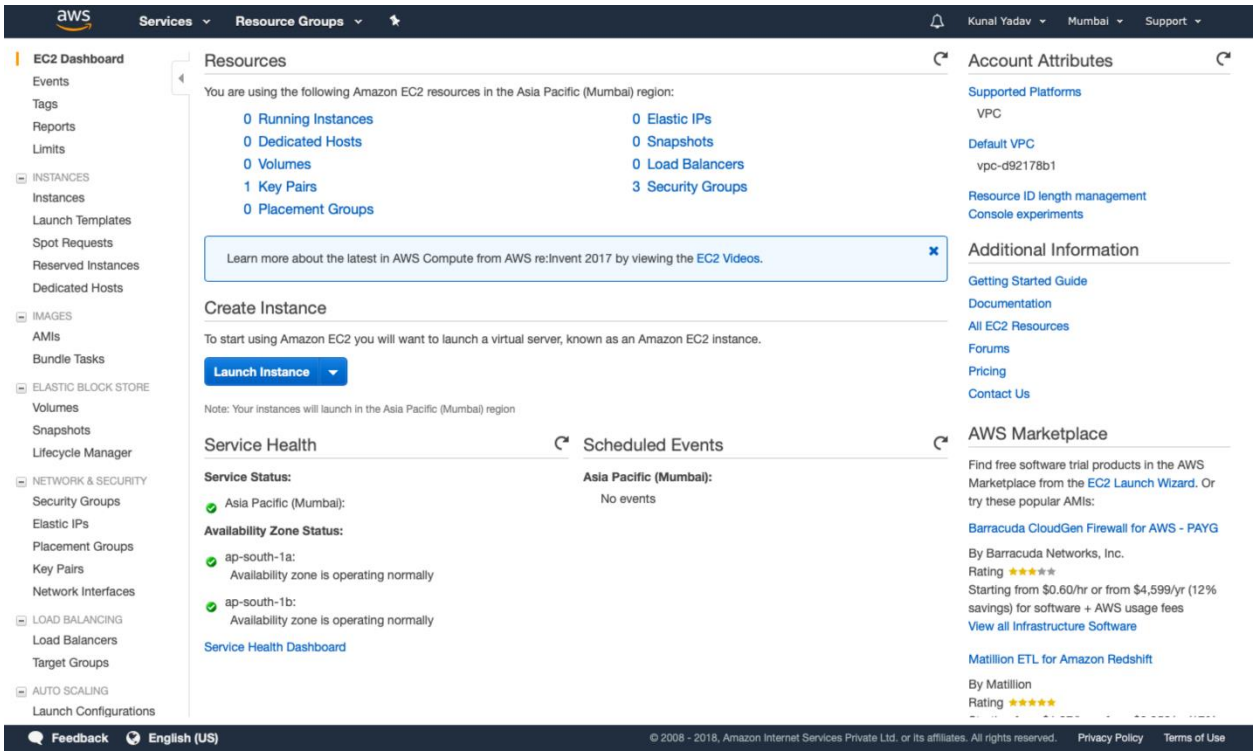


Рисунок 7 – Интерфейс пользователя AWS IoT

Слева располагается панель навигации по сервисам AWS, так как подсистема IoT не единственная, которая использует облачные сервисы Amazon. На главном экране или в панели «Instances» можно увидеть уже подключенные устройства пользователя, а также статистику по ним, или добавить новое устройство с поддержкой AWS IoT SDK.

1.3.2.2 Платформа «Azure IoT»

Azure IoT – это облачная платформа с открытым исходным кодом, разработанная компанией Microsoft, которая предназначена для различных отраслей промышленности, от производства до транспорта и розничной

торговли [5]. Данная система включает в себя службы IoT Central, которые позволяют создавать окружение управляемых устройств по заранее заданным шаблонам.

Рассматриваемое решение бесплатное на начальном этапе, однако для масштабирования создаваемой экосистемы требуется оплата слотов под добавляемые устройства.

Для мониторинга системы Microsoft предоставляет web-приложение IoT Hub, которое также можно установить на компьютер с операционной системой Windows или Linux.

Пользовательский интерфейс рассматриваемой системы представлен на рисунке 8.

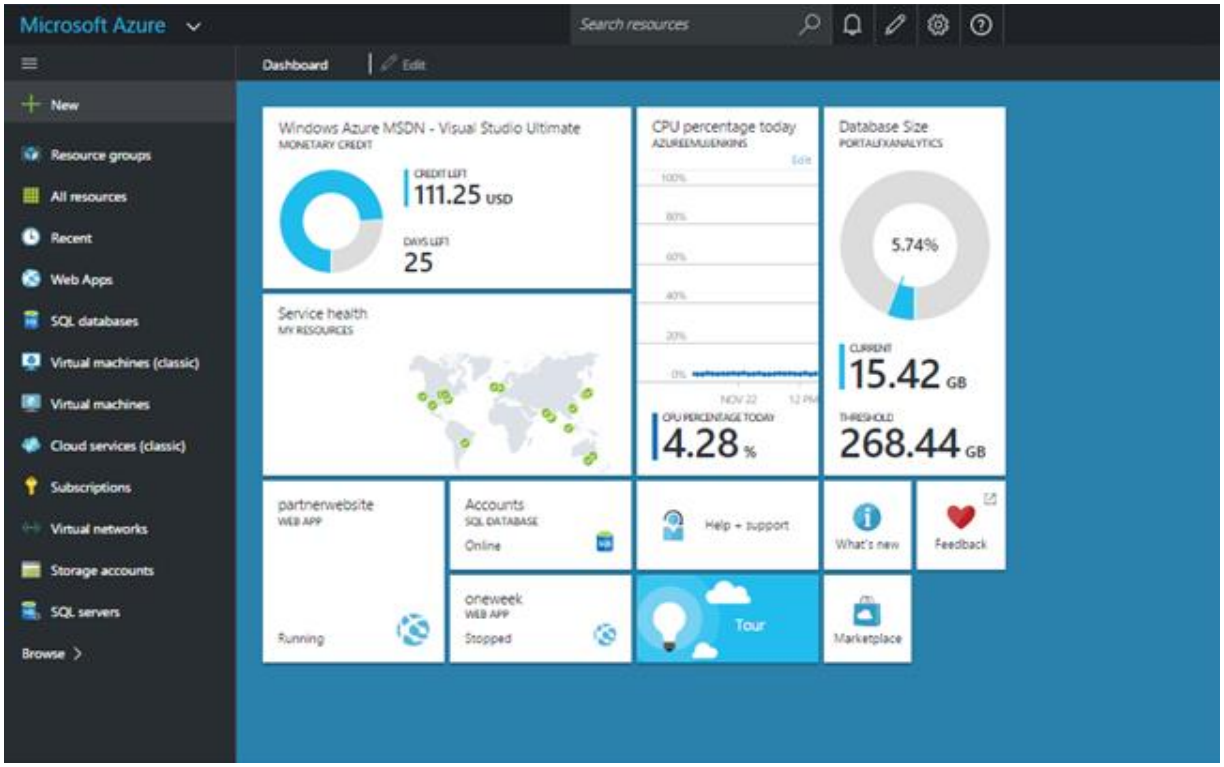


Рисунок 8 – Интерфейс пользователя Azure IoT

На главном экране расположена оперативная информация о состоянии недавно добеленных устройств. В левой части расположены вкладки для перехода по основным разделам программы.

В приложении имеется возможность просмотреть состояние каждого устройства с указанными метриками. Страница мониторинга состояния системы показана на рисунке 9.

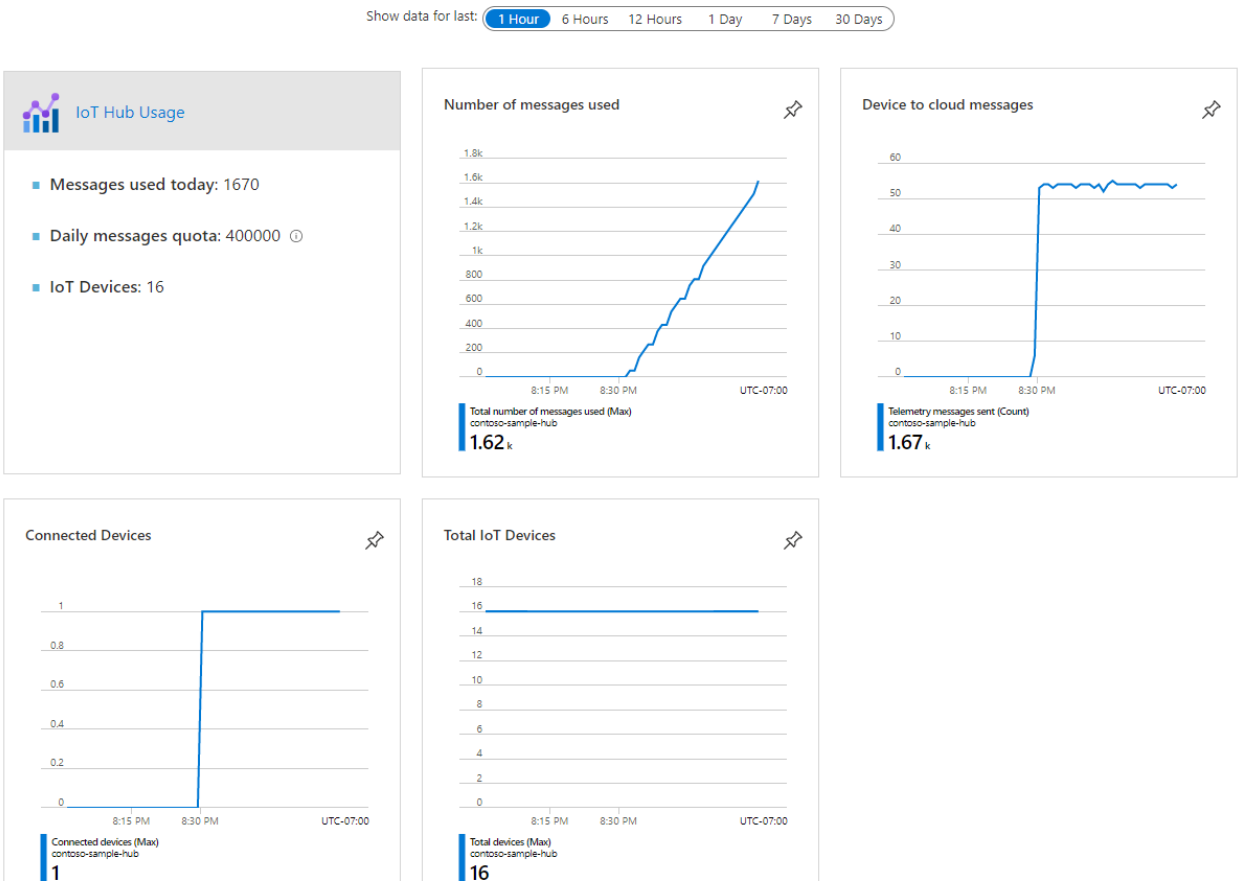


Рисунок 9 – Страница мониторинга

Помимо вышеперечисленного у Azure имеется огромный плюс в виде безопасности. Система предоставляет пользователям аппаратные средства безопасности, многоуровневые программные механизмы защиты, различные виды шифрования и аутентификации.

Однако данное программное обеспечение ориентировано более на сбор и предоставление пользователю статистики по подключенному средству. Возможности по получению и анализу самих событий, приходящих от устройств, а также программирование ответа на данные события в рассматриваемом приложении сильно ограничено. Как и выбор устройств, которые должны быть программно совместимы с данной платформой.

1.3.2.3 Платформа «Cisco IoT»

Как и ранее рассмотренные компании, предприятие Cisco также выпустила свой продукт по созданию экосистемы интернета вещей под названием Cisco IoT. Данная система решает сразу большое количество задач по облачным вычислениям, в том числе обработка событий от подключенных устройств и реакция на них.

Для упрощения подключения устройств к создаваемой системе, сотрудники Cisco разработали Cisco IoT Connected Grid Network – специальную сеть для включения программных модулей систем в пользовательское окружение [6].

Кроме того, было разработано web-приложение для мониторинга подключенных устройств под названием Field Network Director (FND). Пользовательский интерфейс рассмотренного решения представлен на рисунке 10.

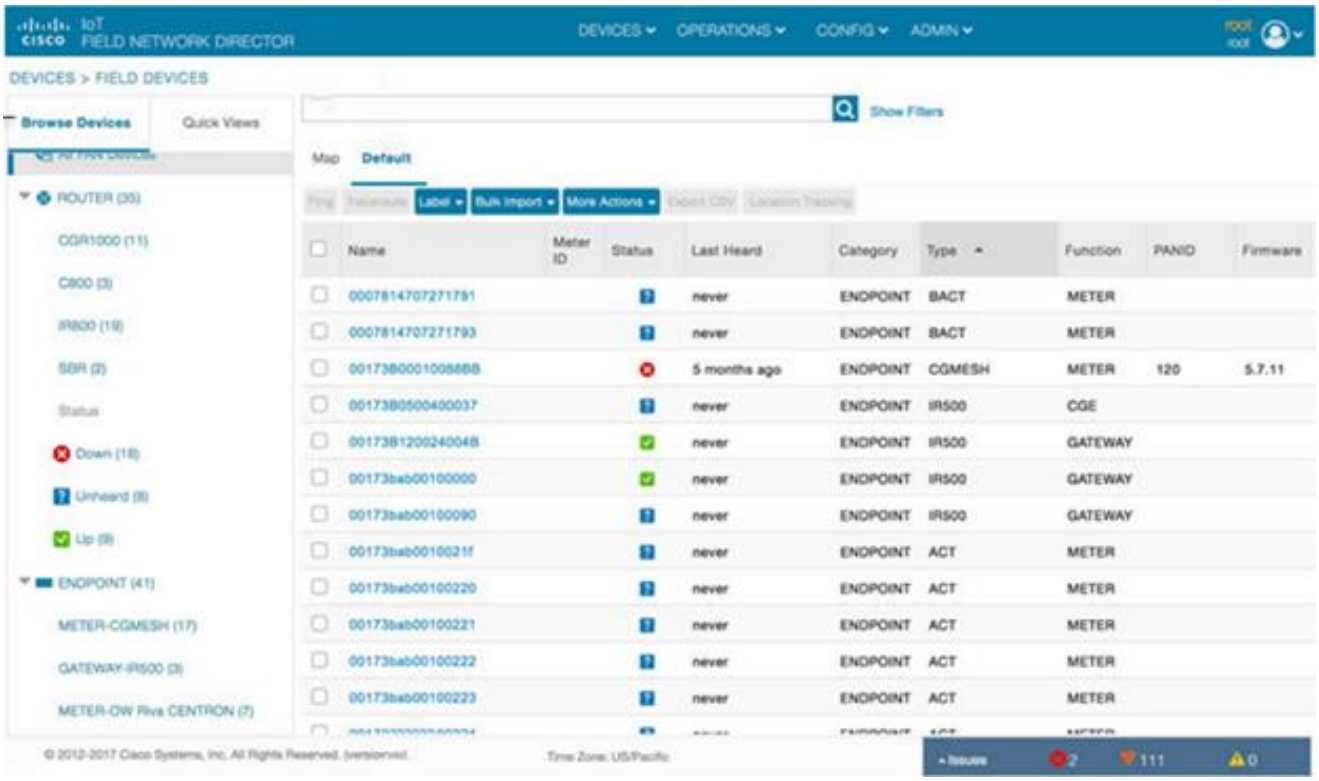


Рисунок 10 – Главная страница Cisco IoT FND

Данное программное обеспечение уже более узко направлено на мониторинг и управление устройств пользовательской среды и предоставляет более полное и удобное представление создаваемого окружения.

В левой части интерфейса располагается дерево устройств, в котором можно иерархически выбрать требуемый аппарат. После чего в центральной области будет выведена таблица датчиков, подключенных к устройству, с главными характеристиками. Выше располагается панель поиска нужного датчика и панель фильтрации таблицы по выбранным критериям.

1.3.2.4 Сравнительный анализ программ-аналогов

После рассмотрения аналогичных программных продуктов были выявлены основные функции, которые должны присутствовать в каждом приложении управления метаданными потоковой обработки событий и активных правил, а также критерии, которым оно должно соответствовать.

Среди критериев, по которым была проведена сравнительная характеристика аналогов, были выделены следующие:

- удобство интерфейса;
- кроссплатформенность;
- подробность отображаемой информации;
- производительность;
- связываемость.

Рассмотрим подробно каждый из приведенных выше критериев.

Критерий удобства интерфейса показывает, насколько результативно конечный потребитель программного продукта будет работать с приложением через пользовательский интерфейс системы, и насколько быстро он будет ориентироваться в программе. Оценка критерия от 0 до 5. Расшифровка оценок критерия представлена в таблице 2.

Таблица 2 – Расшифровка оценок критерия «Удобство интерфейса»

Оценка	Описание
0	Использование пользовательского интерфейса программы максимально неудобно, логика в его построении отсутствует, взаимодействие с элементами данного интерфейса очень сильно затруднено.
1	Интерфейс неудобен в использовании и трудно воспринимаем, логические элементы присутствуют, но расположены в порядке, затрудняющем взаимодействие с ними.
2	Удобство интерфейса на низком уровне, имеются небольшие затруднения при взаимодействии с логическими элементами их трудно воспринимать.
3	Удобство интерфейса на среднем уровне, затруднений при взаимодействии с логическими элементами интерфейса нет, но их трудно воспринимать.
4	Интерфейс удобен для использования, затруднений при взаимодействии с логическими элементами нет, при этом есть небольшие затруднения с их восприятием.
5	Интерфейс максимально удобен для использования, нет затруднений с восприятием его логических элементов и взаимодействием с данными элементами.

Критерий кроссплатформенности показывает, насколько сильно зависит возможность использования программы на ЭВМ от ее аппаратного и программного обеспечения. Оценка критерия от 0 до 2. Расшифровка оценок критерия приведена в таблице 3.

Таблица 3 – Расшифровка оценок критерия «Кроссплатформенность»

Оценка	Описание
0	Программа работает только на ЭВМ одной архитектуры, операционных системах одного семейства.
1	Программа работает на множестве ЭВМ с различной архитектурой и множестве операционных систем разных семейств.
2	Приложение почти не зависит от архитектуры ЭВМ и ее программной или аппаратной реализации

Критерий подробности отображаемой информации показывает, насколько полно предоставляются данные об объектах пользовательской

системы, а также об их состоянии в конкретные моменты времени. Оценка критерия от 0 до 5. Расшифровка оценок критерия представлена в таблице 4.

Таблица 4 – Расшифровка оценок критерия «Подробность отображаемой информации»

Оценка	Описание
0	Информация об объектах системы отсутствует. Отображаются только наименования объектов и связи.
1	Основные данные об объекте показываются пользователю. Статистика по объекту отсутствует.
2	Имеется возможность отобразить достаточное для управления количество характеристик объекта, статистика указывается только за очень короткий период и при этом не сохраняется.
3	Имеется возможность отобразить большинство характеристик объекта, статистика указывается только за последнее недлительное время.
4	Имеется возможность отобразить все характеристики объекта, статистика указывается только за выбранный фиксированный строк.
5	Имеется возможность отобразить полную характеристику объекта со всеми параметрами, а также его статистику по указанному времени из всего цикла существования объекта.

Критерий производительности указывает на то, насколько быстро работает система управления метаданными, насколько сильно она нагружает используемую систему, а также насколько быстро отображает полученную с подключенных устройств информацию и реагирует на нее. Оценка критерия от 0 до 5. Расшифровка оценок представлена в таблице 5.

Таблица 5 – Расшифровка оценок критерия «Производительность»

Оценка	Описание
0	Очень низкая скорость работы и производительность программного продукта, своевременное реагирование на входящие события невозможно.

Продолжение таблицы 5

Оценка	Описание
1	Низкая скорость работы и производительность системы, своевременное реагирование на входящие события невозможно.
2	Скорость работы программы чуть выше низкого уровня, управление статическими элементами системы возможно, но своевременное реагирование на входящие события сильно затруднено.
3	Средняя скорость работы и производительности программы, управление окружением возможно, своевременное реагирование только на простые события, не требующие обработки.
4	Скорость работы выше среднего, управление окружением быстрое, задержки в реакции системы проявляются только на агрегированных событиях.
5	Высокая скорость работы программы. Обработка любых типов событий в реальном времени.

Критерий связываемости показывает возможность подключения к окружению разных типов устройств, различающихся аппаратно или программно. Оценка критерия от 0 до 2. Расшифровка оценок приведена в таблице 6.

Таблица 6 – Расшифровка оценок критерия «Связываемость»

Оценка	Описание
0	Возможно подключение только верифицированных устройств, произведенных компанией-разработчиком рассматриваемого программного обеспечения
1	Имеются средства для разработки специального программного обеспечения или встраиваемые операционные системы для связывания уже использующегося стороннего оборудования с пользовательским окружением
2	Имеется возможность подключать любые сторонние устройства без сильной его подстройки под разрабатываемую экосистему

Результаты сравнения программ-аналогов по представленным выше характеристикам приведены в таблице 7.

Таблица 7 – Сравнительная характеристика продуктов-аналогов

Критерий	AWS IoT	Azure IoT	Cisco IoT
Удобство интерфейса	3	4	5
Кроссплатформенность	2	1 (не работает на MacOS)	2
Подробность отображаемой информации	2	4	4
Производительность	4	5	4
Связываемость	1	0	2

В результате сравнения программ-аналогов можно сказать, что ни одна из рассмотренных программ не обладает характеристиками, необходимыми для полного решения задачи пользовательского управления метаданными событий и активных правил.

Наиболее удобный интерфейс имеет система Cisco IoT. Его интерфейс четко структурирован и разработан под решение представленного рода задач.

Из представленных программных продуктов только Azure IoT не имеет собственного web-приложения и инсталлируется на компьютер пользователя локально. При этом данная система не поддерживается на компьютерах с операционной системой MacOS.

По подробности отображаемой информации высокие результаты показывают сервисы Azure IoT и Cisco IoT, позволяющие анализировать все возможные характеристики объектов окружения и мониторить состояние подключенных устройств.

С точки зрения производительности все рассмотренные аналоги показали хороший результат по обработке как простых, так и сложных событий.

По связываемости устройств с создаваемой экосистемой лидером является Cisco IoT, позволяющий при помощи Connected Grid Network внедрять в окружение устройства без больших затрат времени на их программирование и подготовку. Также хорошо в данном плане показал себя AWS IoT, предлагающий широкий выбор по перепрограммированию устройств, начиная от отдельного SDK, и заканчивая внедряемой операционной системой. Связываемость Azure IoT почти отсутствует, так как система поддерживает только устройства, произведенные специально под данное программное обеспечение.

1.3.3 Перечень автоматизируемых функций инструментальных программных средств

При рассмотрении назначения, роли и места инструментальных программных средств управления метаданными в системе обработки событий, а также анализе программных продуктов аналогичного назначения был выявлен следующий перечень автоматизируемых функций, реализуемый разрабатываемым приложением:

– авторизация пользователя по введенным данным;

– расшифровка JWT для авторизации пользователей;

– предоставление пользователю функционала, зависящего от его роли в системе;

– автоматический сбор информации с полей пользовательского приложения;

– автоматическая валидация введенных пользователем данных;

– автоматическая упаковка данных в JSON-объекты;

– автоматическое обращение к REST-серверу с требуемыми запросами;

– автоматический анализ ответа сервера, систематизация и представление информации из него;

– автоматическое кеширование запрошенных данных.

Также рассматриваемые инструментальные средства должны обладать автоматизируемыми функциями по работе с доставленными от сервера данными на стороне клиента, такими как сортировка или фильтрация по указанным критериям, сохранение данных в локальные хранилища, архивация части метаданных и т.д.

1.4 Планирование процесса разработки инструментальных средств управления метаданными

1.4.1 Выбор технологии и описание жизненного цикла разработки комплекса программных средств сервиса потоковой обработки событий

Технология разработки программного обеспечения – это комплекс организационных мер по созданию программных продуктов требуемого качества в рамках отведенного бюджета и обозначенных сроках. Данная деятельность включает в себя несколько этапов, с которыми так или иначе придётся столкнуться при разработке достаточно крупного программного обеспечения [7].

Одной из основных частей технологии разработки программного обеспечения является модель жизненного цикла. Модель жизненного цикла программного обеспечения характеризует подход команды к разработке программных продуктов. Она отражает акценты и приоритеты во всём процессе изготовления программы, а самое главное, порядок следования этапов создания программных продуктов.

На сегодняшний день существует множество моделей жизненного цикла разработки программного продукта.

Каскадная (водопадная) модель строго следует последовательности всех этапов разработки ПО и не предполагает возвращения с текущего этапа на предыдущий. Также следующий шаг в разработке не может начаться, пока не будет завершен предыдущий. Каскадную модель желательно использовать в проектах, где четко определены требования и не предвидятся какие-либо изменения в процессе разработки.

Модель прототипирования предполагает создание на протяжении всего процесса разработки несколько рабочих версий программы (прототипов) с неполным функционалом. При окончании каждого витка спирали должен получаться готовый, протестированный, дополненный с учетом прошлых пройденных витков, прототип. Удовлетворяющий всем требованиям прототип идет в релиз.

В V-образной модели для каждого этапа разработки существует отдельный уровень тестов. Также имеется возможность вернуться на некоторые этапы разработки и уточнить нужные требования.

Итеративный (спиральный) метод заключается в разработке несколькими «витками». В каждом витке предполагается реализация определенного функционала разрабатываемого ПО при помощи выполнения некоторых этапов разработки, включающих в себя выполнение работ с их параллельным анализом. В ходе выполнения этапов может быть проведена корректировка предыдущих этапов разработки. Итеративная модель используется при построении «гибких» (Agile) методов в разработке ПО, таких как Scrum, DSDM, XP, FDD и т.д.

RAD-модель (rapid application development – быстрая разработка приложений) ориентирована в первую очередь на быстроту и удобство программирования. Команда делает акцент именно на разработке, а большая часть работы по составлению требований и описанию пользователей возлагается на заказчика.

Для реализации инструментальных программных средств сервиса потоковой обработки событий была выбрана каскадная модель, так как сроки разработки существенно ограничены, а требования к итоговому программному продукту конкретно определены. В связи с этим были выделены основные этапы разработки инструментальных программных средств:

– анализ предметной области и постановка задачи на разработку;
– разработка требований и проектирование ПО;

– реализация ПО;
– тестирование;
– интеграция;
– тестирование;
– интеграция ПО в систему обработки событий.

Также в каскадную модель входит этап эксплуатации программного продукта, однако в будущие расчеты данный пункт не будет включен, так как его сложно прогнозировать.

1.4.2 Планирование процесса разработки инструментальных средств управления метаданными

Планирование является одним из важнейших процессов, от которого зависит эффективность разработки и будущей поддержки программного продукта. Без качественного планирования невозможно выполнение работ в срок с требуемым качеством и низким количеством затрат.

Основная цель структурного планирования заключается в описании состава и взаимосвязи работ, которые требуется выполнить для реализации проекта. Кроме того, на данном шаге требуется оценить продолжительности будущих работ.

Далее представлена таблица 8 с перечнем задач работы над проектом.

Таблица 8 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало разработки проекта	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Анализ актуальности разработки	2 дня	1
4	Сравнительный анализ аналогичных инструментов	3 дня	1
5	Постановка задачи на разработку инструментальных средств управления метаданными	3 дня	3;4
6	Анализ предметной области и постановка задачи на разработку закончены	-	5
7	Разработка требований	-	-
8	Разработка функциональных требований	2 дня	6

Продолжение таблицы 8

№	Название задачи	Длительность	Предшественники
9	Разработка требований к пользовательскому интерфейсу	2 дня	8
10	Разработка требований к архитектуре	4 дня	8
11	Выбор технологий и средств разработки	2 дня	10;9
12	Разработка требований к аппаратно-программному обеспечению	2 дня	11
13	Разработка требований закончена	-	12
14	Разработка инструментальных средств управления метаданными	-	
15	Разработка программных средств взаимодействия с сервером	13 дней	13
16	Разработка пользовательского интерфейса	10 дней	13
17	Разработка программных средств реализации основных автоматизируемых функций	11 дней	13
18	Разработка инструментальных средств управления метаданными закончена	-	17;16;15
19	Тестирование	-	-
20	Выбор методов тестирования	1 день	18
21	Проведение тестирования	6 дней	20
22	Тестирование закончено	-	21
23	Интеграция с системой обработки событий	-	
24	Встраивание инструментальных средств в систему обработки событий	1 день	22
25	Тестирование инструментальных средств в качестве микросервиса системы	2 дня	24
26	Интеграция с системой обработки событий закончена	-	25
27	Сдача проекта	-	-
28	Составление документации	5 дней	26
29	Ввод в эксплуатацию	2 дня	28
30	Сдача проекта закончена	-	29
31	Конец проекта	-	30

Таким образом, для реализации проекта требуется выполнить 31 задачу. В таблице 9 представлено описание затрачиваемых ресурсов.

Таблица 9 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Аналитик	Трудовой	150 Р/ч	250 Р/ч
Программист1	Трудовой	200 Р/ч	300 Р/ч
Программист2	Трудовой	200 Р/ч	300 Р/ч
Дизайнер	Трудовой	180 Р/ч	280 Р/ч
Тестировщик	Трудовой	150 Р/ч	300 Р/ч
Компьютер	Материальный	40 000 Р	-

Продолжение таблицы 9

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Бумага	Материальный	500 Р	-
Канцелярские принадлежности	Материальный	250 Р	-
Электричество	Затраты	-	-
Интернет	Затраты	-	-

Распределение ресурсов по задачам представлено в таблице 10.

Таблица 10 – Назначение ресурсов

№	Название задачи	Ресурс	Единицы
1	Начало разработки проекта	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Анализ актуальности разработки	Аналитик Электричество Интернет	50% 50 Р 100 Р
4	Сравнительный анализ аналогичных инструментов	Аналитик Электричество Интернет	50% 200 Р 300 Р
5	Постановка задачи на разработку инструментальных средств управления метаданными	Аналитик; Программист1 Программист2 Электричество Бумага Канцелярские принадлежности	100% 20% 20% 150 Р 1 1
6	Анализ предметной области и постановка задачи на разработку закончены	-	-
7	Разработка требований	-	-
8	Разработка функциональных требований	Программист1 Электричество	100% 100,00 Р
9	Разработка требований к пользовательскому интерфейсу	Программист2 Электричество	100% 150,00 Р
10	Разработка требований к архитектуре	Программист1 Электричество	100% 200
11	Выбор технологий и средств разработки	Программист1 Программист2 Электричество	50% 20% 100 Р

Продолжение таблицы 10

№	Название задачи	Ресурс	Единицы
12	Разработка требований к аппаратно-программному обеспечению	Программист1 Электричество	100% 150 Р
13	Разработка требований закончена	-	-
14	Разработка инструментальных средств управления метаданными	-	-
15	Разработка программных средств взаимодействия с сервером	Программист1 Электричество Интернет	80% 800 Р 1000 Р
16	Разработка пользовательского интерфейса	Дизайнер; Программист2 Электричество Интернет	100% 40% 400 Р 600 Р
17	Разработка программных средств реализации основных автоматизируемых функций	Программист2 Электричество Интернет Программист1	60% 600 Р 1000 Р 20%
18	Разработка инструментальных средств управления метаданными закончена	-	-
19	Тестирование	-	-
20	Выбор методов тестирования	Тестировщик; Электричество Интернет	100% 100 Р 300 Р
21	Проведение тестирования	Тестировщик; Электричество Интернет	100% 500 Р 200 Р
22	Тестирование закончено	-	-
23	Интеграция с системой обработки событий	-	-
24	Встраивание инструментальных средств в систему обработки событий	Программист1 Электричество Интернет	100% 100 Р 200 Р
25	Тестирование инструментальных средств в качестве микросервиса системы	Тестировщик Электричество Интернет	100% 150 Р 200 Р
26	Интеграция с системой обработки событий закончена	-	-
27	Сдача проекта	-	-

Окончание таблицы 10

№	Название задачи	Ресурс	Единицы
28	Составление документации	Аналитик Бумага Канцелярские принадлежности Программист1 Электричество	100% 1 1 50% 300 Р
29	Ввод в эксплуатацию	Программист1 Программист2 Электричество Интернет	50% 50% 100 Р 250 Р
30	Сдача проекта закончена	-	-
31	Конец проекта	-	-

Из представленных таблиц видно, что реализацией проекта будут заниматься 2 программиста, 1 дизайнер, 1 аналитик и 1 тестировщик.

Сетевой график представлен на рисунке 11.

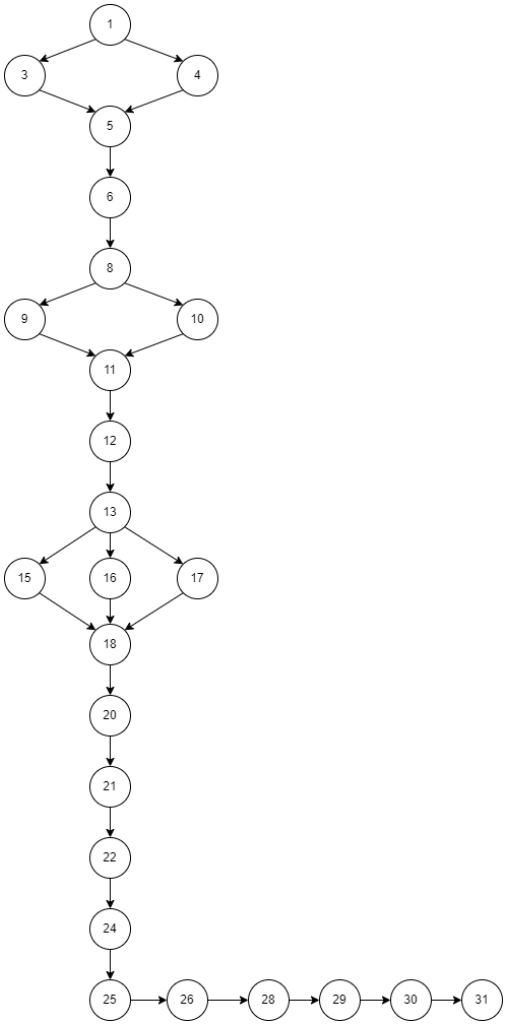


Рисунок 11 – Сетевой график

По данному графику можно увидеть, что некоторые из задач выполняются в параллельном режиме.

1.4.3 Анализ ресурсных затрат на разработку

Для уточнения длительностей работ, а также для расчета трудовых и материальных затрат было проведено календарное планирование в системе управления проектами Microsoft Project [8]. В данную программу были внесены список задач, список ресурсов и распределение ресурсов по самим задачам. Данные продемонстрированы на рисунках 12 и 13.

	Режим задачи	Название задачи	Длительн	Начало	Окончани	Предш	Названия ресурсов
1		Начало разработки проекта	0 дней	Ср 13.04.22	Ср 13.04.22		
2		**Анализ предметной области и постановка задачи на разработку**	**6 дней**	**Ср 13.04.22**	**Чт 21.04.22**		
3		Анализ актуальности разработки	2 дней	Ср 13.04.22	Чт 14.04.22	1	Аналитик[50%];Электричество[50,00 ₽];Интернет[100,00 ₽];Компьютер[3]
4		Сравнительный анализ аналогичных инструментов	3 дней	Ср 13.04.22	Пт 15.04.22	1	Аналитик[50%];Электричество[200,00 ₽];Интернет[300,00 ₽]
5		Постановка задачи на разработку инструментальных средств управления метаданными	3 дней	Пн 18.04.22	Ср 20.04.22	3;4	Аналитик;Программист1[20%];Программист2[20%];Электричество[150,00 ₽];Бумага[1];Канцелярские принадлежности[1]
6		Анализ предметной области и постановка задачи на разработку закончены	0 дней	Чт 21.04.22	Чт 21.04.22	5	
7		**Разработка требований**	**10 дней**	**Чт 21.04.22**	**Чт 05.05.22**		
8		Разработка функциональных требований	2 дней	Чт 21.04.22	Пт 22.04.22	6	Программист1;Электричество[100,00 ₽]
9		Разработка требований к пользовательскому интерфейсу	2 дней	Пн 25.04.22	Вт 26.04.22	8	Программист2;Электричество[150,00 ₽]
10		Разработка требований к архитектуре	4 дней	Пн 25.04.22	Чт 28.04.22	8	Программист1;Электричество[200,00 ₽]
11		Выбор технологий и средств разработки	2 дней	Пт 29.04.22	Пн 02.05.22	10;9	Программист1[50%];Программист2[20%];Электричество[100,00 ₽]
12		Разработка требований к аппаратно - программному обеспечению	2 дней	Вт 03.05.22	Ср 04.05.22	11	Программист1;Электричество[150,00 ₽]
13		Разработка требований закончена	0 дней	Чт 05.05.22	Чт 05.05.22	12	
14		**Разработка инструментальных средств управления метаданными**	**13 дней**	**Чт 05.05.22**	**Вт 24.05.22**		
15		Разработка программных средств взаимодействия с сервером	13 дней	Чт 05.05.22	Пн 23.05.22	13	Программист1[80%];Электричество[800,00 ₽];Интернет[1 000,00 ₽]
16		Разработка пользовательского интерфейса	10 дней	Чт 05.05.22	Ср 18.05.22	13	Дизайнер;Программист2[40%];Электричество[400,00 ₽];Интернет[600,00 ₽]
17		Разработка программных средств реализации основных автоматизируемых функций	11 дней	Чт 05.05.22	Чт 19.05.22	13	Программист2[60%];Электричество[600,00 ₽];Интернет[1 000,00 ₽];Программист1[20%]
18		Разработка инструментальных средств управления метаданными закончена	0 дней	Вт 24.05.22	Вт 24.05.22	17;16;15	
19		**Тестирование**	**7 дней**	**Вт 24.05.22**	**Чт 02.06.22**		
20		Выбор методов тестирования	1 день	Вт 24.05.22	Вт 24.05.22	18	Тестировщик;Электричество[100,00 ₽];Интернет[300,00 ₽]
21		Проведение тестирования	6 дней	Ср 25.05.22	Ср 01.06.22	20	Тестировщик;Электричество[500,00 ₽];Интернет[200,00 ₽]
22		Тестирование закончено	0 дней	Чт 02.06.22	Чт 02.06.22	21	
23		**Интеграция с ситемой обработки событий**	**3 дней**	**Чт 02.06.22**	**Вт 07.06.22**		
24		Встраивание инструментальных средств в систему обработки событий	1 день	Чт 02.06.22	Чт 02.06.22	22	Программист1;Электричество[100,00 ₽];Интернет[200,00 ₽]
25		Тестирование инструментальных средств в качестве микросервиса системы	2 дней	Пт 03.06.22	Пн 06.06.22	24	Тестировщик;Электричество[150,00 ₽];Интернет[200,00 ₽]
26		Интеграция с ситемой обработки событий закончена	0 дней	Вт 07.06.22	Вт 07.06.22	25	
27		**Сдача проекта**	**7 дней**	**Вт 07.06.22**	**Чт 16.06.22**		
28		Составление документации	5 дней	Вт 07.06.22	Пн 13.06.22	26	Аналитик;Бумага[1];Канцелярские принадлежности[1];Программист1[50%];Электричество[300,00 ₽]
29		Ввод в эксплуатацию	2 дней	Вт 14.06.22	Ср 15.06.22	28	Программист1[50%];Программист2[50%];Электричество[100,00 ₽];Интернет[250,00 ₽]
30		Сдача проекта закончена	0 дней	Чт 16.06.22	Чт 16.06.22	29	
31		Конец проекта	0 дней	Чт 16.06.22	Чт 16.06.22	30	

Рисунок 12 – Список задач в Microsoft Project

	Название ресурса	Тип	Единицы измерения материалс	Краткое названи	Группа	Макс. единиц	Стандартна ставка	Ставка сверхурочн	Затраты на использ.	Начисление	Базовый календарь
1	Аналитик	Трудовой		А		100%	150,00 ₽/ч	250,00 ₽/ч	0,00 ₽	Пропорциональ	Стандартный
2	Программист1	Трудовой		П1		100%	200,00 ₽/ч	300,00 ₽/ч	0,00 ₽	Пропорциональ	Стандартный
3	Программист2	Трудовой		П2		100%	200,00 ₽/ч	300,00 ₽/ч	0,00 ₽	Пропорциональ	Стандартный
4	Дизайнер	Трудовой		Д		100%	180,00 ₽/ч	280,00 ₽/ч	0,00 ₽	Пропорциональ	Стандартный
5	Тестировщик	Трудовой		Т		100%	150,00 ₽/ч	300,00 ₽/ч	0,00 ₽	Пропорциональ	Стандартный
6	Компьютер	Материальный		К			40 000,00 ₽		0,00 ₽	Пропорциональ	
7	Бумага	Материальный		Б			500,00 ₽		0,00 ₽	Пропорциональ	
8	Канцелярские принадлежности	Материальный		К			250,00 ₽		0,00 ₽	Пропорциональ	
9	Электричество	Затраты		Э						Пропорциональ	
10	Интернет	Затраты		И						Пропорциональ	

Рисунок 13 – Лист ресурсов в Microsoft Project

На основе введенных данных была построена диаграмма Ганта. За время начала проекта была принята дата 13.04.2022. Диаграмма изображена на рисунке 14.

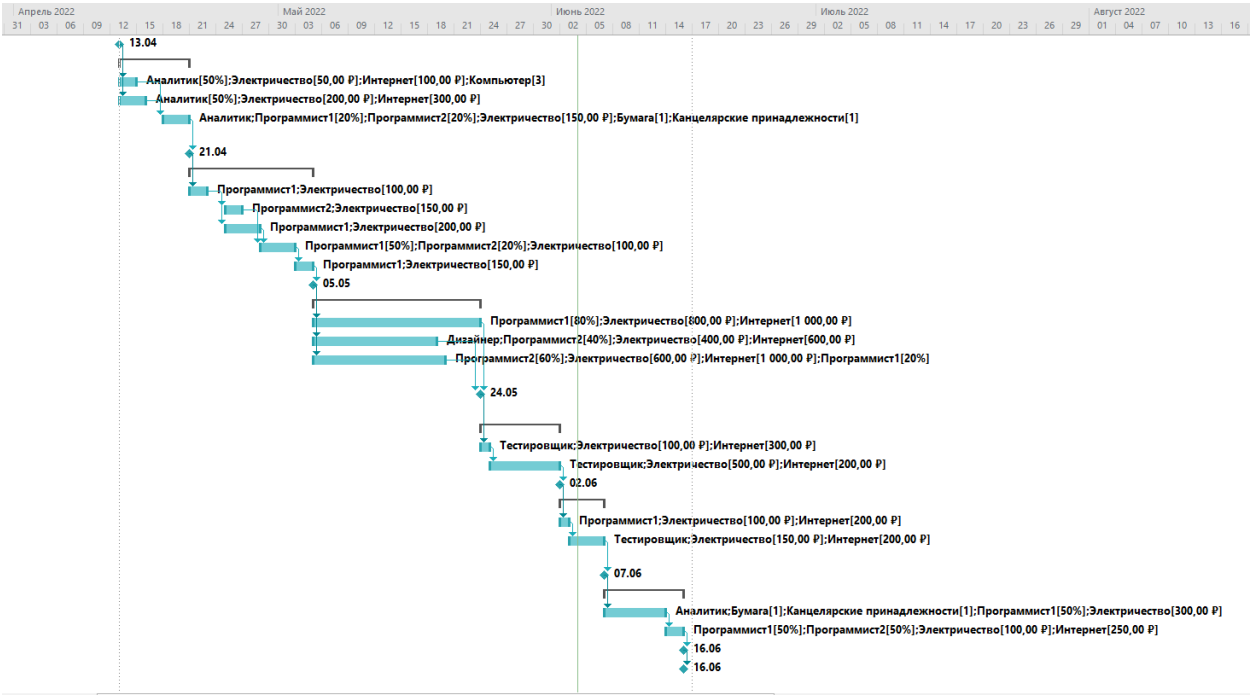


Рисунок 14 – Диаграмма Ганта

На основании данной диаграммы можно сделать вывод, что критический путь проходит через задачи 1, 4, 5, 8, 10, 11, 12, 15, 18, 20, 21, 24, 25, 28, 29.

На рисунке 15 представлен график трудового ресурса «Тестировщик». Графики других ресурсов представлены в приложении Б.

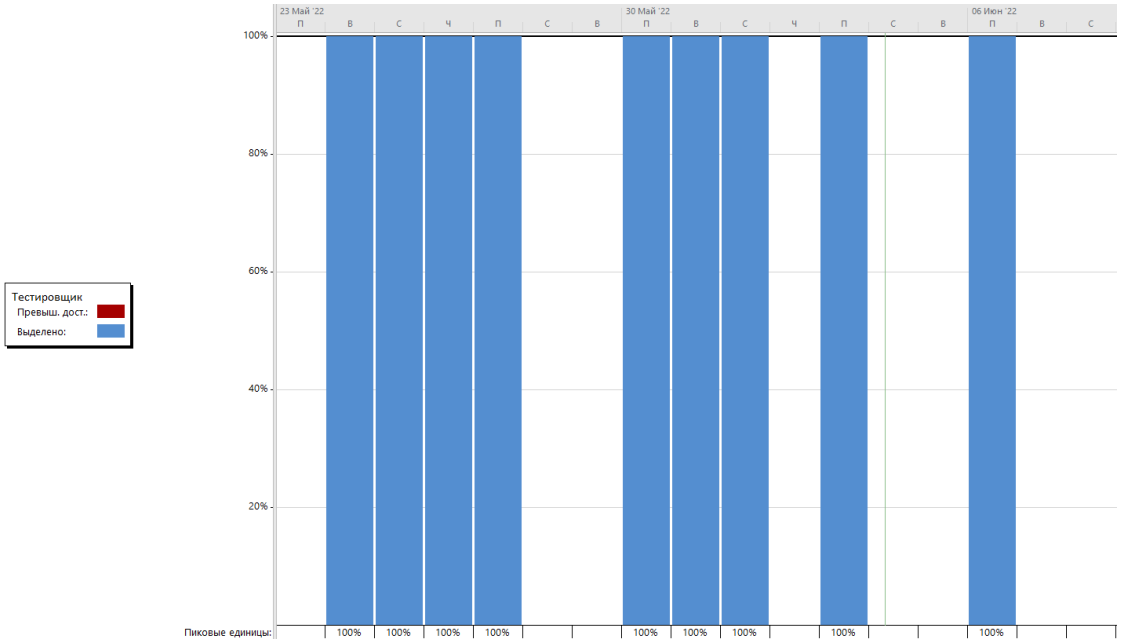


Рисунок 15 – График ресурсов для ресурса «Тестировщик»

Как видно на данном графике и на графиках из приложения Б, перегрузок не у одного из указанных ресурсов не наблюдается, однако

замечены недогрузки в некоторые периоды времени у «Программиста1», «Программиста2» и «Аналитика».

На рисунках 16 и 17 представлены обзоры по денежным затратам проекта.

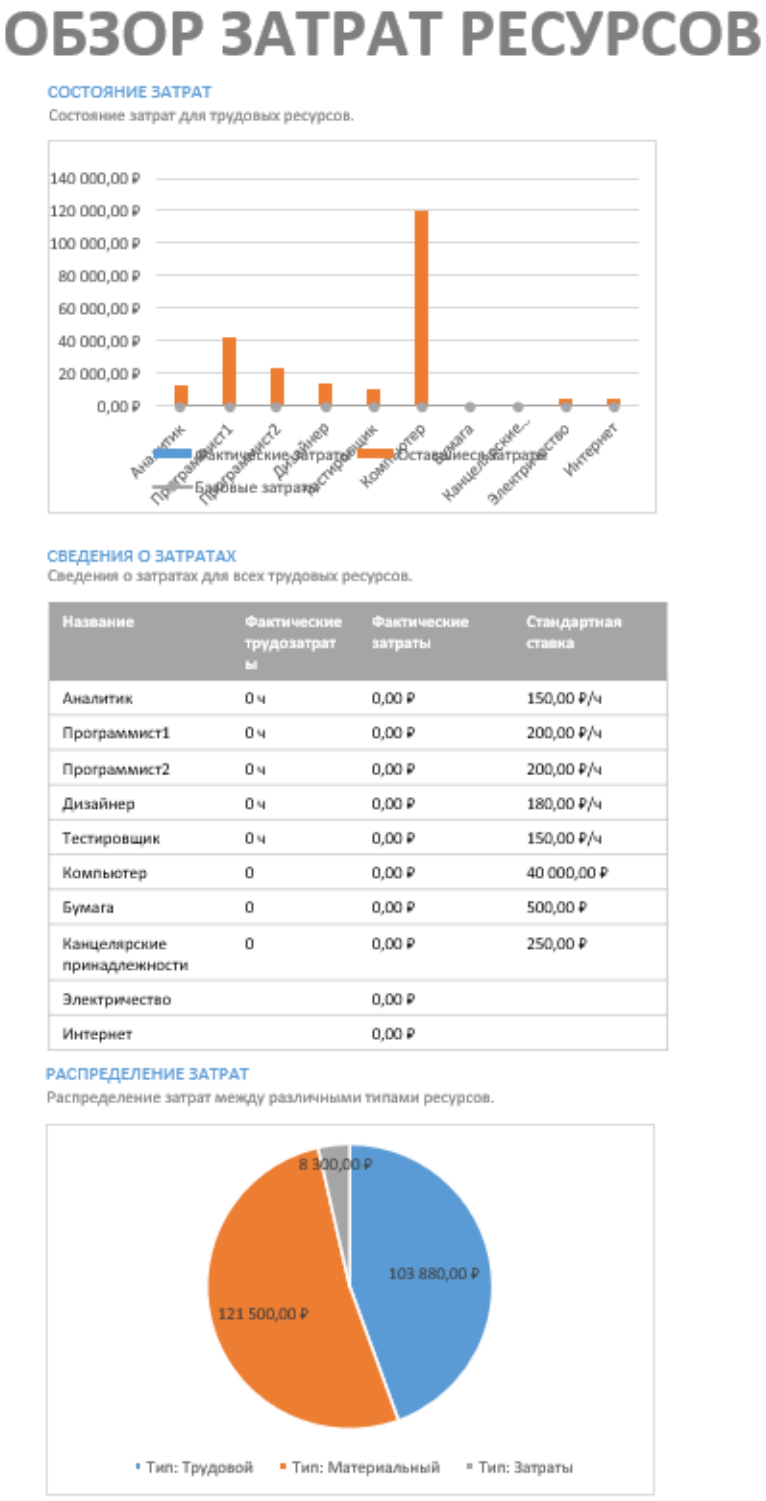


Рисунок 16 – Обзор затрат проекта

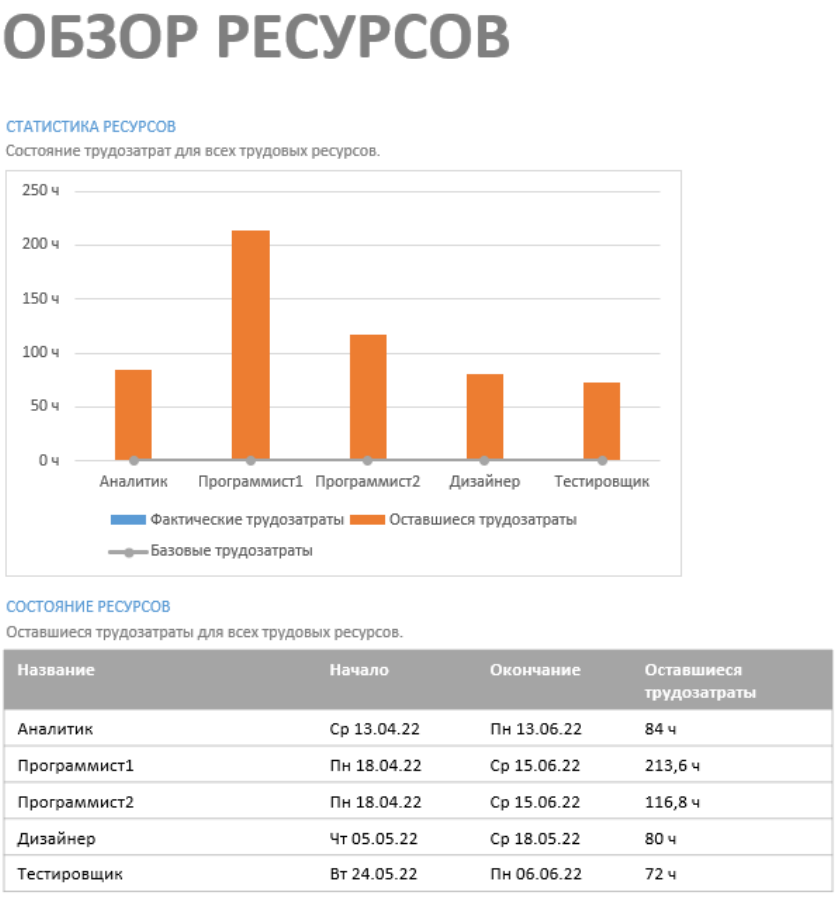
Название	Оставшиеся затраты	Фактические затраты	Затраты	ФСВР	БСВР	БСЗР
Начало разработки проекта	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Анализ предметной области и постановка задачи на разработку	130 070,00 ₽	0,00 ₽	130 070,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Разработка требований	18 940,00 ₽	0,00 ₽	18 940,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Разработка инструментальных средств управления метаданными	55 920,00 ₽	0,00 ₽	55 920,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Тестирование	9 500,00 ₽	0,00 ₽	9 500,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Интеграция с ситемой обработки событий	4 650,00 ₽	0,00 ₽	4 650,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Сдача проекта	14 600,00 ₽	0,00 ₽	14 600,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Конец проекта	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽

Рисунок 17 – Обзор движений денежных средств

Из данных двух данных рисунков видно, что общие предполагаемые затраты на проект составят 233680 рублей.

На рисунках 18 и 19 указаны временные затраты проекта.

Рисунок 18 – Обзор трудозатрат



Название	Начало	Окончание	Оставшиеся трудозатраты
Аналитик	Ср 13.04.22	Пн 13.06.22	84 ч
Программист1	Пн 18.04.22	Ср 15.06.22	213,6 ч
Программист2	Пн 18.04.22	Ср 15.06.22	116,8 ч
Дизайнер	Чт 05.05.22	Ср 18.05.22	80 ч
Тестировщик	Вт 24.05.22	Пн 06.06.22	72 ч

Рисунок 19 – Обзор ресурсов

По данным диаграммам можно определить, что разработка будет длиться 566 часов с 13.04.22 по 16.06.22.

На основе произведенных оценок стоимости и сроков проекта можно сделать вывод о его экономической целесообразности.

2 Разработка требований к инструментальным средствам управления метаданными для сервиса потоковой обработки событий

2.1 Функциональные требования к инструментальным средствам управления метаданными

Так как разрабатываемая система ориентирована на обработку событийных потоков информации, основными ее объектами являются события разных видов, а также активные правила, которые позволяют реагировать на появления определенных событийных шаблонов.

События разделены на следующие типы:

– атомарное (простое) событие;
– агрегированное событие;
– сложное событие.

Разрабатываемое клиентское приложение должно позволять работать специалистам с указанными частями системы, которые определяются в зависимости от роли пользователя в этой самой системе.

Выделяются следующие роли пользователей:

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

Администратор должен иметь возможность влиять на основные составляющие компоненты создаваемого окружения (такие как системы, события, приборы), а также управлять другими пользователями программного продукта.

Для разработчика должна быть реализована функциональность по созданию новых или изменению и удалению уже существующих типов событий и активных правил, связыванию их друг с другом, а также с устройствами, присоединенными к системе.

Системный аналитик должен получать инструмент по организации созданных разработчиком сущностей в отдельные системы, манипулированию и мониторингу уже созданных систем.

Бизнес-аналитику должна быть разработана панель отображения основных свойств системы, а также должны быть дана возможность детализации собираемых и отображаемых отчетов для будущего анализа работоспособности собранного окружения.

Кроме того, пользователь, независимо от своей роли в системе, должен иметь возможность входить в свой аккаунт для работы с окружением, а также изменять настройки своего профиля.

Метаданные, с которыми работают пользователи системы, доставляются при помощи REST-сервера. Именно поэтому инструментальные средства управления метаданными должны также обрабатывать ответы, получаемые от промежуточного сервера.

Исходя из описанных требований была разработана диаграмма вариантов использования, а также описаны спецификации каждого прецедента. Пример спецификации представлен на рисунке 20, где изображен вариант использования «Войти в портал управления».

Войти в портал управления
Id: 1
Описание: Пользователь входит в существующий аккаунт или регистрирует новый
Действующие лица: Пользователь (роль не важна), REST-сервер
Предусловия: Пользователь не входил ранее в аккаунт на данном устройстве или он ранее вышел из приложения
Основной поток: 1. Пользователь заходит на портал 2. Пользователь вводит Email-адрес 3. Пользователь вводит пароль 4. Пользователь нажимает на кнопку «Войти» 5. REST-сервер получает идентификационный код от базы данных 6. Система сохраняет пользовательский идентификационный код
Постусловие: Пользователь вошел в портал управления

Рисунок 20 – Спецификация для варианта использования «Войти в портал управления»

Сама диаграмма вариантов использования показана на рисунке 21 [9].

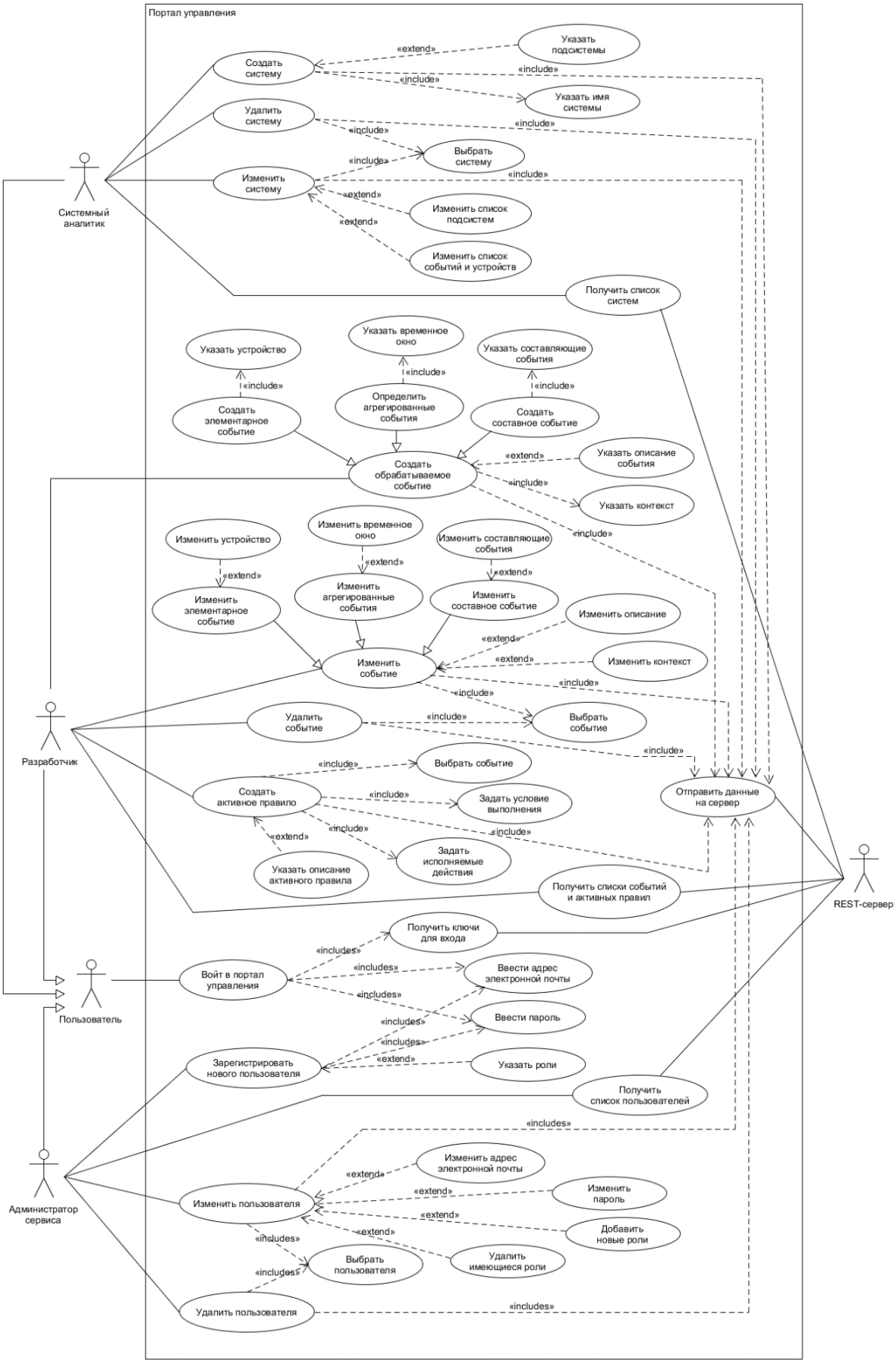


Рисунок 21 – Диаграмма вариантов использования клиентской части портала управления

На рисунке 22 изображена спецификация варианта использования «Зарегистрировать пользователя».

Зарегистрировать пользователя
Id: 2
Описание: Администратор регистрирует нового пользователя в системе
Действующие лица: Администратор, REST-сервер
Предусловия: Администратор вошел в систему
Основной поток: 1. Администратор нажимает на кнопку «Зарегистрировать пользователя» 2. Администратор вводит Email-адрес пользователя 3. Пользователь вводит пароль пользователя 4. Администратор добавляет ячейки для ролей пользователя 5. Администратор выбирает роли пользователя 6. Администратор нажимает на кнопку «Принять» 7. REST-сервер отправляет данные пользователя в базу данных 8. REST-сервер получает индификационные данные созданного пользователя
Постусловие: Администратор зарегистрировал нового пользователя

Рисунок 22 – Спецификация для варианта использования «Зарегистрировать пользователя»

На рисунке 23 изображена спецификация варианта использования «Создать элементарное событие».

Создать элементарное событие
Id: 3
Описание: Разработчик создает новое элементарное событие в системе
Действующие лица: Разработчик, REST-сервер
Предусловия: Разработчик вошел в систему
Основной поток: 1. Разработчик нажимает на кнопку «Создать событие» 2. Разработчик указывает тип «Элементарное событие» 3. Разработчик вводит описание события 4. Разработчик указывает имя события 5. Разработчик выбирает устройство, генерирующее данное событие 6. Разработчик вводит Email-адрес пользователя 7. Разработчик нажимает на кнопку «Принять» 8. REST-сервер отправляет данные о событии в базу данных 9. REST-сервер получает данные созданного события
Постусловие: Разработчик создал новое элементарное событие

Рисунок 23 – Спецификация для варианта использования «Создать элементарное событие»

На рисунке 24 изображена спецификация варианта использования «Изменить систему».

Изменить систему
Id: 4
Описание: Системный аналитик изменяет существующую систему
Действующие лица: Системный аналитик, REST-сервер
Предусловия: Системный аналитик вошел в систему
Основной поток: 1. Системный аналитик выбирает из списка требуемую систему 2. Системный аналитик нажимает на кнопку «Настройки» 3. Системный аналитик добавляет новые подсистемы 4. Системный аналитик удаляет существующие подсистемы 5. Системный аналитик вводит в систему события 6. Системный аналитик вводит в систему приборы 7. Системный аналитик нажимает на кнопку «Принять» 8. REST-сервер отправляет данные об обновлении системы 9. REST-сервер получает подтверждение обновления системы
Постусловие: Системный аналитик изменил существующую систему

Рисунок 24 – Спецификация для варианта использования «Изменить систему»

В результате анализа функциональных требований была составлена диаграмма вариантов использования для клиентской части портала управления. Также были рассмотрены спецификации основных вариантов использования и описаны последовательности действий, производимых при выполнении определенных функций.

2.2 Требования к пользовательскому интерфейсу инструментальных средств управления метаданными

Для удобного доступа пользователей к метаданным сервиса потоковой обработки событий требуется разработать пользовательский интерфейс, который должен быть функционален, понятен и логичен в своей структуре. Также он должен обеспечивать высокую скорость работы и минимальную возможность ошибки пользователей, а соответственно, легкую интерфейсную обучаемость.

Интерфейс должен быть разделен на страницы в зависимости от их функциональности и от ролей пользователя, который на них работает. Основываясь на функциональных требованиях, возникает потребность в разработке следующих страниц:

– страница авторизации;
– страница настроек пользователя;
– страница администратора;
– страница разработчика;
– страница системного аналитика;
– страница бизнес-аналитика.

На странице авторизации требуется расположить формы для ввода идентификационных данных пользователя, а также предусмотреть форму для восстановления пароля при необходимости.

На странице администратора должна фиксироваться информация о пользователях в системе с возможностью удобно ей манипулировать (добавлять, изменять, удалять пользователей), а также данные о других метаданных системы для мониторинга и управления ими.

Страница разработчика должна включать в себя область для отображения метаданных событий и активных правил, через которую возможно манипулирование данными объектами окружения. Данная область, помимо всего прочего, должна быть способна отображать метаданные в различных представлениях (табличном, в виде графа или сети Петри и т.д.). Кроме того, на данной странице должны быть реализованы интерфейсные элементы поиска и сортировки требуемых событий и активных правил.

Страница системного аналитика должна содержать функциональность по созданию и изменению систем, вложенных подсистем и компонентов, которые содержатся в них. Аналогично с рабочей областью разработчика, должна быть предусмотрена возможность поиска, сортировки и фильтрации систем по указанным признакам.

Страница бизнес-аналитика должна показывать специалисту требуемую информацию как в целом о работе сервиса, так и об отдельных его компонентах. Должна быть возможность переключения отображения информации в разных форматах, например, в числовом, тестовом, в виде диаграмм или графиков, где это возможно.

Также требуется предусмотреть в интерфейсе пользователей каждой роли панель, через которую пользователь сможет сменить свою роль, выбрав доступную ему, выйти из аккаунта, а также перейти в меню пользовательских настроек.

В меню настроек пользователя должны быть формы, при помощи которых пользователь сможет изменить данные о себе или о своем аккаунте, например, аватар профиля, пароль, общую информацию о себе и т.д.

2.3 Требования к архитектуре программных средств управления метаданными

Так как инструментальное средство управления метаданными в последующем будет встраиваться в микросервисную архитектуру системы, он также должен разрабатываться в качестве микросервиса и иметь интерфейсы для взаимодействия с REST-сервером, который обеспечивает доступ к базе данных. Это обеспечит простоту его интеграции в сервис, а в будущем упростит расширение и доработку данного программного средства. Архитектура портала управления сервисом потоковой обработки событий должна представлять собой многопользовательское приложение, основанное на использовании REST-API (Representational State Transfer). Данная архитектура лучше всего подходит для многопользовательской системы, в которой имеется единая база данных и возможность работы с ней для нескольких удалённых пользователей. Портал управления включает в себя две части – клиентскую и серверную.

Серверная часть представляет собой специальный REST-сервер. Он является промежуточным звеном между запрашивающим ресурсом и базой данных.

При помощи разработанных интерфейсов, REST-сервер принимает HTTP-запросы от клиента или микросервиса, верифицирует данные запросы, после чего обращается к базе данных, отсылая ему заранее описанный SQL-запрос. Ответ базы данных также обрабатывается, упаковывается в объект

JSON определенной структуры и отправляется микросервису, требующему эти данные.

Для разных видов метаданных, а также для отличных друг от друга проверок этих самых данных, REST-сервер содержит в себе следующие модули:

- модуль регистрации пользователей;
- модуль авторизации и аутентификации пользователей;
- модуль хранения метаданных пользователей и их ролей;
- модуль событий;
- модуль компонентов;
- модуль систем;
- модуль активных правил;
- модуль работы с базой данных;
- модуль работы с микросервисами;
- модуль работы с локальными репозиториями.

Для более быстрого доступа к данным в сервис обработки потоков событий и исполнения активных правил внедрена резидентная система управления базами данных Redis. В данном случае она используется в качестве буферного слоя, позволяющего хранить требуемые метаданные микросервисов локально.

Управляющий модуль данного инструмента развернут на REST-сервере, а на микросервисах располагаются локальные хранилища данных.

Во время создания или обновления метаданных REST-сервер взаимодействует с микросервисами для синхронизации глобального и локальных репозиториев.

При запросе метаданных микросервис сначала проверят наличие требуемой информации в локальном репозитории Redis, и при отсутствии там запрашиваемых данных он обращается к REST-серверу. REST-сервер, во время отправки требуемых метаданных, обновляет локальные репозитории, тем самым актуализируя кэш микросервиса.

Клиентская часть должна быть выполнена в виде web-приложения для его большей доступности. Программное обеспечение на основе web-технологий позволяет пользователям работать с активными правилами и событиями как при помощи браузера, так и в отдельном приложении.

На рисунке 25 представлена схема взаимодействия между компонентами сервиса потоковой обработки событий.

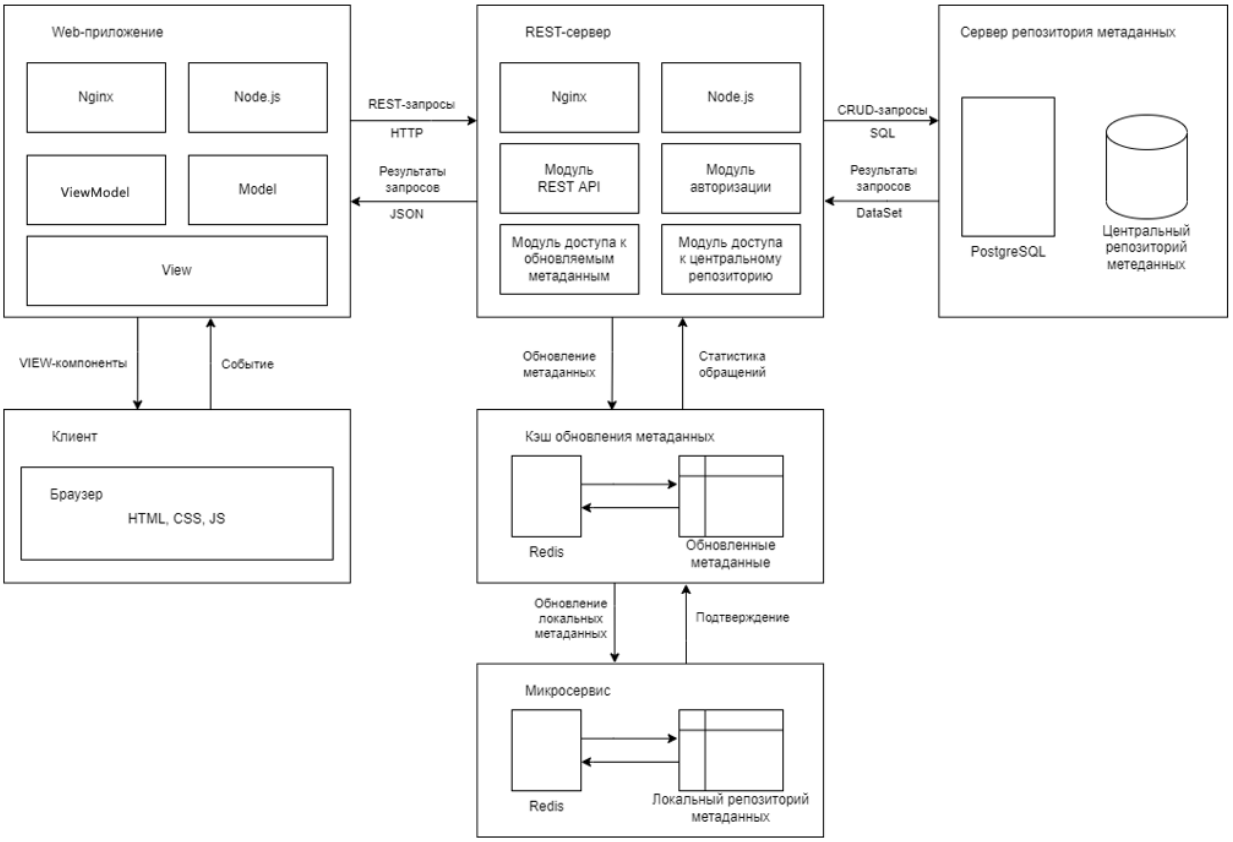


Рисунок 25 – Схема взаимодействия компонентов сервиса

Как можно наблюдать на данной схеме, в структуру взаимосвязей компонентов системы должно быть встроено web-приложение, обеспечивающее двухсторонний обмен информацией о метаданных при помощи JSON объектов с REST-сервером, а также с браузером клиента благодаря событиям представления. Кроме того, клиентское приложение должно быть построено согласно паттерну Model-View-View Model, схема которого изображена на рисунке 26 [10].

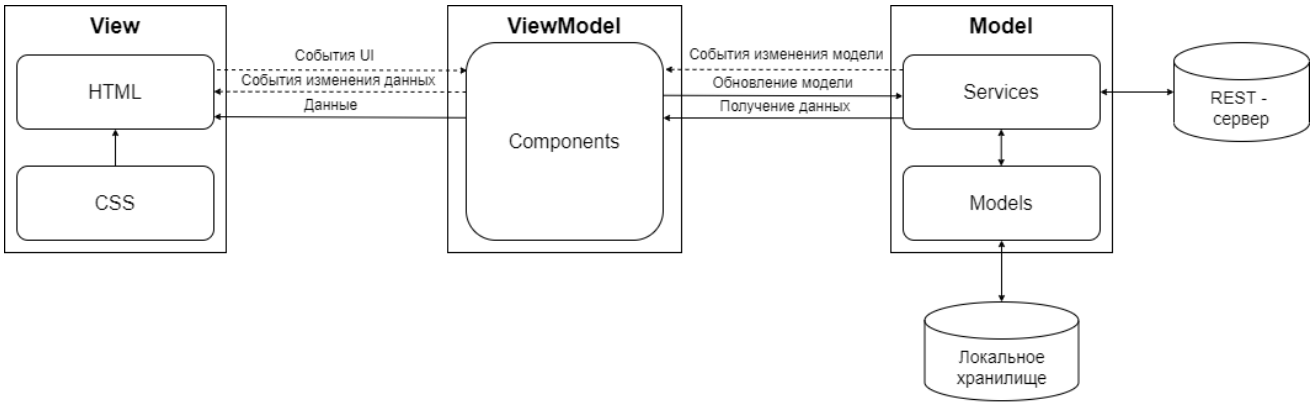


Рисунок 26 – Схема построения приложения на MVVM архитектуре

Исходя из данного паттерна проектирования, представление должно быть связанно с соответствующими свойствами и событиями View-модели. Сама View-модель, в свою очередь, должна реализовывать интерфейс, который используется для автоматического обновления элементов представления. Данные механизмы обмена и отображения данных называются связыванием (binding).

2.4 Обоснование выбора технологий и средств разработки инструментальных программных средств управления метаданными

Исходя из функциональных требований, требований к интерфейсу и общей архитектуре приложения, были выбраны основные технологии, на которых в дальнейшем будет производится разработка инструментальных средств управления метаданными.

Для реализации интерфейса приложения было решено использовать язык гипертекстовой разметки HTML совместно с языком таблицы стилей CSS для создания web-страниц, так как исполнение данных языков поддерживается всеми известными интернет-браузерами [11]. Также они легко интегрируемы с другими популярными языками веб-программирования, хорошо структурированы и синтаксически просты, что сильно упрощает разработку пользовательских интерфейсов.

В связке с HTML и CSS для разработки функциональности сайта было решено использовать языки программирования JavaScript и расширяющий его

возможности TypeScript. Преимущества языка JavaScript состоят в том, что он поддерживается на большинстве операционных системах и во всех известных видах браузеров. Кроме того, JavaScript является интерпретируемым языком, что позволяет не затрачивать время на его компиляцию и уменьшает размер программы. TypeScript расширяет возможности JavaScript, упрощает организацию базового кода для очень больших и сложных программ и вводит статическую типизацию для большей предсказуемости написанного программного кода.

Помимо всего вышеперечисленного, в разработке программного средства используются следующие технологии:

– Angular 12.2 – фреймворк языка JavaScript, позволяющий организовать разработку компонентной системы и являющийся основным инструментом разработки сервиса, с помощью которого появляется возможность писать переиспользуемые компоненты представления или модели данных, применять связывание и упрощать разработку при помощи новых директив;

– vis.js – динамическая библиотека визуализации, позволяющая отображать большие объемы динамических данных (например, в виде графов);

– Jasmine и Karma – средства для написания и запуска тестов. Данные фреймворки встроены в Angular и позволяют писать структурированные и понятные модульные и интеграционные тесты, создавать среды их выполнения, генерировать отчеты по результатам тестирования;

– Bootstrap 4 – HTML, CSS и JS фреймворк, который содержит в себе большой набор стилей и шаблонов для упрощения и ускорения верстки интернет-ресурсов;

– Node.js – среда выполнения JavaScript, расширяющая возможности языка и лежащая в основе многих фреймворков;

– Docker – программное обеспечение для автоматизации развёртывания и управления приложениями;

– Figma – редактор векторной графики с большим набором инструментов для разработки шаблонов пользовательского интерфеса.

Разработка программных средств будет проведена с использованием операционной системы Windows 10 и браузера Google Chrome в среде разработки Microsoft Visual Studio Code.

2.5 Требования к аппаратно-программному обеспечению

Основываясь на выбранных технологиях реализации инструментальных программных средств управления метаданными были обозначены основные требования к аппаратно-программному обеспечению.

Разрабатываемое программное средство будет распространятся в виде web-приложения, доступ к которому будет осуществляться через интернет-браузер.

В связи с этим в таблице 11 представлен список поддерживаемых браузеров, а также версии, на которых будет иметься возможность запустить пользовательские инструментальные средства.

Таблица 11 – Требования к интернет-браузерам

Наименование браузера	Поддерживаемые версии
Chrome	с версии 72.0.3626 и выше
Firefox	с версии 67.0b19 и выше
Edge	с версии 44.18362 для Edge Legacy и с версии 79.0.309 для Edge Chromium
Safari	с версии 13.0 и выше
Opera	с версии 75 и выше
Яндекс.Браузер	с версии 18.2.1 и выше
Internet Explorer	не поддерживается

Исходя из описанных требований к браузерам, на большинстве современных компьютерных систем с выходом в интернет гарантируется работоспособность разрабатываемого программного обеспечения.

3 Разработка инструментальных средств управления метаданными для сервиса потоковой обработки событий

3.1 Разработка архитектуры развертывания инструментальных программных средств

Благодаря анализу выдвинутых требований к разрабатываемым программным средствам были выделены основные компоненты, составляющие инструментальные программные средства. Их структура изображена на рисунке 27.

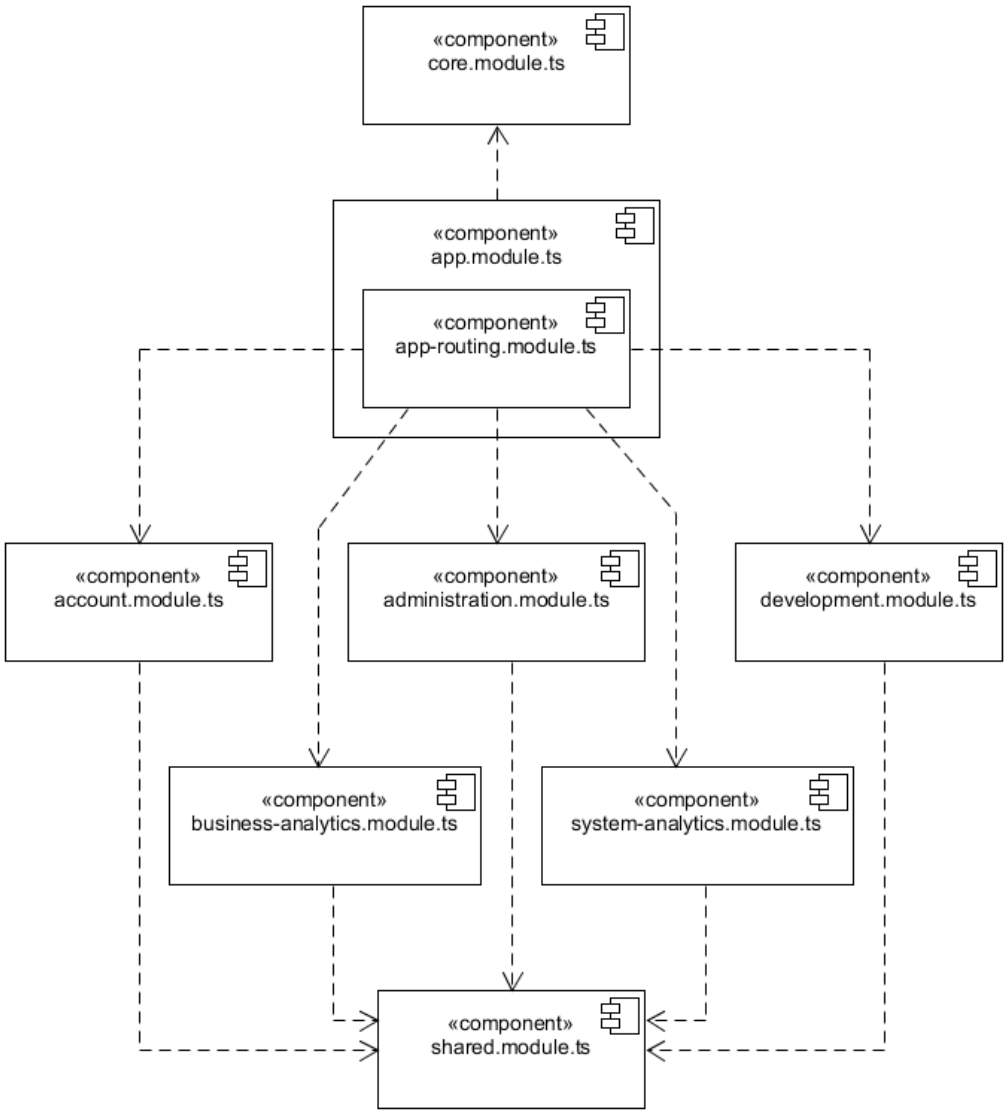


Рисунок 27 – Диаграмма основных компонентов приложения

Основополагающие компоненты инструментальных средств управления метаданными представляют собой контейнеры для программной реализации соответствующих страниц web-приложения, а также дополнительных вспомогательных компонентов:

– app.module – центральный модуль приложения, являющийся базой и содержащий другие компоненты системы;

– account.module – модуль, реализующий страницу авторизации пользователя в системе;

– administration.module – модуль рабочего пространства администратора;

– development.module – модуль страницы разработчика событий и активных правил;

– business-analytics.module – модуль, отображающий страницу бизнес-аналитика;

– system-analytics.module – модуль, содержащий в себе логику страницы системного аналитика;

– core.module – модуль, который хранит в себе сервисы, общие для всех страниц программного комплекса;

– shared.module – модуль с визуальными компонентами, которые переиспользуются на других страницах.

Данные программные модули содержат в себе другие компоненты, которые реализуют более узкоспециализированные задачи, будь то отображение какой-либо области окна, или же реализация функции запроса к определенному серверному интерфейсу.

Более детальное содержание компонентов приведено на диаграмме пакетов, которая изображена на рисунке 28.

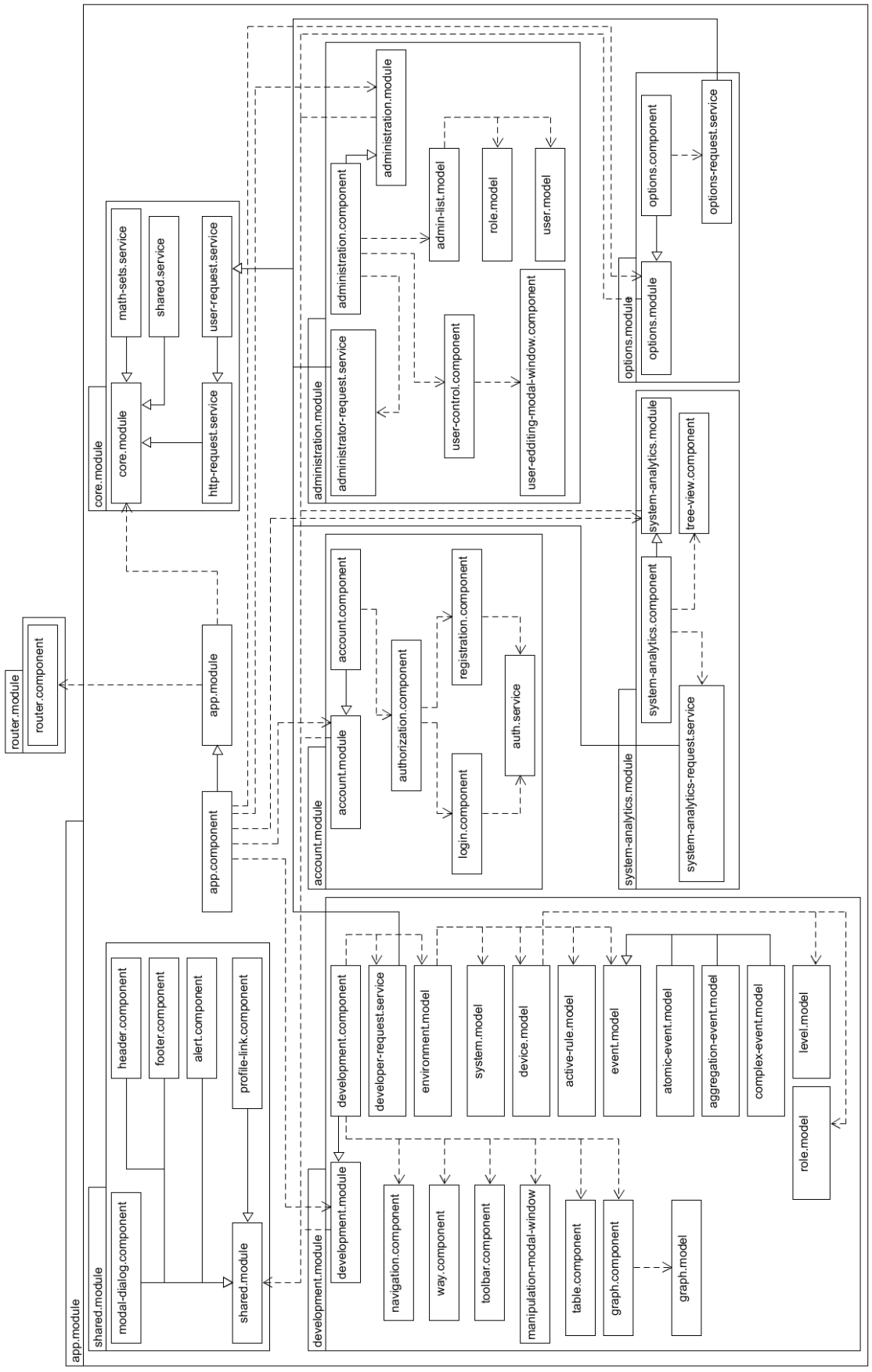


Рисунок 28 – Подробная архитектура элементов приложения в виде диаграммы пакетов

Описание сущностей диаграммы пакетов представлено в таблице 12.

Таблица 12 – Описание сущностей диаграммы пакетов

№	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	router.component	Компонент, управляющий переадресацией на страницы
4	core.module	Модуль, включающий в себя сервисы, экземпляры которых находятся в системе в единичном экземпляре
5	http-request.service	Сервис для обмена HTTP запросами
6	user-request.service	Сервис, предоставляющий HTTP запросы для всех типов пользователей
7	shared.service	Сервис для обмена данными между компонентами и сервисами
8	math-sets.service	Математический сервис для обработки вектором и массивов
9	shared.module	Модуль содержащий переиспользуемые компоненты
10	modal-dialog.component	Компонент простого модального окна
11	header.component	Компонент «шапки» сайта
12	footer.component	Компонент «подвала» сайта
13	alert.component	Компонент окна предупреждения или ошибки
14	profile-link.component	Компонент ссылки на настройки пользователя
15	development.module	Модуль, содержащий компоненты страницы разработчика
16	development.component	Компонент, отображающий и обрабатывающий запросы страницы пользователя
17	developer-request.service	Сервис HTTP запросов разработчика
18	navigation.component	Компонент навигации в окружении
19	way.component	Компонент отображения пути до выбранного объекта
20	toolbar.component	Компонент панели инструментов
21	manipulation-modal-window	Компонент модального окна для редактирования или добавления метаданных
22	table.component	Компонент табличного отображения
23	graph.component	Компонент графового отображения
24	graph.model	Модель для описания графа
25	environment.model	Модель для описания создаваемого окружения
26	system.model	Модель системы
27	device.model	Модель подключаемого устройства
28	active-rule.model	Модель активного правила
29	event.model	Модель общих сведений о событии
30	atomic-event.model	Модель элементарного события

Продолжение таблицы 12

№	Сущность	Описание
31	aggregation-event.model	Модель агрегирующего события
32	complex-event.model	Модель сложного события
33	role.model	Модель роли устройства в системе
34	level.model	Модель приоритета устройства в системе
35	account.module	Модуль, содержащий компоненты страницы авторизации пользователя
36	account.component	Компонент, отображающий и обрабатывающий запросы страницы авторизации
37	authorization.component	Компонент, представляющий основную форму страницы авторизации
38	login.component	Компонент входа в существующий аккаунт
39	registration.component	Компонент регистрации в сервисе
40	auth.service	Сервис HTTP запросов для авторизации
41	administration.module	Модуль, содержащий компоненты страницы администратора
42	administration.component	Компонент, отображающий и обрабатывающий запросы страницы администратора
43	administrator-request.service	Сервис HTTP запросов администратора
44	user-control.component	Компонент таблицы пользователей системы
45	user-edditing-modal-window.component	Компонент модального окна изменения или добавления пользователя
46	admin-list.model	Модель листа администратора
47	role.model	Модель роли пользователя в системе
48	user.model	Модель пользователя в системе
49	system-analytics.module	Модуль, содержащий компоненты страницы системного аналитика
50	system-analytics.component	Компонент, отображающий и обрабатывающий запросы страницы системного аналитика
51	system-analytics-request.service	Сервис HTTP запросов системного аналитика
52	tree-view.component	Компонент, отображающий системы окружения в виде древовидной структуры
53	options.module	Модуль, содержащий компоненты страницы настроек пользователя
54	options.component	Компонент, отображающий и обрабатывающий запросы страницы пользовательских настроек
55	options-request.service	Сервис HTTP запросов страницы настроек пользователя

Данные сущности разделены на три типа: компоненты, сервисы и модели.

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

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

Модели хранят в себе локальные данные, полученные от сторонних источников или сгенерированные самим приложением.

3.2 Разработка программных средств взаимодействия с сервером управления метаданными

3.2.1 Разработка программных средств уровня модели

Для более удобной и быстрой работы над объектами внутри окружения, метаданные о них, полученные от REST-сервера, сохраняются локально в кэше браузера. После входа пользователя в систему, данные считываются из кэша, что занимает намного меньше времени по сравнению с запросом данных от REST-сервера. Рассмотрим некоторые модели метаданных более подробно.

На рисунке 29 представлен диаграмма классов модели списка пользователей, реализуемая для работы администратора сервиса.

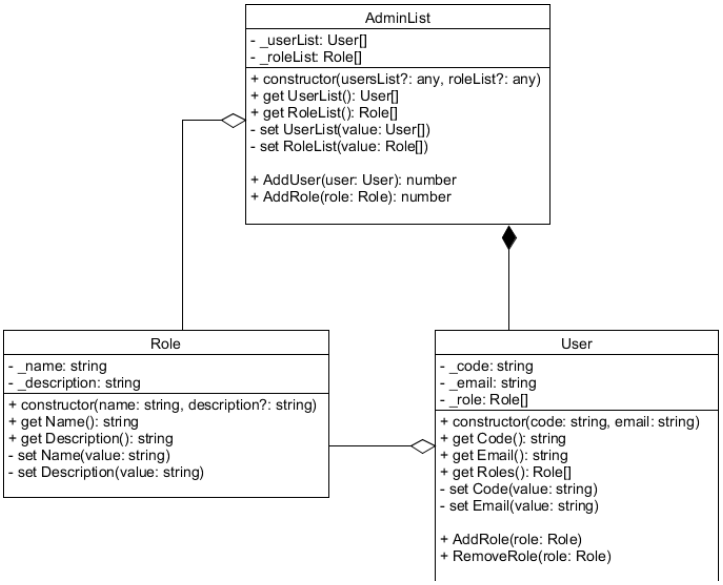


Рисунок 29 – Диаграмма классов модели списка пользователей

Для данной диаграммы классов были составлены спецификации каждого класса. Спецификация класса AdminList представлена на рисунке 30.

Наименование класса: AdminList
Назначение класса: хранение списка пользователя и списка доступных ролей
Поля класса: _userList: User[] – список пользователей системы _roleList: Role[] – список существующих ролей в системе
Методы класса: constructor(usersList?: any, roleList?: any) – конструктор модели для администратора get UserList(): User[] - геттер списка пользователей get RoleList(): Role[] - геттер списка доступных ролей set UserList(value: User[]) - сеттер списка пользователей set RoleList(value: Role[]) - сеттер списка доступных ролей AddUser(user: User): number – функция добавления пользователя в список AddRole(role: Role): number – функция добавления роли в список

Рисунок 30 – Спецификация класса «AdminList»

Спецификация класса Role представлена на рисунке 31.

Наименование класса: Role
Назначение класса: хранение метаданных об определенной роли пользователя в системе
Поля класса: – _name: string – наименование роли – _description: string – описание роли
Методы класса: – constructor(name: string, description?: string) – конструктор роли – get Name(): string - геттер наименования роли – get Description(): string - геттер описания роли – set Name(value: string) - сеттер наименования роли – set Description(value: string) - сеттер описания роли

Рисунок 31 – Спецификация класса «Role»

Спецификация класса User представлена на рисунке 32.

Наименование класса: User
Назначение класса: хранение метаданных о пользователе
Поля класса: – _code: string – уникальный код пользователя – _email: string – email адрес пользователя – _role: Role[] – список ролей пользователя
Методы класса: – constructor(name: string, description?: string) – конструктор пользователя – get Code(): string - геттер уникального кода пользователя – get Email(): string - геттер email адреса – get Roles(): Role[] - геттер списка ролей – set Code(value: string) - сеттер уникального кода – set Email(value: string) - сеттер email адреса – AddRole(role: Role) – функция добавления роли в список – RemoveRole(role: Role) – функция удаления роли из списка

Рисунок 32 – Спецификация класса «User»

Также важной моделью данных является модель метаданных окружения для разработчика событий и активных правил.

Диаграмма, описывающая данную модель, изображена на рисунке 33. Описание классов, представленных на диаграмме, приведено в таблице 13.

Таблица 13 – Описание классов модели окружения

№	Наименование класса	Описание
1	Environment	Класс представления создаваемого окружения
2	System	Класс представления систем
3	Device	Класс представления подключенных устройств
4	Role	Класс роли устройства в системе
5	Level	Класс приоритета устройства в системе
6	ActiveRule	Класс представления активных правил
7	Event	Класс представления общей информации о событии
8	Parameter	Класс параметра контекста события
9	AtomicEvent	Класс представления элементарного события
10	AggregationEvent	Класс представления агрегирующего события
11	ComplexEvent	Класс представления сложного события

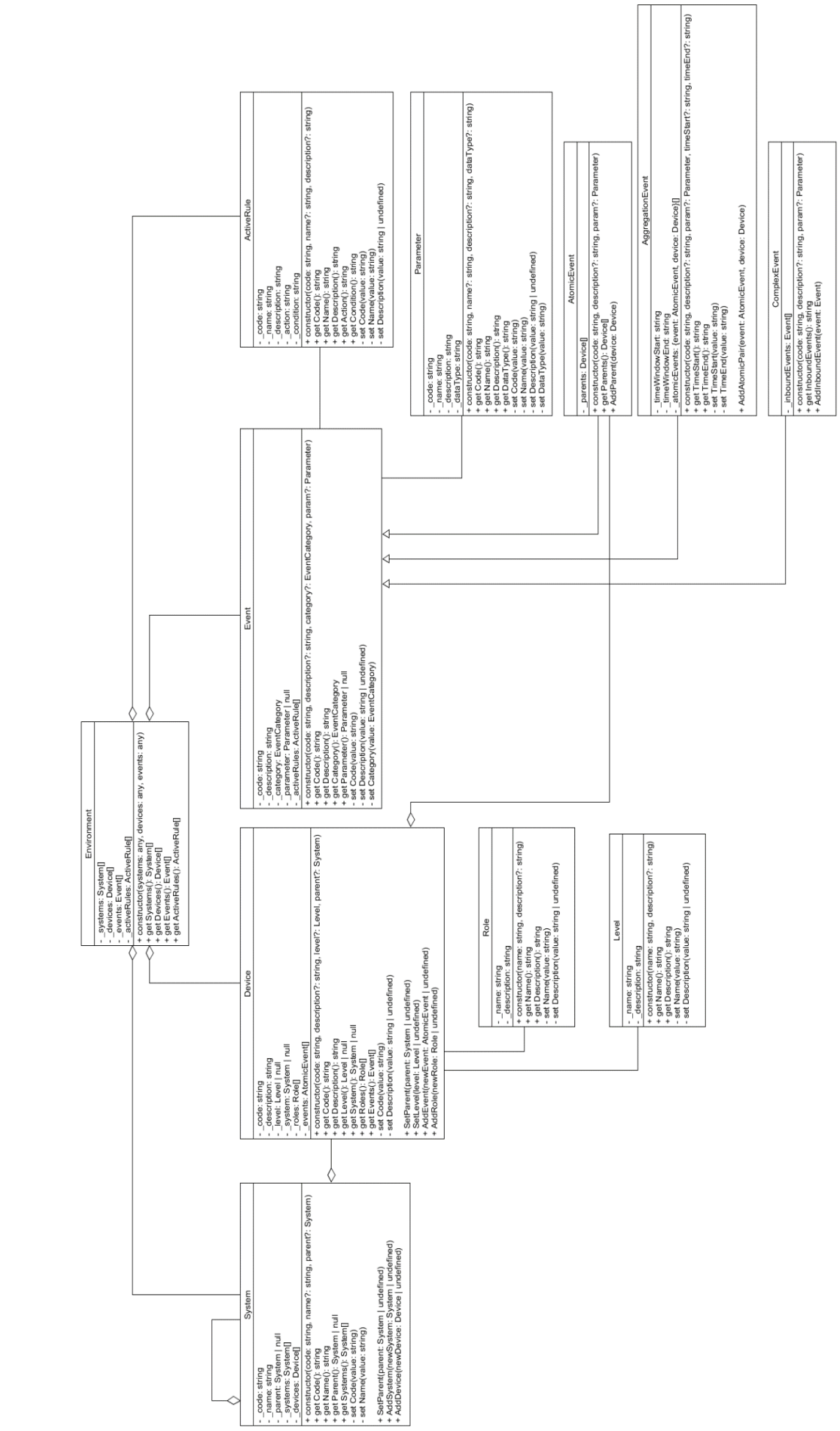


Рисунок 33 – Диаграмма классов модели окружения для разработчика

Программный код реализации моделей в инструментальных программных средствах управления метаданными расположен в приложении В.

3.2.2 Разработка программных средств взаимодействия модели с базой метаданных

Для получения метаданных с REST-сервера и их обновления как в локальном хранилище, так и в удаленном репозитории, были разработаны специальные сервисы, осуществляющие обращения к REST-серверу при помощи HTTP запросов через реализованные API сервера.

Диаграмма классов данных сервисов представлена на рисунке 34.

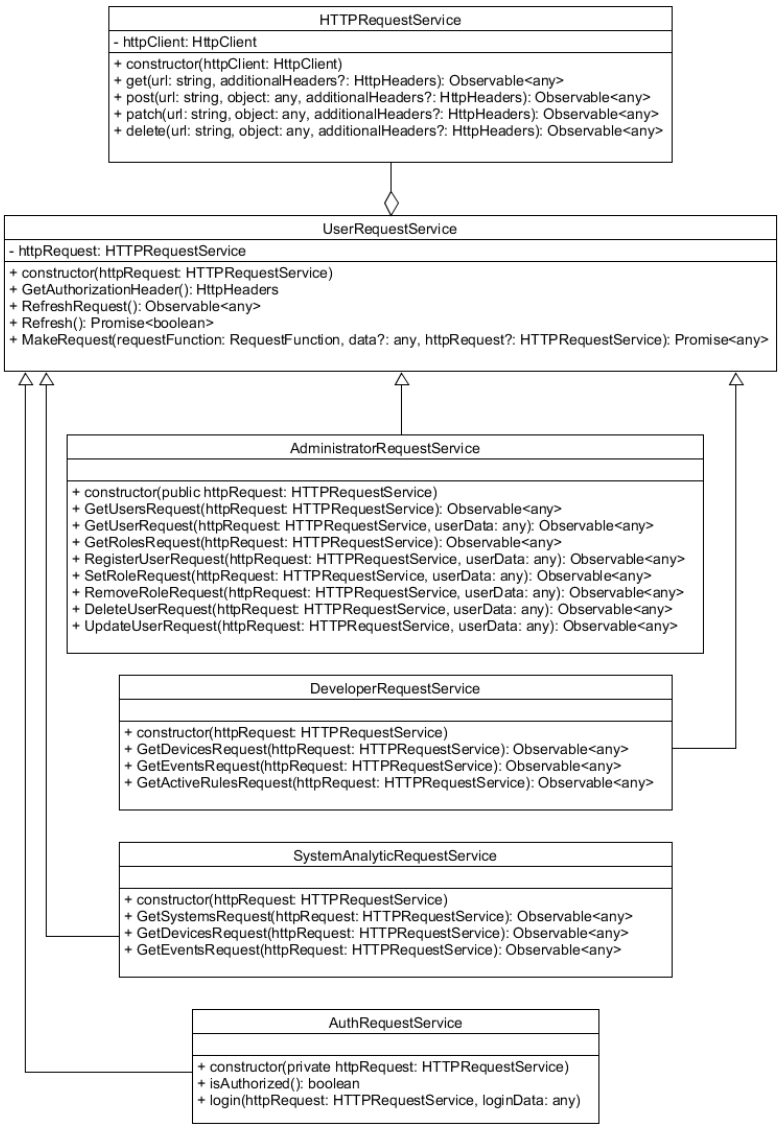


Рисунок 34 – Диаграмма классов сервисов взаимодействия с REST-сервером

Описание сущностей рассматриваемой диаграммы классов представлено в таблице 14.

Таблица 14 – Описание классов сервисов взаимодействия с REST-сервером

№	Наименование класса	Описание
1	HTTPRequestService	Класс сервиса, предоставляющего основные HTTP запросы
2	UserRequestService	Класс сервиса пользовательских запросов, реализующий основные запросы, не зависящие от роли пользователя
3	AuthRequestService	Класс сервиса авторизации пользователя в системе
4	SystemAnalyticRequestService	Класс сервиса для запросов системного аналитика
5	AdministratorRequestService	Класс сервиса для запросов администратора
6	DeveloperRequestService	Класс сервиса для запросов разработчика событий и активных правил

Все функции запросов определенных ролей пользователей обрабатываются при помощи функции MakeRequest. Код данной функции приведен в листинге 1.

Листинг 1 – Код функции MakeRequest()

```
public async MakeRequest(requestFunction: RequestFunction, data?:
any, httpRequest?: HTTPRequestService): Promise<any>
  {
    let requestService: HTTPRequestService;
    if(httpRequest == undefined)
    {
      requestService = this.httpRequest
    }
    else
    {
      requestService = httpRequest;
    }

    let _callback: {isError: boolean, responce: any} = {isError:
false, responce: {}};
```

```
    await requestFunction(requestService, data).toPromise()
    .then
    (
      (responce:any) =>
      {
        _callback.responce = responce;
      }
    )
    .catch
    (
      async (error:any) =>
      {
        if(error.error.statusCode == 403)
        {
          await this.Refresh().then
          (
            async (hasRefreshed: any) =>
            {
              if(hasRefreshed)
              {
                await             requestFunction(requestService,
data).toPromise()
                  .then((responce:any) =>
                    _callback.responce = responce
                  })
                  .catch((errorResponce: any) =>
                  {
                    _callback.isError = true;
                    _callback = errorResponce;
                  });
              }
              else
              {
                _callback.isError = true;
              }
            }
          );
        }
        else
        {
          _callback.isError = true;
          _callback.responce = error;
        }
      }
    );

    return _callback;
  }
```

В данную функцию передается требуемый запрос, после чего она возвращает унифицированный системный ответ.

Программный код, реализующий данные сервисы расположен в приложении Г.

3.3 Разработка пользовательского интерфейса

3.3.1 Разработка сценариев взаимодействия пользователя с инструментальными средствами

Так как разрабатываемое инструментальное программное средство реализуется для пользовательского управления системой, оно должно быть ориентировано на человека и обладать логичным, интуитивным интерфейсом, при взаимодействии с которым у пользователей не будут возникать трудности.

Для лучшего понимания и выстраивания юзабилити, а также дальнейшего упрощения проектирования интерфейса были разработаны сценарии взаимодействия пользователя с инструментальными средствами.

Данные сценарии были изображены в виде диаграмм, которые предлагает использовать Головач В. В. в своем материале для создания структуры системы окон (функциональных блоков) пользовательского приложения [12]. На рассматриваемых диаграммах изображаются экраны при помощи прямоугольников и области экрана при помощи прямоугольников с закругленными краями. Основные действия пользователя между блоками отображаются сплошной линией, в альтернативные – пунктирной. Основной поток выделяется цветом, также допустимы зацикливания для повторяющихся элементов.

Рассмотрим один из таких сценариев взаимодействия разработчика с интерфейсом по реализации функции создания метаданных события. На рисунке 35 представлена диаграмма функциональных блоков пользовательского интерфейса, показывающая последовательность действий для исполнения данной функции.

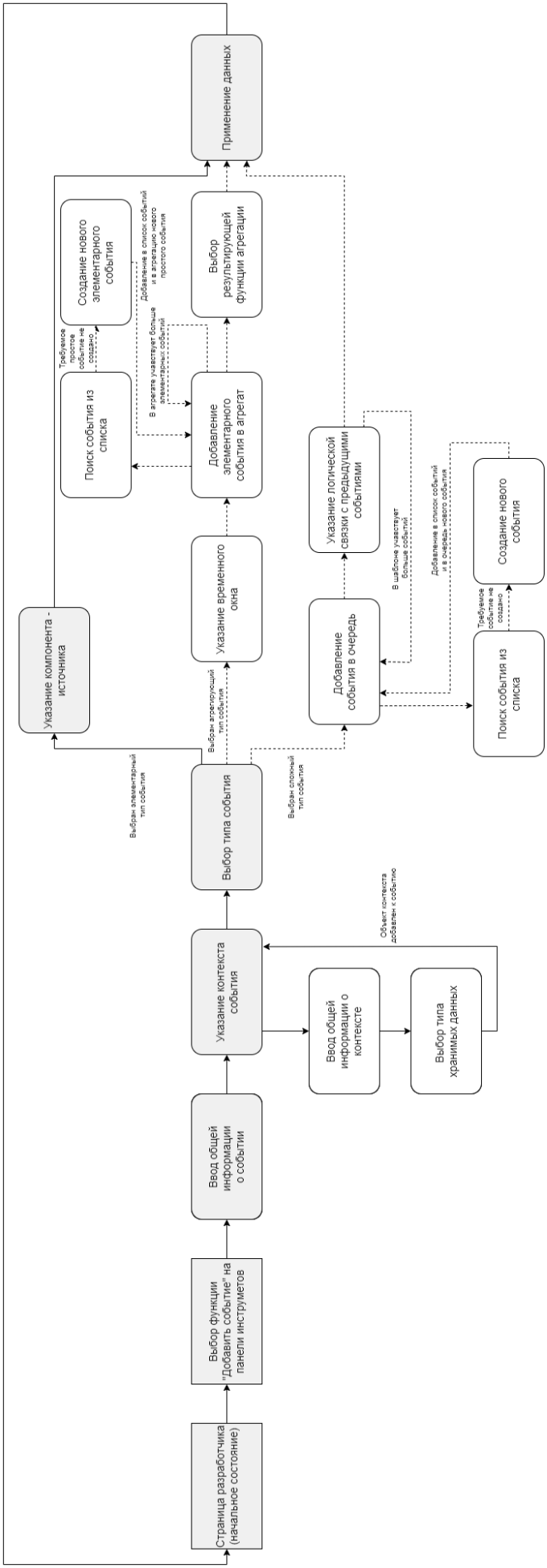


Рисунок 35 – Диаграмма функциональных блоков пользовательского интерфейса при реализации функции «Создать событие»

При помощи данной диаграммы были выявлены основные поля и блоки, которые должны содержаться в интерфейсе пользователя, а также порядок их построения:

– окно для создания события в основном интерфейсе;
– поле для ввода описания события;
– поля для указания контекста события (описания и выбора типа данных констекста);
– поле для выбора типа события;
– поле для выбора компонента-источника;
– поля для указания временного окна;
– список элементарных событий;
– блок очереди событий.

При этом данная диаграмма позволила учесть и блоки для альтернативных путей использования данной функции.

Диаграммы функциональных блоков пользовательского интерфейса для других функций сервиса расположены в приложении Д.

3.3.2 Разрабатка структуры пользовательского интерфейса и прототипов пользовательских страниц и окон

На основе диаграмм функциональных блоков пользовательского интерфейса был разработан графический интерфейс web-приложения инструментального средства управления метаданными.

Данный интерфейс разделен на несколько окон:

– страница авторизации пользователя;
– рабочее пространство администратора;
– рабочее пространство разработчика;
– рабочее пространство системного аналитика;
– страница пользовательских настроек.

В случае первичного входа на web-сервис пользователю открывается страница авторизации. Данная страница представлена на рисунке 36.

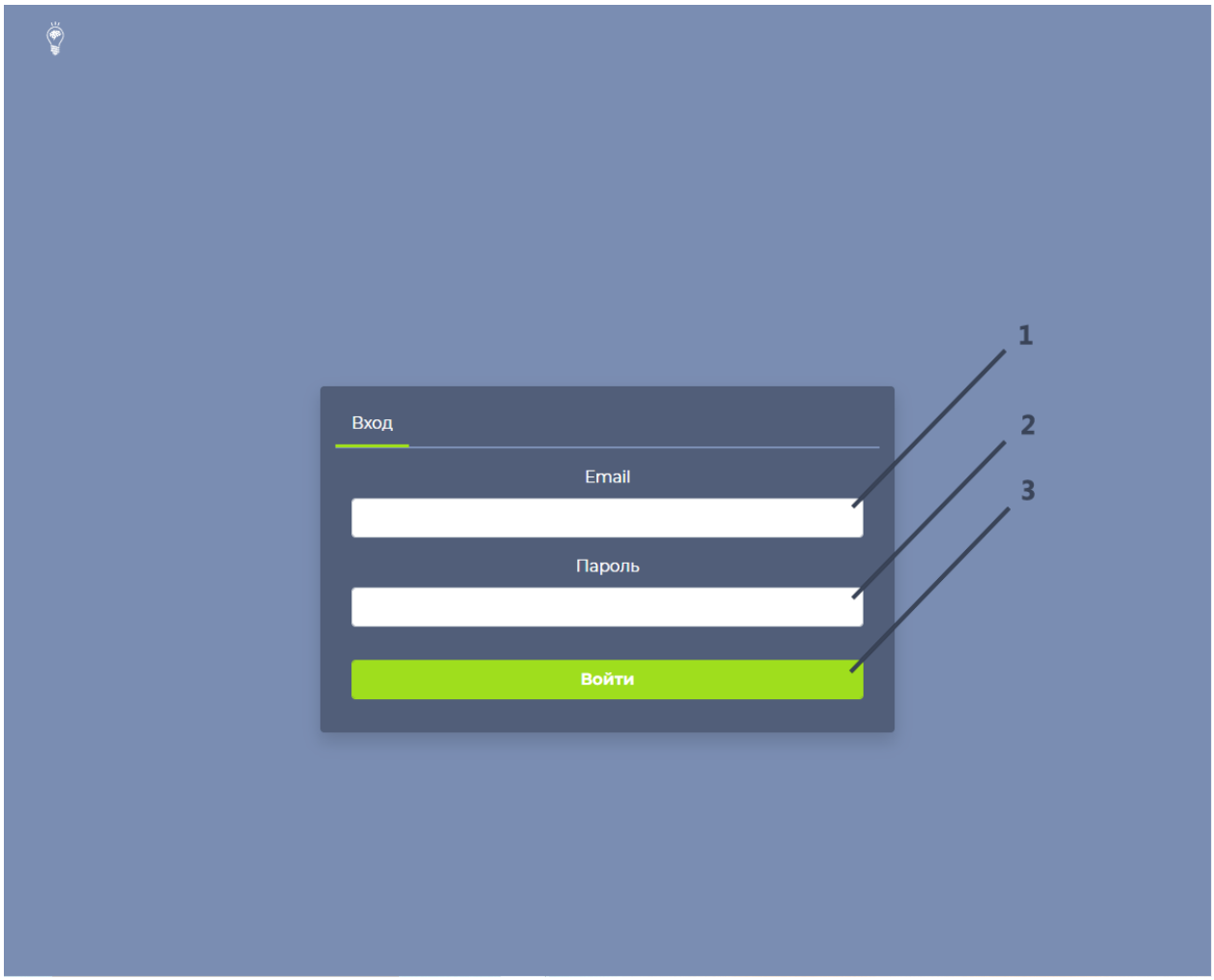


Рисунок 36 – Страница авторизации пользователя

Список элементов, расположенных на странице, показанной на рисунке 37:

1 – поле ввода электронной почты;

2 – поле ввода пароля;

3 – кнопка входа.

При авторизации пользователя переносит на страницу одной из рабочих областей в зависимости от его роли в системе.

Первоначально рассмотрим рабочее пространство администратора сервиса. Оно изображено на рисунке 37.

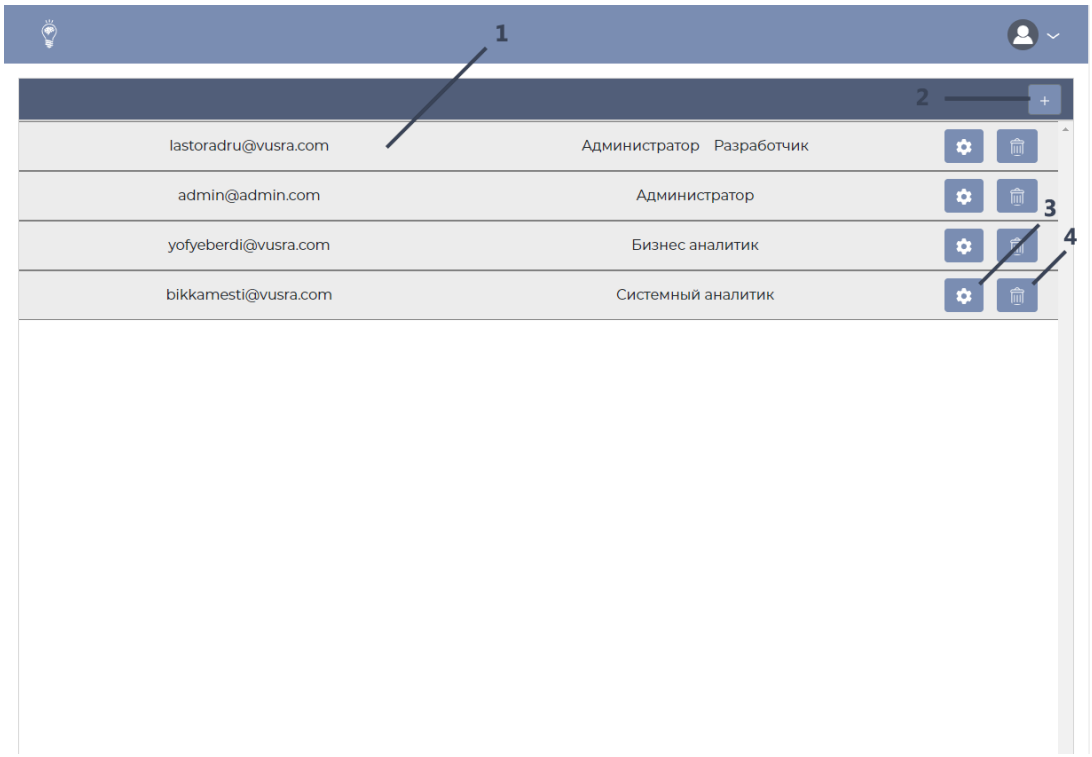


Рисунок 37 – Страница рабочего пространства администратора

Данная страница состоит из следующих элементов:

1 – таблица пользователей, зарегистрированных в системе, с указанием их основных данных;

2 – кнопка регистрации нового пользователя;

3 – кнопка для вызова окна редактирования данных пользователя;

4 – кнопка удаления пользователя из системы.

При нажатии на кнопку добавления нового или изменения существующего пользователя появится модальное окно, показанное на рисунке 38.

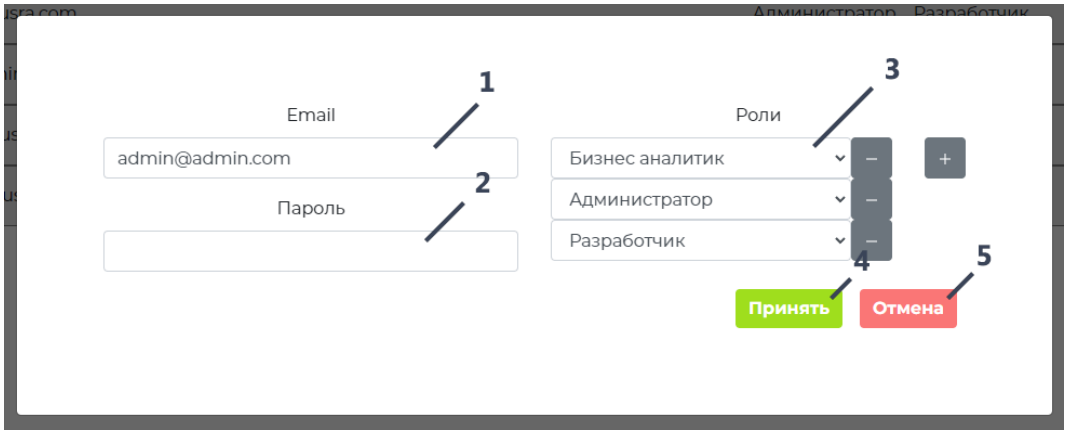


Рисунок 38 – Модальное окно добавления нового пользователя

Основные элементы данного модального окна:

1 – поле электронной почты пользователя;

2 – поле для задания или изменения пароля пользователя;

3 – список ролей пользователя (предоставляет функции добавления, исключение и выбор ролей);

4 – кнопка принять изменения (в зависимости от выбранной функции создает нового пользователя или изменяет существующего);

5 – отмена изменений (закрытие модального окна).

Также при попытке удалить пользователя (как и любой объект в системе) появляется модальное окно с подтверждением операции. Оно изображено на рисунке 39.

Рисунок 39 – Модальное окно подтверждения удаления пользователя

В данном модальном окне располагаются три элемента:

1 – сообщение об удалении определенного объекта системы;

2 – кнопка подтверждения;

3 – кнопка отмены действия.

Разработчик в свою очередь тоже имеет свое рабочее пространство в системе. Данная страница представлена на рисунке 40.

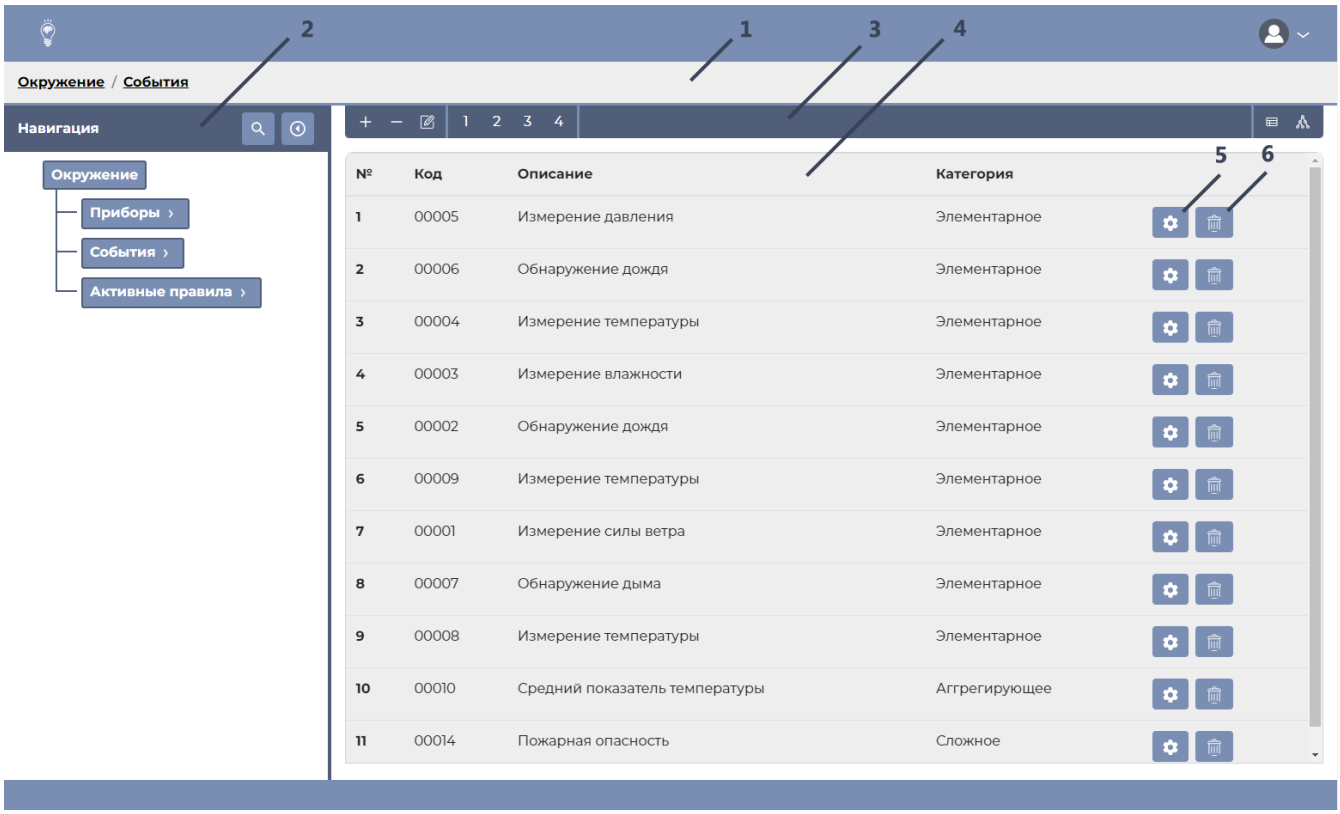


Рисунок 40 – Страница рабочего пространства разработчика

Элементы рассматриваемой страницы:

1 – строка пути до выбранных объектов в системе;

2 – область навигации в элементах окружения;

3 – панель инструментов (предоставляет функции добавления, редактирования и удаления метаданных, а также переключения отображения);

4 – область отображения элементов окружения (может принимать как табличный вид, так и вид графа при выборе определенных метаданных);

5 – кнопка перехода к настройкам объекта;

6 – кнопка удаления объекта системы.

При выборе функции настройки или создания метаданных нового объекта открывается модальное окно. Оно имеет разную структуру при работе с разными объектами системы. На рисунке 41 показано рассматриваемое модальное окно при работе с событиями.

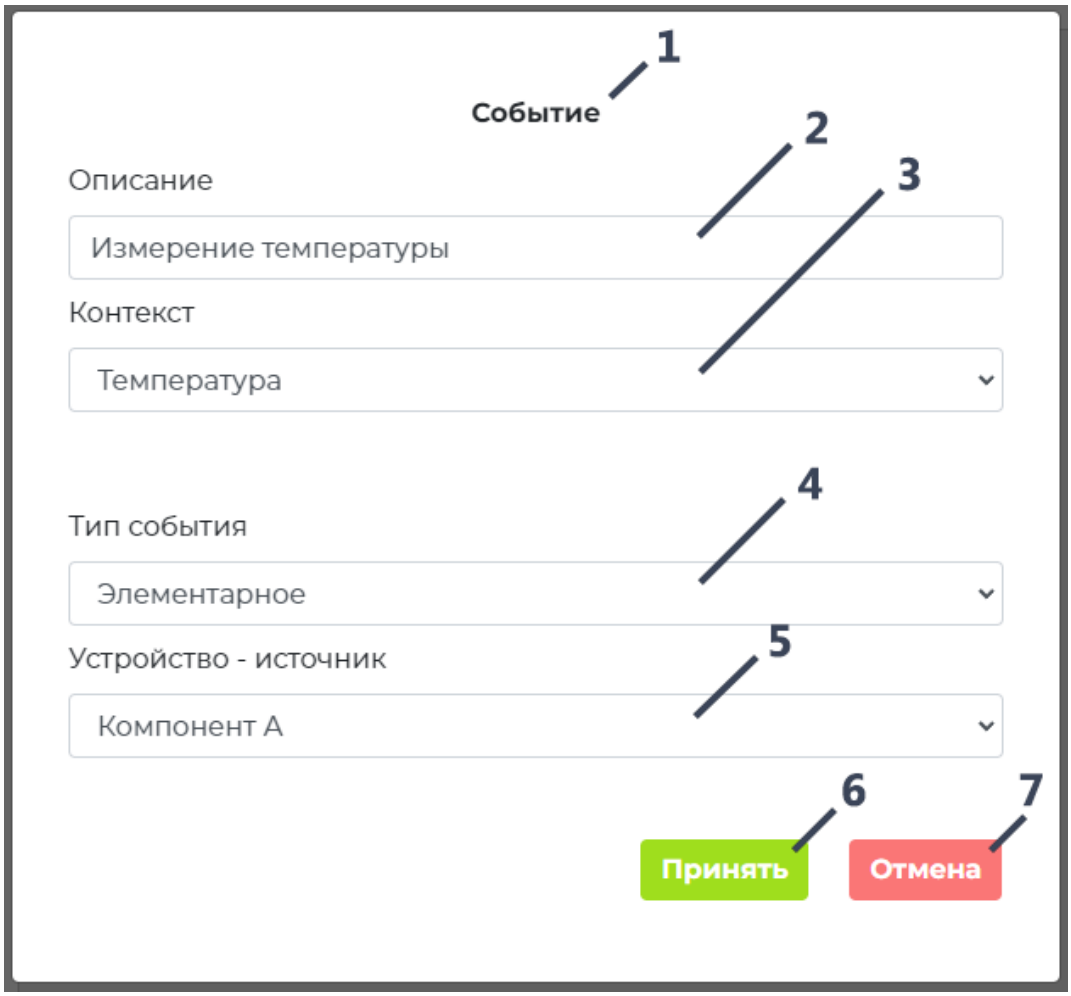


Рисунок 41 – Модальное окно для работы над метаданными объектов

Данное модальное окно состоит из следующих элементов:

1 – наименование типа создаваемого (редактируемого) объекта;

2 – поле описания объекта;

3 – поле выбора контекста объекта;

4 – выбор типа события;

5 – поля ввода данных для определенного типа событий (в данном случае выбор устройства, которое будет испускать данное событие);

6 – кнопка принять изменения;

7 – кнопка отмены изменений и закрытия модального окна.

Для работы системного аналитика была спроектирована страница с соответствующей рабочей областью. Данная страница показана на рисунке 42.

Рисунок 42 – Рабочая область системного аналитика

Составляющие страницы рабочей области системного аналитика:

1 – панель инструментов, позволяющая вызывать функции добавления, изменения и удаления систем;

2 – список систем в виде дерева;

3 – кнопка настройки системы;

4 – кнопка удаления системы.

Кроме того, на каждой странице, исключая окно регистрации, располагается шапка сайта, на которой расположен элемент пользовательской ссылки. Данный элемент представлен на рисунке 43.

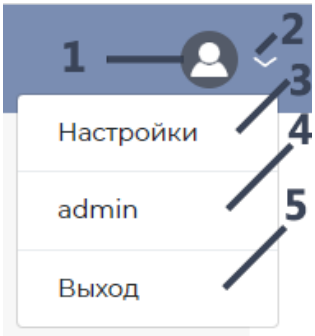


Рисунок 43 – Элемент пользовательской ссылки

Данный элемент состоит из следующих частей:

1 – аватар пользователя;

2 – кнопка раскрытия всплывающего меню;

3 – кнопка перехода в настройки;

4 – кнопки для перехода к соответствующим рабочим областям;

5 – кнопка выхода из системы.

При нажатии на аватар пользователя или на соответствующую кнопку в выпадающем меню, открывается страница пользовательских настроек, которая показана на рисунке 44.

Рисунок 44 – Страница пользовательских настроек

Рассматриваемая страница включает в себя следующие элементы:

1 – электронная почта пользователя;

2 – аватар пользователя;

3 – кнопка смены аватара;

4 – поле старого пароля для его смены;

5 – поле нового пароля;

6 – кнопка смены пароля пользователя.

Все рассмотренные страницы были разработаны средствами языка гипертекстовой разметки HTML и языка таблицы стилей CSS. Также активно применялся фреймворк Bootstrap 4 для упрощения разметки и верстки пользовательского интерфейса.

3.3.3 Разработка программных средств уровня представления

С целью объединения визуальных средств отображения и добавления им функциональности в фреймворке Angular используются компоненты. Компонент angular представляет собой обособленную часть функционала со своей логикой, HTML-шаблоном и CSS-стилями [13]. Рассмотрим более подробно структуру построения основных страниц инструментальных средств управления метаданными.

На рисунке 45 изображена диаграмма пакетов страницы авторизации пользователя.

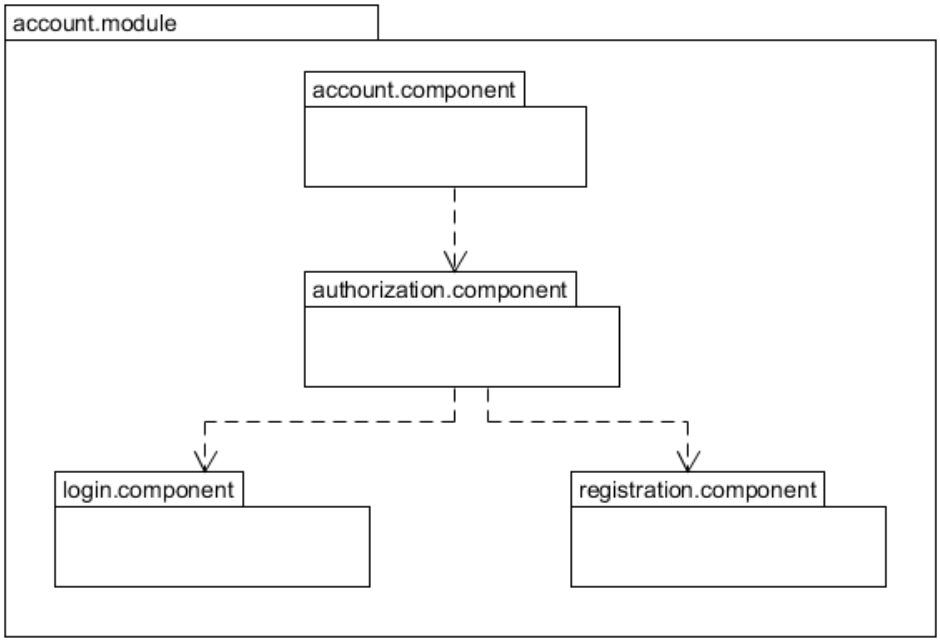


Рисунок 45 – Диаграмма пакетов страницы авторизации

Описание пакетов представлено в таблице 15.

Таблица 15 – Описание пакетов из диаграммы страницы авторизации

№	Наименование	Описание
1	account.component	Angular компонент для отображения страницы авторизации
2	authorization.component	Angular компонент для отображения основного контента и формы

Продолжение таблицы 15

№	Наименование	Описание
3	login.component	Angular компонент для отображения и функционирования формы входа в аккаунт
4	recover.component	Angular компонент для отображения и функционирования формы восстановления пароля

Далее рассмотрим структуру элементов представления страницы администратора. Она отображена в диаграмме пакетов на рисунке 46.

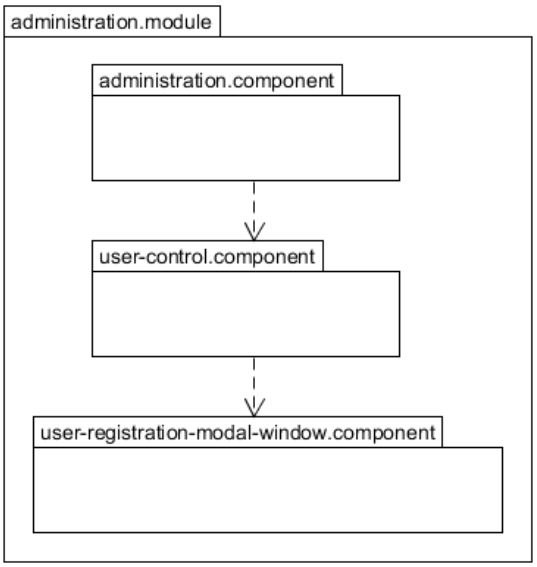


Рисунок 46 – Диаграмма пакетов страницы администратора

Описание пакетов данной диаграммы представлено в таблице 16.

Таблица 16 – Описание пакетов из диаграммы страницы авторизации

№	Наименование	Описание
1	administration.component	Angular компонент для отображения страницы администратора
2	user-control.component	Angular компонент для отображения таблицы пользователей, а также инструментов для работы с ними
3	user-registration-modal-window.component	Angular компонент для отображения модального окна регистрации и изменения пользователя

Следующей структурой, взятой на рассмотрение, является структура элементов представления страницы разработчика. Она отображена в диаграмме пакетов на рисунке 47.

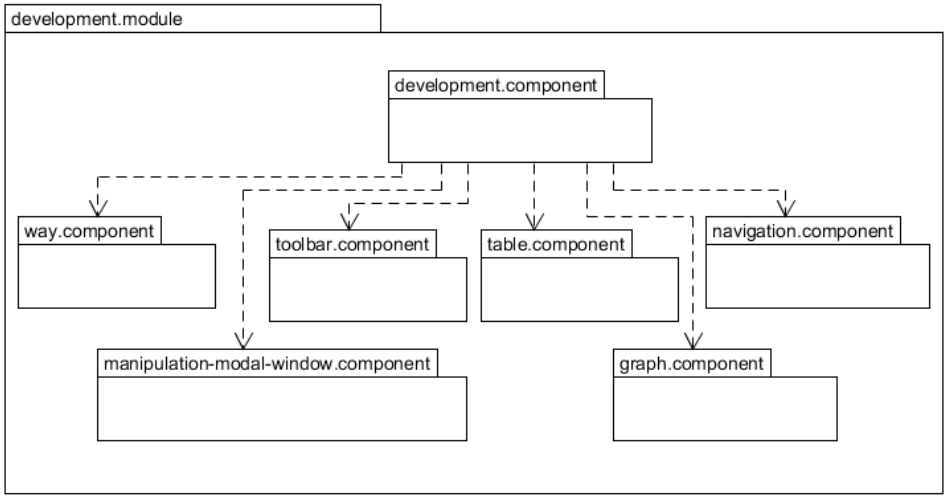


Рисунок 47 – Диаграмма пакетов страницы разработчика

Описание пакетов рассматриваемой диаграммы представлено в таблице 17.

Таблица 17 – Описание пакетов из диаграммы страницы авторизации

№	Наименование	Описание
1	development.component	Angular компонент для отображения страницы разработчика
2	way.component	Angular компонент для отображения строки пути до выбранного объекта системы
3	toolbar.component	Angular компонент для отображения панели инструментов
4	table.component	Angular компонент для отображения таблицы объектов
5	navigation.component	Angular компонент для отображения области навигации
6	graph.component	Angular компонент для отображения графов
7	manipulation-modal-window.component	Angular компонент для отображения модального окна манипулирования объектами системы

Данные Angular компоненты также используют модели для их отображения и изменения.

4 Контроль качества инструментальных программных средств управления метаданными для сервиса потоковой обработки событий

4.1 Планирование процесса контроля качества серверных программных средств

4.1.1 Мероприятия по контролю качества программных средств

Качество программных средств – многогранное понятие, которое может рассматриваться с разных точек зрения. В данное понятие включаются следующие характеристики:

– соответствие характеристикам, определенных в требованиях к программным средствам;

– отсутствие дефектов и ошибок в работе программных средств;

– способность к легкому изменению программного кода и возможности дальнейшего расширения программных средств [14].

Первый пункт определяет, в какой степени программное средство решает задачи конечных пользователей. Несоответствие данному условию нивелируется при грамотной постановке требований к программному продукту, а также правильно проведенном проектировании.

Последующие пункты списка характеристик качества программных средств напрямую зависят от программного кода программы. При его неверном выстраивании возникают как неудобные решения, не позволяющие дополнять и расширять функциональность приложения, так и прямые ошибки в работе программного средства.

Неверные шаги при выстраивании программного кода, которые его слишком укрупняют, тормозят или блокируют его дальнейшее написание, а соответственно и разработку программных средств возможно предупредить заранее при помощи высчитывания метрик программного кода.

Ошибки в работе разрабатываемого программного средства возможно отследить при помощи различных методов тестирования программных продуктов.

4.1.2 Обоснование выбора моделей тестирования

Выделяют несколько видов тестирования программных средств.

Функциональное тестирование – тестирование программного обеспечения в целях проверки его на соответствие функциональным требованиям.

Модульное тестирование – тестирование отдельных модулей программного продукта. Модули приложения проверяются независимо друг от друга как целостный объект в системе.

Интеграционное тестирование – тестирование программного обеспечение, при котором проверяются модули приложения, объединенные в группы, связи между связанными модулями и их взаимодействие. Перед данным видом тестирования проводится модульное тестирование, чтобы исключить ошибки внутри исполняемых модулей.

Тестирование производительности – проверка того, насколько сильно разработанное приложение нагружает ресурсы системы, на котором оно функционирует.

Дымчатое тестирование – тестирование программного обеспечения на ранних этапах разработки. Включает в себя минимальный набор тестов на явные ошибки.

Конфигурационное тестирование – тестирование работы программного обеспечения на различных программных и аппаратных окружениях.

Регрессионное тестирование – тестирование, направленное на обнаружение ошибок в уже протестированных участках программного кода. Применяется на программном средстве, которое будет использоваться на разных платформах, браузерах или использовать разные версии драйверов.

Для разработанного web-приложения инструментальных средств управления метаданными было выбрано несколько видов тестирования, которые включают в себя:

– модульное тестирование для нахождения ошибок внутри отдельных компонентов программы при их написании;

– интеграционное тестирование, направленное на проверку взаимосвязей разработанных компонентов;

– конфигурационное тестирование в целях проверки работы системы в разных web-браузерах.

Тестирование производительности не было проведено в связи с малыми системными и аппаратными требованиями к системе от web-браузеров, которые отображают web-сервис управления метаданными. Дымчатое тестирование не производилось, так как модульное тестирование покрывало все требования к тестированию отдельных компонент на этапе их разработки.

4.1.3 Обоснование выбора метрик программного кода

Существует множество метрик, которые позволяют оценить программный код с разных сторон, при этом затрачивая разное количество усилий и времени и используя разные подходы в подсчёте результатов [15].

Однако были взяты две основные метрики, позволяющие оценить сложность программного кода, а также правильность построения структуры программы.

Первой характеристикой программного кода были выбраны метрики Холстеда, рассчитывающие показатели программы, основываясь на операторах и операндах программы. С их помощью возможно быстро определить объем разработанной программы, а также возможность ее расширения.

Второй метрикой программного кода решено было применить метрики абстрактности и устойчивости. Данные метрики, а также производные от них,

позволяют определить роли модулей, из которых состоит структура программного средства и выявить необходимые и бесполезные модули.

4.1.4 Обоснование выбора методов оценки и метрик качества пользовательского интерфейса

Для оценки качества пользовательского интерфейса следует рассматривать влияние выстроенного юзабилити на пользовательский опыт взаимодействия с интерфейсом.

Для описания и оценки данного влияния используют следующие юзабилити-оценки:

– окончательная оценка, которая показывает, выполнена задача в конечном итоге или нет;

– метод встречающихся проблем, при котором отмечаются основные ошибки пользователей, после чего они ранжируются по частоте встречи с данным затруднением;

– метод времени на задачу, в котором высчитывается время выполнения задачи или комплекса задач для оценивания эффективности и производительности пользователя при использовании данного интерфейса;

– уровень удовлетворения тестом, при котором по завершении теста участникам требуется пройти анкетирование о простоте использования тестируемого сервиса;

– конверсия, которая отображает количество пользователей, совершивших отслеживаемое действие, и применяется в основном на массовых сайтах [16].

Для тестирования разработанного пользовательского интерфейса были проведены тесты времени на задачу, так как данный метод позволяет оценить эффективность выстроенного интерфейса (в отличие от метода окончательной оценки), а также не требует большого количества испытаний, как в методе встречающихся проблем, при подсчете конверсии или уровне удовлетворения тестом.

4.2 Тестирование серверных программных средств

4.2.1 Модульное тестирование

В результате разработки программы было проведено модульное тестирование. Модульные тесты были разработаны с помощью встроенного в фреймворк Angular инструмента Jasmine [17]. Рассмотрим тестирование страницы авторизации пользователя. Исходный код модульных тестов приведен в приложении Е. План тестирования представлен в таблице 18.

Таблица 18 – План модульного тестирования

Тестируемый класс	Тест	Ожидаемый результат	Актуальный результат
Authorization Component	Должен быть создан	Пройден	Пройден (рисунок 48)
	Должен отобразить форму	Пройден	Пройден (рисунок 48)
	Должен переключить вкладку входа на вкладку регистрации	Пройден	Пройден (рисунок 48)
Login Component	Должен быть создан	Пройден	Пройден (рисунок 48)
	Должен считать значения с полей формы	Пройден	Пройден (рисунок 48)
	Должен записать в localStorage полученный accessToken при верно введенных значениях	Пройден	Пройден (рисунок 48)
	Должен выдать ошибку при вводе неверных данных	Пройден	Пройден (рисунок 48)
Registration Component	Должен быть создан	Пройден	Пройден (рисунок 48)
	Должен считать значения с полей формы	Пройден	Пройден (рисунок 48)
	Должен записать в localStorage полученный accessToken при верно введенных значениях	Пройден	Пройден (рисунок 48)
	Должен выдать ошибку при вводе неверных данных	Пройден	Пройден (рисунок 48)

Результаты модульного тестирования представлены на рисунке 48.

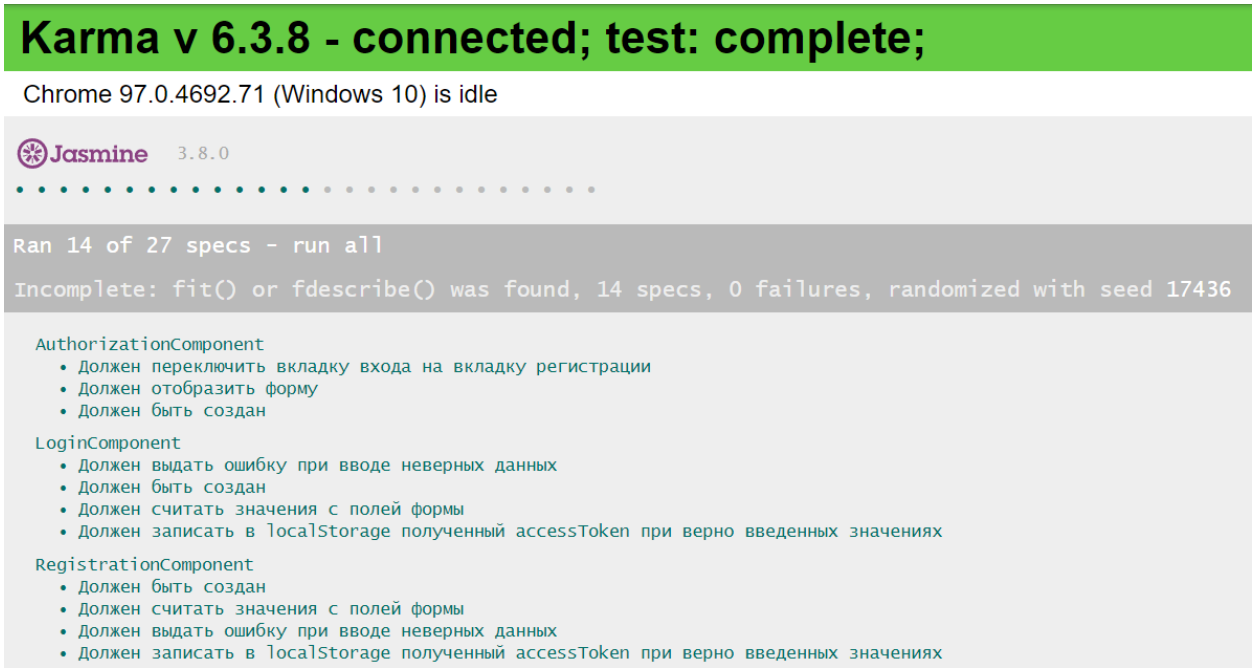
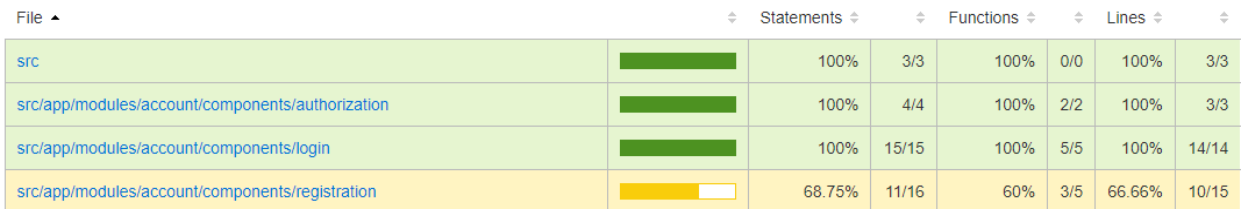


Рисунок 48 – Результаты модульного тестирования

Статистика по покрытию кода приложения модульными тестами представлена на рисунке 49.



File	Statements		Functions		Lines	
src	100%	3/3	100%	0/0	100%	3/3
src/app/modules/account/components/authorization	100%	4/4	100%	2/2	100%	3/3
src/app/modules/account/components/login	100%	15/15	100%	5/5	100%	14/14
src/app/modules/account/components/registration	68.75%	11/16	60%	3/5	66.66%	10/15

Рисунок 49 – Статистика покрытия программного кода модульными тестами

Программа успешно прошла все этапы модульного тестирования.

4.2.2 Интеграционное тестирование

Следующим этапом после модульных тестов было проведено интеграционное тестирование. Оно же предназначено для более комплексной проверки системы на наличие ошибок, когда отдельно проверенные программные модули объединяются в более крупные множества и создают связи для передачи сообщений друг другу. Целью данного вида тестирования является выявление ошибок при взаимодействии между программными

модулями системы и в первую очередь направлен на проверку обмена данными между этими самими модулями.

Интеграционное тестирование было разработано с помощью того же инструментария, что и unit-тесты – фреймворка Jasmine. Но в данных тестах были подключены внешние зависимости, а именно компонент для посылки и приема HTTP-запросов на сервер и компонент маршрутизации. Исходный код интеграционных тестов представлен в приложении Ж. План и результаты проведенного интеграционного тестирования представлен в таблице 19.

Таблица 19 – План интеграционного тестирования

Тестируемый класс	Тест	Ожидаемый результат	Актуальный результат
Login Component	Должен обратиться к серверу и записать в localStorage полученный accessToken при верно введенных значениях	Пройден	Пройден (рисунок 50)
	Должен перейти на страницу /workspace при получении accessToken	Пройден	Пройден (рисунок 50)
	Должен обратиться к серверу и выдать ошибку при вводе неверных данных	Пройден	Пройден (рисунок 50)
Registration Component	Должен обратиться к серверу и записать в localStorage полученный accessToken при верно введенных значениях	Пройден	Пройден (рисунок 50)
	Должен перейти на страницу /workspace при получении accessToken	Пройден	Пройден (рисунок 50)
	Должен обратиться к серверу и выдать ошибку при вводе неверных данных	Пройден	Пройден (рисунок 50)

Результаты проведения интеграционного тестирования представлены на рисунке 50.

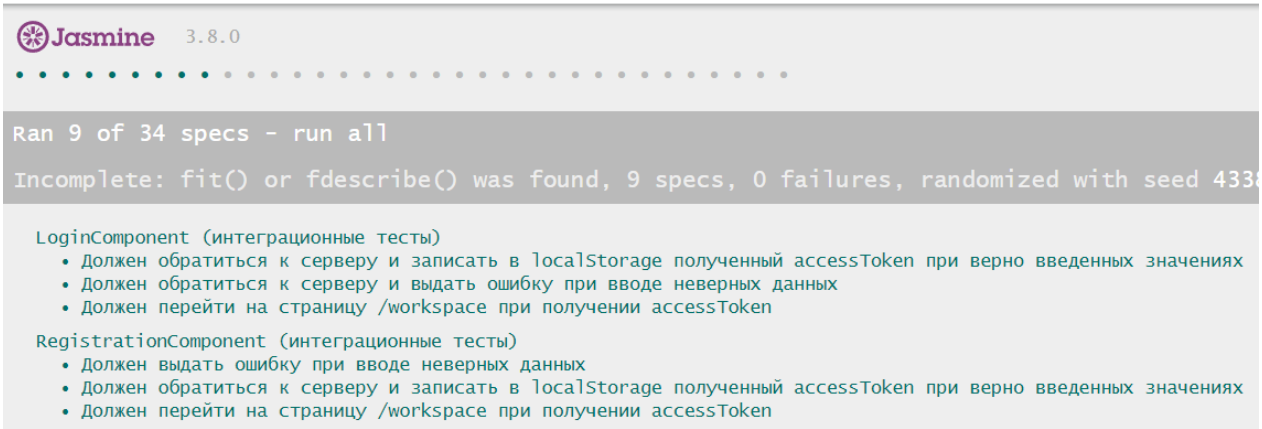


Рисунок 50 – Результаты модульного тестирования

Программные средства успешно прошли интеграционное тестирование.

4.2.3 Конфигурационное тестирование

Также было проведено конфигурационное тестирование разрабатываемого web-приложения. В общем случае конфигурационное тестирование – это проверка работы программного обеспечения на различных программных и аппаратных окружениях. Тестирование проектируемой программы было проведено с несколькими версиями известных браузеров для проверки ее работоспособности в разных исполняемых окружениях.

Данный вид тестирования производился путем запуска web-приложения на разных браузерах с помощью виртуальной машины TURBO.NET. План конфигурационного тестирования указан в таблице 20. Результаты конфигурационного тестирования расположены в приложении И.

Таблица 20 – План конфигурационного тестирования

Тестируемый браузер	Тест	Ожидаемый результат	Актуальный результат
Google Chrome (версия 99.0)	Отображение страницы входа в аккаунт	Пройден	Пройден (рисунок И.1)
	Получение ошибки при вводе неверных данных	Пройден	Пройден (рисунок И.2)
	Переадресация при вводе верных данных	Пройден	Пройден (рисунок И.3)

Продолжение таблицы 20

Тестируемый браузер	Тест	Ожидаемый результат	Актуальный результат
Microsoft Edge (версия 99.0)	Отображение страницы входа в аккаунт	Пройден	Пройден (рисунок И.4)
	Получение ошибки при вводе неверных данных	Пройден	Пройден (рисунок И.5)
	Переадресация при вводе верных данных	Пройден	Пройден (рисунок И.6)
Firefox Browser (версия 72.0.2)	Отображение страницы входа в аккаунт	Пройден	Пройден (рисунок И.7)
	Получение ошибки при вводе неверных данных	Пройден	Пройден (рисунок И.8)
	Переадресация при вводе верных данных	Пройден	Пройден (рисунок И.9)
Яндекс.Браузер (версия 22.3.0)	Отображение страницы входа в аккаунт	Пройден	Пройден (рисунок И.10)
	Получение ошибки при вводе неверных данных	Пройден	Пройден (рисунок И.11)
	Переадресация при вводе верных данных	Пройден	Пройден (рисунок И.12)
Opera (версия 85.0)	Отображение страницы входа в аккаунт	Пройден	Пройден (рисунок И.13)
	Получение ошибки при вводе неверных данных	Пройден	Пройден (рисунок И.14)
	Переадресация при вводе верных данных	Пройден	Пройден (рисунок И.15)

Инструментальные средства управления метаданными были успешно запущены в разных web-браузерах.

4.3 Оценка качества программного кода

4.3.1 Метрики Холстеда

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

Для расчёта метрик Холстеда был взят класс, выполняющий HTTP-запросы к серверу. Код данного модуля представлен в листинге 2.

Листинг 2 – Код сервиса запросов

```
@Injectable({providedIn: 'root'})  //Сервис общий для проекта
export class HTTPRequestService
{
  //Импорт объекта сервиса для отправки http запросов
  constructor(private httpClient: HttpClient) {}
  /**
   * Функция запроса GET к серверу
   * @param url URL-адрес, к которому направляется запрос
   * @param additionalHeaders Дополнительные заголовки запроса
   * @returns Объект типа Observable, при подписке возвращает информацию от
сервера
  */
  public get(url: string, additionalHeaders?: HttpHeaders): Observable<any>
  {
    let headers = new HttpHeaders
    (
      {
        'Content-Type': 'application/json',
        'Accept': '*/*',
        // 'Host': 'PostmanRuntime/7.29.0',
        // 'Accept-Encoding': 'gzip, deflate, br',
        // 'Connection': 'keep-alive'
        // 'Access-Control-Allow-Origin': 'http://localhost:4200',
        // "Access-Control-Allow-Methods": "GET,PUT,POST,DELETE,PATCH,OPTIONS"
      }
    );

        //Добавление дополнительный заголовков к запросу
    if(additionalHeaders)
    {
      additionalHeaders.keys().forEach(key =>
      {
        let value = additionalHeaders.get(key);

        if(headers.has(key))
          headers = headers.set(key, value ? value : "");
        else
          headers = headers.append(key, value ? value : "");
      });
    }

    //Объект опций для ответа на запрос
    let options: Object =
    {
      headers: headers,
      observe: 'body',
      withCredentials: true
    };
    return this.httpClient.get<any>(url, options);
  }

  /**
   * Функция запроса POST к серверу
   * @param url URL-адрес, к которому направляется запрос
   * @param additionalHeaders Дополнительные заголовки запроса
   * @returns Объект типа Observable, при подписке возвращает информацию от
сервера
  */
  public post(url: string, object: any, additionalHeaders?: HttpHeaders):
Observable<any>
  {
```

```
    let headers = new HttpHeaders
    (
      {
        'Content-Type': 'application/json',
        'Accept': '*/*',
        // 'Host': 'PostmanRuntime/7.29.0',
        // 'Accept-Encoding': 'gzip, deflate, br',
        // 'Connection': 'keep-alive'
        // 'Access-Control-Allow-Origin': 'http://127.0.0.1:3000',
        // 'Access-Control-Allow-Headers': 'Content-Type, Authorization'
        // "Access-Control-Allow-Methods": "GET,PUT,POST,DELETE,PATCH,OPTIONS"
      }
    );

      //Добавление дополнительный заголовков к запросу
  if(additionalHeaders)
  {
    additionalHeaders.keys().forEach(key =>
    {
      let value = additionalHeaders.get(key);

      if(headers.has(key))
        headers = headers.set(key, value ? value : "");
      else
        headers = headers.append(key, value ? value : "");
    });
  }

      //Объект опций для ответа на запрос
    let options: Object =
    {
      headers: headers,
      observe: 'body',
      withCredentials: true
    };

    return this.httpClient.post<any>(url, object, options);
  }

  /**
   * Функция запроса PATCH к серверу
   * @param url URL-адрес, к которому направляется запрос
   * @param additionalHeaders Дополнительные заголовки запроса
   * @returns Объект типа Observable, при подписке возвращает информацию от
сервера
   */
  public patch(url: string, object: any, additionalHeaders?: HttpHeaders):
Observable<any>
  {
    let headers = new HttpHeaders
    (
      {
        'Content-Type': 'application/json',
        'Accept': '*/*',
      }
    );

  //Добавление дополнительный заголовков к запросу
  if(additionalHeaders)
  {
    additionalHeaders.keys().forEach(key =>
    {
      let value = additionalHeaders.get(key);
```

```
      if(headers.has(key))
        headers = headers.set(key, value ? value : "");
      else
        headers = headers.append(key, value ? value : "");
    });
  }

  //Объект опций для ответа на запрос
    let options: Object =
    {
      headers: headers,
      observe: 'body',
      withCredentials: true
    };

    return this.httpClient.patch<any>(url, object, options);
  }

  /**
   * Функция запроса PATCH к серверу
   * @param url URL-адрес, к которому направляется запрос
   * @param additionalHeaders Дополнительные заголовки запроса
   * @returns Объект типа Observable, при подписке возвращает информацию от
сервера
  */
   public delete(url: string, object: any, additionalHeaders?: HttpHeaders):
Observable<any>
   {
     let headers = new HttpHeaders
     (
       {
         'Content-Type': 'application/json',
         'Accept': '*/*',
       }
     );

   //Добавление дополнительный заголовков к запросу
   if(additionalHeaders)
   {
     additionalHeaders.keys().forEach(key =>
     {
       let value = additionalHeaders.get(key);

       if(headers.has(key))
         headers = headers.set(key, value ? value : "");
       else
         headers = headers.append(key, value ? value : "");
     });
   }

   //Объект опций для ответа на запрос
     let options: Object =
     {
       headers: headers,
       observe: 'body',
       withCredentials: true,
       body: object
     };

     return this.httpClient.delete<any>(url, options);
   }


  public GetAuthorizationHeader(): HttpHeaders
```

```
  {
    return new HttpHeaders
    (
      {
        'Authorization': 'Bearer ' + localStorage.getItem("accessToken")
      }
    );
  }
}
```

Количество используемых операндов и операторов, а также их общее число в программе приведены в таблицах 21 и 22 соответственно.

Таблица 21 – Операторы

Операторы	Число вхождений
:	51
?	12
< >	8
=	20
( )	54
.	38
=>	4
{ }	23
,	38
;	34
‘ ’	22
this	4
additionalHeaders.get	4
headers.set	4
httpClient.get	1
httpClient.post	1
httpClient.patch	1
httpClient.delete	1
localStorage.getItem	1
return	5
n1 = 20	N1 = 326

Таблица 22 – Операнды

Операнды	Число вхождений
httpClient	5
url	8
additionalHeaders	16
headers	37
'Content-Type'	4
'application/json'	4
key	20
value	20
"""	8
options	8
'body'	4
object	6
n2 = 12	N2 = 140

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 326 + 140 = 466.$$

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 20 + 12 = 32.$$

Объем программы рассчитывается по формуле:

$$H_V = N * log_2(n) = 466 * 5 = 2330.$$

Потенциальный объем программы

$$V* = (2 + n`) * log2(2 + n`) = 6 * log2(6) = 15{,}51.$$

Теоретический уровень программы рассчитывается по формуле:

$$L` = V* / H_V = 15{,}51 / 2330 = 0.0067.$$

Сложность программы рассчитывается по формуле:

$$HD = (n1 / 2) * (N2 / n2) = (20 / 2) * (140 / 12) = 116{,}67.$$

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

4.3.2 Метрики устойчивости и абстрактности

Для разработанного web-приложения были рассчитаны соотношения между абстрактностью и устойчивостью. Вычисление рассматриваемых метрик производилось исходя из пакетов, в которые были организованы программные классы приложения. Всего имеется 8 основных модулей и один встроенный от фреймворка: App, Shared, Core, Development, Account, Administration, System-analytics, Options.

Для расчёта устойчивости компонентов используется следующая формула:

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

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

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

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

Была рассчитана величина устойчивости для каждого компонента системы:

- $I_1 = \frac{7}{7+0} = 1;$
- $I_2 = \frac{0}{6+0} = 0;$
- $I_3 = \frac{0}{1+0} = 0;$
- $I_4 = \frac{2}{1+2} = 0{,}67;$
- $I_5 = \frac{1}{1+1} = 0{,}5;$

– $I_6 = \frac{2}{1+2} = 0{,}67;$

– $I_7 = \frac{2}{1+2} = 0{,}67;$

– $I_8 = \frac{2}{1+2} = 0{,}67.$

Для расчета абстрактности компонентов использовалась формула, приведенная ниже:

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

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

Nc – количество классов в компоненте;

Na – количество абстрактных классов в компоненте.

Данная метрика также была рассчитана для каждого компонента сервиса:

– $A_1 = \frac{1}{2} = 0{,}5;$

– $A_2 = \frac{1}{6} = 0{,}167;$

– $A_3 = \frac{2}{5} = 0{,}4;$

– $A_4 = \frac{2}{20} = 0{,}1;$

– $A_5 = \frac{1}{6} = 0{,}167;$

– $A_6 = \frac{1}{8} = 0{,}125;$

– $A_7 = \frac{1}{4} = 0{,}25;$

– $A_8 = \frac{1}{3} = 0{,}3.$

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

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

где A – абстрактность;

I – устойчивость.

Был проведен расчёт данной метрики для каждого компонента.

– $D'_1 = |1 + 0{,}5 - 1| = 0{,}5$;
– $D'_2 = |0 + 0{,}167 - 1| = 0{,}83$;
– $D'_3 = |0 + 0{,}4 - 1| = 0{,}6$;
– $D'_4 = |0{,}67 + 0{,}1 - 1| = 0{,}23$;
– $D'_5 = |0{,}5 + 0{,}167 - 1| = 0{,}3$;
– $D'_6 = |0{,}67 + 0{,}125 - 1| = 0{,}205$;
– $D'_7 = |0{,}67 + 0{,}25 - 1| = 0{,}08$;
– $D'_8 = |0{,}67 + 0{,}3 - 1| = 0{,}03$.

Данный показатель возможно представить при помощи диаграммы, изображенной на рисунке 51. В данной координатной сетке располагаются зона неприятностей (где компоненты слишком устойчивы и не абстрактны), зона бесполезности (компоненты в данной области в большой степени неустойчивы и абстрактны), а также главная последовательность, к которой должны стремиться компоненты по рассматриваемым показателям.

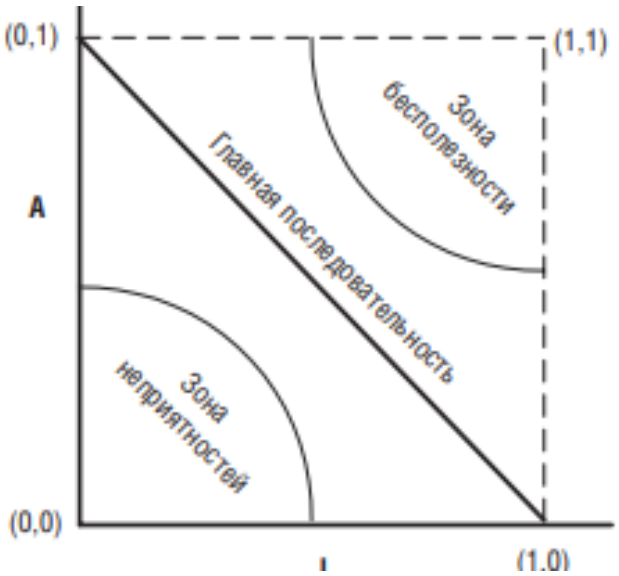


Рисунок 51 – Диаграмма возможных значений главной последовательности

По рассчитанным значениям для компонентов приложения был построен аналогичный график, изображенный на рисунке 52.

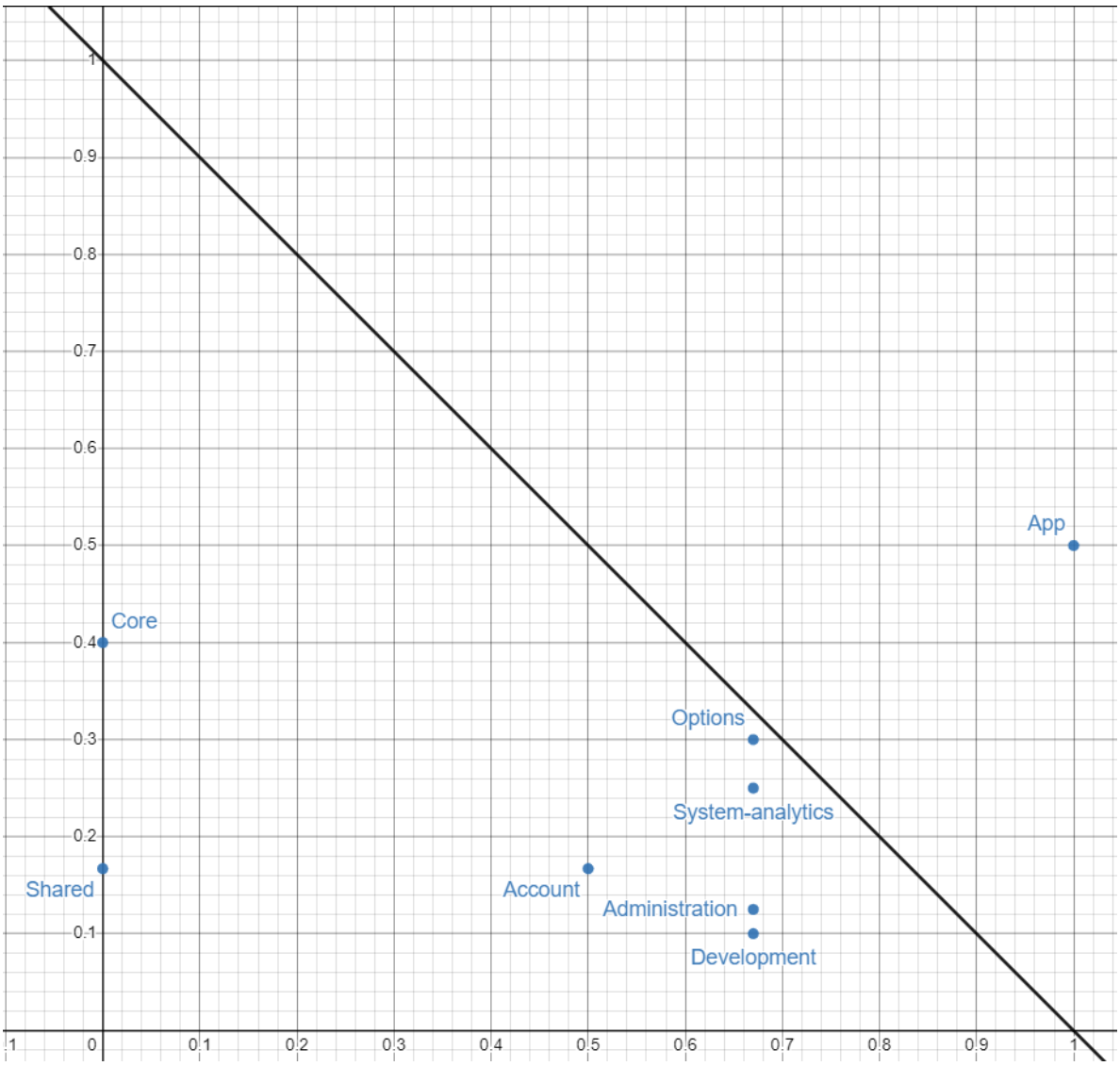


Рисунок 52 – Метрики кода на графе значений главной последовательности

Компоненты Core и Shared располагаются далеко от главной последовательности и являются слишком устойчивыми. Это объясняется тем, что данные компоненты реализуют базовые функции сервиса и не зависят от других блоков приложения.

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

4.4 Оценка качества пользовательского интерфейса

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

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

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

- адрес электронной почты – hopsererze@vusra.com;
- пароль – XFxMeGnrH7;
- роль – разработчик.

В таблице 23 приведен список составляющих действий, а также длительность каждой проведенной операции.

Таблица 23 – План тестирования эффективности пользовательского интерфейса модального кона добавления пользователя

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

Продолжение таблицы 23

№	Тип действия	Продолжительность	Комментарий
5	П	1,1	Перемещение курсора к полю ввода email адреса
6	М	0,1	Нажатие на кнопку мыши. Перемещение фокуса на поле email адреса
7	Р	0,4	Перенос руки с мыши на клавиатуру
8	К х 21	0,28х21	Ввод электронной почты из 20 символов и нажатия клавиши shift для ввода @
9	Р	0,4	Перенос руки с клавиатуры на мышь
10	П	1,1	Перемещение курсора к полю ввода пароля
11	М	0,1	Нажатие кнопки мыши. Перемещение фокуса на поле ввода пароля
12	Р	0,4	Перенос руки с мыши на клавиатуру
13	К х 15	0,28х15	Ввод пароля из 10 символов и нажатие 5 раз клавиши shift для ввода букв в высоком регистре
14	Р	0,4	Перенос руки с клавиатуры на мышь
15	П	1,1	Перемещение курсора к кнопке «Добавить роль»
16	М	0,1	Нажатие кнопки мыши. Добавление ячейки роли с содержимым «Администратор» по-умолчанию
17	П	1,1	Перемещение курсора к слоту добавленной роли
18	М	0,1	Нажатие кнопки мыши. Раскрытие списка доступных ролей
19	П	1,1	Перемещение курсора к нужной роли
20	М	0,1	Нажатие кнопки мыши. Ввод роли произошел
21	П	1,1	Перемещение курсора к кнопке «Принять»
22	М	0,1	Нажатие кнопки мыши
23	В	0,3	Время реакции системы. Отправка данных на сервер и получение ответа. Последующее закрытие диалогового окна

В итоге, для реализации функции добавления нового пользователя в систему администратор произвел 23 действия, затратив на данные операции суммарно 21,68 секунд. Объем затраченного времени на столь комплексную

задачу невелик, что показывает хороший уровень построения юзабилити в разрабатываемых инструментальных средствах.

Заключение

Во время выполнения выпускной квалификационной работы были разработаны инструментальные средства управления метаданными для сервиса потоковой обработки событий.

Был подробным образом изучен сервис потоковой обработки событий и исполнения активных правил, для которого разрабатывались средства управления. Были рассмотрены назначение и основные функциональные возможности системы, ее микросервисная событийно-ориентированная архитектура и порядок работы. Кроме того, были описаны состав, жизненный цикл метаданных в системе, а также процессы управления ими. После анализа аналогичных систем и определения цели на разработку были описаны основные автоматизируемые функции инструментальных средств управления метаданными. После было произведено планирование разработки, а также расчет временных и ресурсных затрат на создание программных средств. Затраты на проект составили 233680 рублей. Время разработки 71 рабочий день.

На следующем этапе были определены функциональные требования, требования к пользовательскому интерфейсу и архитектуре инструментальных средств. Были выбраны основные языки программирования JavaScript и TypeScript для реализации программных средств, а также другие технологии разработки, интегрированные в рабочий процесс. Помимо этого, были описаны требования к аппаратно-программному обеспечению пользователя для работы с инструментальынми средствами управления метаданными.

Разработка программных средств проводилась в среде разработки Microsoft Visual Studio Code. Инструментальное средство создавалось в виде web-приложения, что позволяет запускать его в любом современном браузере.

Разработанное программное средство было протестировано при помощи модульного, интеграционного и конфигурационного тестирования. Все тесты

приложение прошло успешно. Также были вычислены метрики программного кода и юзабилити выстроенного пользовательского интерфейса, которые показали удовлетворительный уровень данных компонент разработки.

Список использованных источников

1. Как событийно-ориентированная архитектура решает проблемы современных веб-приложений [Электронный ресурс]. URL: https://habr.com/ru/company/piter/blog/530514/ (дата обращения: 17.04.22).

2. Ричардсон К. Микросервисы / К. Ричардсон // Паттерны разработки и рефакторинга. − СПб. : Изд-во Питер, 2019. – 544 с.

3. Беджек Б. Kafka Streams в действии / Б. Беджек // Приложения и микросервисы для работы в реальном времени. − СПб. : Изд-во Питер, 2019. – 304 с.

4. Getting started with AWS IoT Core [Электронный ресурс]. URL: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html (дата обращения 20.04.2022).

5. Интернет вещей Azure [Электронный ресурс]. URL: https://azure.microsoft.com/ru-ru/overview/iot/ (дата обращения 22.04.2022).

6. Cisco IoT Field Network Director Configuration Guide [Электронный ресурс]. URL: https://www.cisco.com/c/en/us/td/docs/routers/connectedgrid/iot_fnd/guide/4_3/iot_fnd_ug4_3.pdf (дата обращения 23.04.2022).

7. И. А. Перл, О. В. Калёнова введение в методологию программной инженерии / И. А. Перл, О. В. Калёнова – СПб. : Изд-во Университет ИТМО, 2019. – 53 с.

8. Использование MS Project для управления проектами по разработке ПО [Электронный ресурс]. URL: https://habr.com/ru/post/151593/ (дата обращения: 29.04.22).

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

10. Паттерны для новичков: MVC vs MVP vs MVVM [Электронный ресурс]. URL: https://habr.com/ru/post/215605/ (дата обращения: 12.05.22).

11. Почему HTML по-прежнему востребован? [Электронный ресурс]. URL: https://apix-drive.com/ru/blog/useful/chto-takoe-html (дата обращения: 17.05.22).

12. Головач В. В. Дизайн пользовательского интерфейса / Головач В. В. // Первоначальное проектирование. − СПб. : Изд-во Usethics, 2000. – 146 с.

13. Компоненты Angular [Электронный ресурс]. URL: https://angdev.ru/doc/angular-components/ (дата обращения: 25.05.22).

14. Звездин С. В. Проблемы измерения качества программного кода / С. В. Звездин // Качество программного обеспечения и качество кода : серия «Компьютерные технологии, управление, радиоэлектроника», выпуск 11 – Челябинск. : Изд-во Вестник ЮУрГУ, 2010. – С. 2 – 4.

15. Программный код и его метрики [Электронный ресурс]. URL: https://habr.com/ru/company/intel/blog/106082/ (дата обращения: 02.06.22).

16. 10 основных юзабилити-метрик [Электронный ресурс]. URL: https://habr.com/ru/sandbox/38589/ (дата обращения: 06.06.22).

17. 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: 11.06.22).

Приложение А

ГЛОССАРИЙ

(справочное)

JWT (JSON Web Token) – стандарт для создания данных для доступа пользователя к сайту, основанный на формате JSON.

SDK – набор средств разработки, позволяющий специалистам по программному обеспечению создавать приложения для определенного программного обеспечения.

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

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

Юзабилити – способность пользовательского интерфейса быть понимаемым, изучаемым, используемым и привлекательным для пользователя.

Приложение Б

ГРАФИКИ ТРУДОВЫХ РЕСУРСОВ

(рекомендуемое)

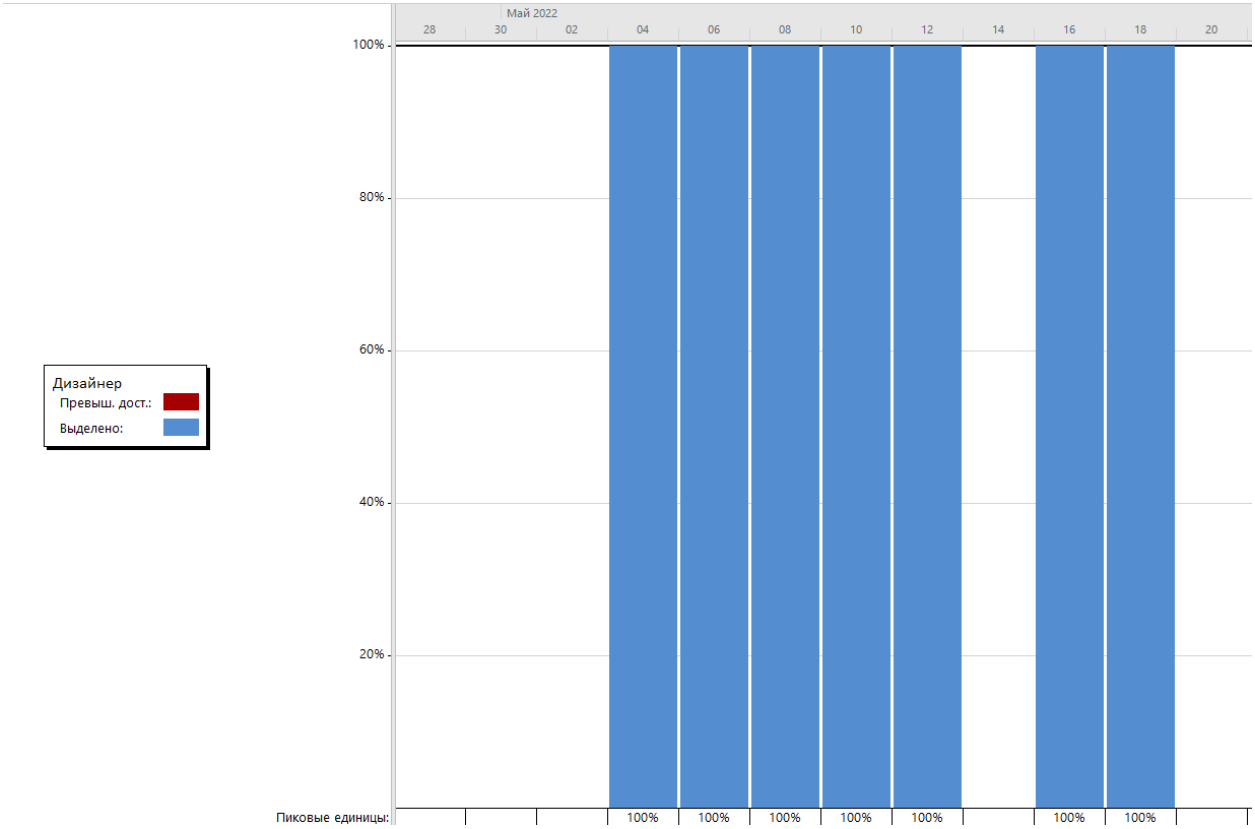


Рисунок Б.1 – График ресурсов для ресурса «Дизайнер»

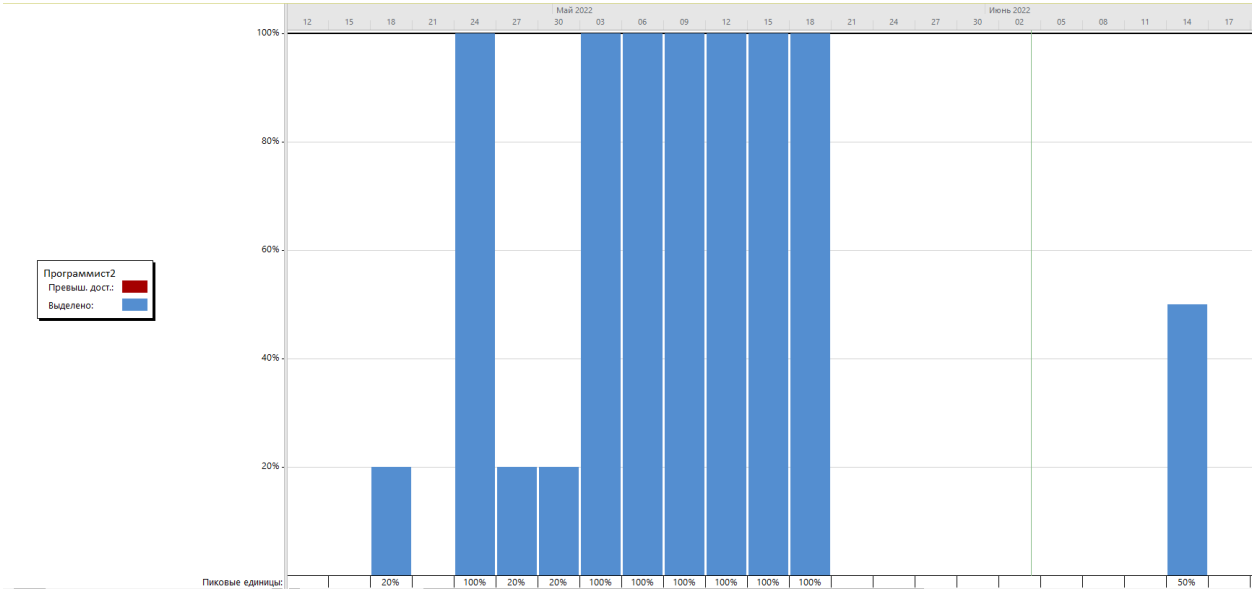


Рисунок Б.2 – График ресурсов для ресурса «Программист2»

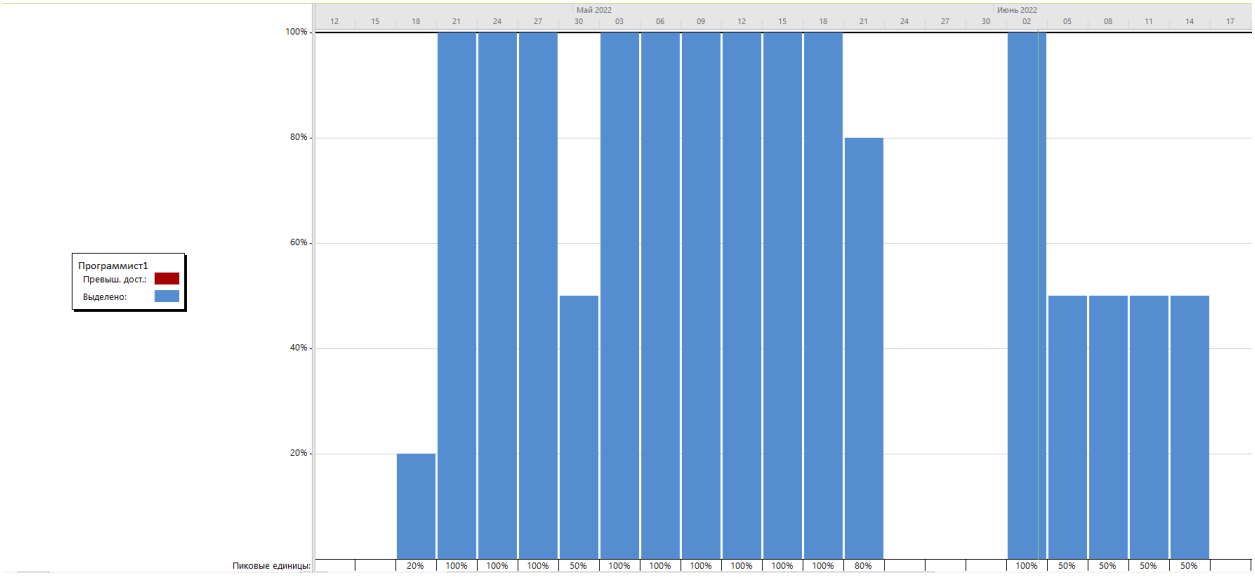


Рисунок Б.3 – График ресурсов для ресурса «Программист1»

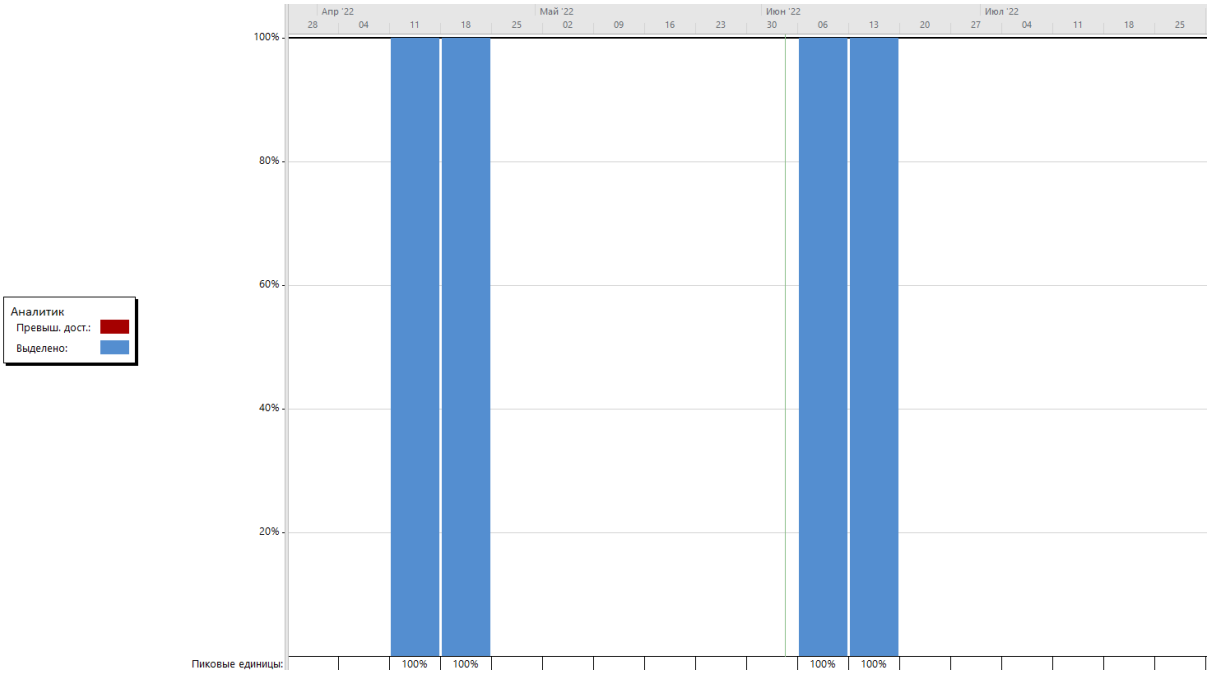


Рисунок Б.4 – График ресурсов для ресурса «Аналитик»

Приложение В

ЛИСТИНГ МОДЕЛЕЙ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ

(рекомендуемое)

Листинг В.1 – Файл admin-list.model.ts

```
import { Role } from "./role.model";
import { User } from "./user.model";

export class AdminList
{
    private _userList: User[] = [];
    private _roleList: Role[] = [];

    public get UserList(): User[]
    {
        return this._userList;
    }
    private set UserList(value: User[])
    {
        if(value)
            this._userList = value;
    }

    public get RoleList(): Role[]
    {
        return this._roleList;
    }
    private set RoleList(value: Role[])
    {
        if(value)
            this._roleList = value;
    }

    constructor(usersList?: any, roleList?: any)
    {
        if(usersList)
        {
            for(let i = 0; i < usersList.length; i++)
            {
                let user = new User(usersList[i].code, usersList[i].email);
                let index = this.AddUser(user);

                for(let j = 0; j < usersList[i].role.length; j++)
                {
                    let role = new Role(usersList[i].role[j].name,
usersList[i].role[j].description);
                    this.UserList[i].AddRole(role);
                }
            }
        }

        if(roleList)
        {
            for(let i = 0; i < roleList.length; i++)
            {
                let role = new Role(roleList[i].name, roleList[i].description);
                let index = this.AddRole(role);
            }
        }
    }

    public AddUser(user: User): number
    {
        if(user)
        {
```

```
            return this.UserList.push(user) - 1;
        }
        else
        {
            return -1;
        }
    }

    public AddRole(role: Role): number
    {
        if(role)
        {
            return this.RoleList.push(role) - 1;
        }
        else
        {
            return -1;
        }
    }
}
```

Листинг В.2 – Файл user/role.model.ts

```
export class Role
{
    private _name: string = '';
    private _description: string = "";

    public get Name(): string
    {
        return this._name;
    }
    private set Name(value: string)
    {
        if(value)
            this._name = value;
    }

    public get Description(): string
    {
        return this._description;
    }
    private set Description(value: string)
    {
        if(value)
            this._description = value;
    }

    constructor(name: string, description?: string)
    {
        this.Name = name;

        if(description)
            this.Description = description;
    }
}
```

Листинг В.3 – Файл user.model.ts

```
import { Role } from "./role.model";

export class User
```

```
{
    private _code: string = '00000';
    private _email: string = "";
    private _role: Role[] = [];

    public get Code(): string
    {
        return this._code;
    }
    private set Code(value: string)
    {
        if(value)
            this._code = value;
    }

    public get Email(): string
    {
        return this._email;
    }
    public set Email(value: string)
    {
        if(value)
            this._email = value;
    }

    public get Roles(): Role[]
    {
        return this._role;
    }

    constructor(code: string, email: string)
    {
        this.Code = code;
        this.Email = email;
    }

    public AddRole(role: Role)
    {
        if(role)
        {
            this._role.push(role);
        }
    }

    public RemoveRole(role: Role)
    {
        if(role)
        {
            this._role.splice(this._role.indexOf(role), 1);
        }
    }
}
```

Листинг В.4 – Файл system.model.ts

```
import { Device } from "./device.model";

export class System
{
    private _code: string = "";
    private _name: string = "";

    private _parent: System | null = null;
```

```
    private _systems: System[] = [];
    private _devices: Device[] = [];

    public get Code(): string
    {
        return this._code;
    }
    private set Code(value: string)
    {
        if(value)
            this._code = value;
    }

    public get Name(): string
    {
        return this._name;
    }
    private set Name(value: string)
    {
        if(value)
            this._name = value;
    }

    public get Parent(): System | null
    {
        return this._parent;
    }

    public get Systems(): System[]
    {
        return this._systems;
    }

    constructor(code: string, name?: string, parent?: System)
    {
        this.Code = code;
        if(name)
            this.Name = name;

        this.SetParent(parent);
    }

    public SetParent(parent: System | undefined)
    {
        if(parent)
            this._parent = parent;
    }

    public AddSystem(newSystem: System | undefined)
    {
        if(newSystem && !this._systems.includes(newSystem))
            this._systems.push(newSystem);
    }

    public AddDevice(newDevice: Device | undefined)
    {
        if(newDevice && !this._devices.includes(newDevice))
            this._devices.push(newDevice);
    }
}
```

Листинг В.5 – Файл device/role.model.ts

```
export class Role
{
    private _name: string = "";
    private _description: string = "";

    public get Name(): string
    {
        return this._name;
    }
    private set Name(value: string)
    {
        if(value)
            this._name = value;
    }

    public get Description(): string
    {
        return this._description;
    }
    private set Description(value: string | undefined)
    {
        if(value)
            this._description = value;
    }

    constructor(name: string, description?: string)
    {
        this.Name = name;
        this.Description = description;
    }
}
```

Листинг В.6 – Файл parameter.model.ts

```
export class Parameter
{
    private _code: string = "";
    private _name: string = "";
    private _description: string = "";
    private _dataType: string = "";

    public get Code(): string
    {
        return this._code;
    }
    private set Code(value: string)
    {
        if(value)
            this._code = value;
    }

    public get Name(): string
    {
        return this._name;
    }
    private set Name(value: string)
    {
        if(value)
            this._name = value;
    }
```

```
    public get Description(): string
    {
        return this._description;
    }
    private set Description(value: string | undefined)
    {
        if(value)
            this._description = value;
    }

    public get DataType(): string
    {
        return this._dataType;
    }
    private set DataType(value: string)
    {
        if(value)
            this._dataType = value;
    }

    constructor(code: string, name?: string, description?: string, dataType?:
string)
    {
        this.Code = code;
        if(name)
            this.Name = name;
        if(description)
            this.Description = description;
        if(dataType)
            this.DataType = dataType;
    }
}
```

Листинг В.7 – Файл level.model.ts

```
export class Level
{
    private _name: string = "";
    private _description: string = "";

    public get Name(): string
    {
        return this._name;
    }
    private set Name(value: string)
    {
        if(value)
            this._name = value;
    }

    public get Description(): string
    {
        return this._description;
    }
    private set Description(value: string | undefined)
    {
        if(value)
            this._description = value;
    }

    constructor(name: string, description?: string)
    {
```

```
        this.Name = name;
        this.Description = description;
    }
}
```

Листинг В.8 – Файл device.model.ts

```
import { AtomicEvent } from "./events/atomic-event.model";
import { Event } from "./events/event.model";
import { Level } from "./level.model";
import { Role } from "./role.model";
import { System } from "./system.model";


export class Device
{
    private _code: string = "";
    private _description: string = "";

    private _level: Level | null = null;
    private _system: System | null = null;
    private _roles: Role[] = [];
    private _events: AtomicEvent[] = [];

    public get Code(): string
    {
        return this._code;
    }
    private set Code(value: string)
    {
        if(value)
            this._code = value;
    }

    public get Description(): string
    {
        return this._description;
    }
    private set Description(value: string | undefined)
    {
        if(value)
            this._description = value;
    }

    public get Level(): Level | null
    {
        return this._level;
    }
    public get System(): System | null
    {
        return this._system;
    }
    public get Roles(): Role[]
    {
        return this._roles;
    }
    public get Events(): Event[]
    {
        return this._events;
    }

    constructor(code: string, description?: string, level?: Level, parent?:
System)
```

```
    {
        this.Code = code;
        this.Description = description;
        this.SetLevel(level);
        this.SetParent(parent);
    }

    public SetParent(parent: System | undefined)
    {
        if(parent)
            this._system = parent;
    }

    public SetLevel(level: Level | undefined)
    {
        if(level)
            this._level = level;
    }

    public AddEvent(newEvent: AtomicEvent | undefined)
    {
        if(newEvent && !this._events.includes(newEvent))
            this._events.push(newEvent);
    }

    public AddRole(newRole: Role | undefined)
    {
        if(newRole && !this._roles.includes(newRole))
            this._roles.push(newRole);
    }
}
```

Листинг В.9 – Файл developer-environment.model.ts

```
import { Device } from "./device.model";
import { Level } from "./level.model";
import { System } from "./system.model";
import { Event, EventCategory } from "./events/event.model";
import { Parameter } from "./parameter.model";
import { Role } from "./role.model";
import { AtomicEvent } from "./events/atomic-event.model";
import { AggregationEvent } from "./events/aggregation-event.model";
import { ComplexEvent } from "./events/complex-event.model";
import { ActiveRule } from "./active-rule.model";

export class DeveloperEnvironment
{
    private _systems: System[] = [];
    private _devices: Device[] = [];
    private _events: Event[] = [];
    private _activeRules: ActiveRule[] = [];

    public get Systems(): System[]
    {
        return this._systems;
    }
    public get Devices(): Device[]
    {
        return this._devices;
    }
    public get Events(): Event[]
    {
        return this._events;
```

```
    }
    public get ActiveRules(): ActiveRule[]
    {
        return this._activeRules;
    }

    constructor(systems: any, devices: any, events: any, activeRules: any)
    {
        //Systems
        for(let i = 0; i < systems.length; i++)
        {
            this._systems.push(new System(systems[i].code, systems[i].name));
        }

        for(let i = 0; i < systems.length; i++)
        {
            if(systems[i].parent)
            {
                let parent = this._systems.find(system => system.Code ===
systems[i].parent.code);
                let child = this._systems.find(system => system.Code ===
systems[i].code);

                if(child)
                    child.SetParent(parent);

                if(parent)
                    parent.AddSystem(child);
            }
        }

        let parameter
        //AtomicEvents
        for(let i = 0; i < events.length; i++)
        {
            if(events[i].categoryEvent == "atomic")
            {
                parameter    =    new    Parameter(events[i].contextParam.code,
events[i].contextParam.name,               events[i].contextParam.description,
events[i].contextParam.dataType);
                this._events.push(new              AtomicEvent(events[i].code,
events[i].description, parameter));
            }
        }

        //Devices
        for(let i = 0; i < devices.length; i++)
        {
            let parent;

            if(devices[i].system)
                parent   =   this._systems.find(system   =>   system.Code   ===
devices[i].system.code);
            else
                parent = undefined;

            let newDevice = new Device(devices[i].code, devices[i].description,
undefined, parent);
            this._devices.push(newDevice);

            for(let role of devices[i].role)
                newDevice.AddRole(new Role(role.name, role.description));

            for(let relatedEvent of devices[i].event)
```

```
            {
                let atomicEvent = this._events.find(event => event.Code ===
relatedEvent.code) as AtomicEvent;
                newDevice.AddEvent(atomicEvent);
                atomicEvent.AddParent(newDevice);
            }
        }

        //AggregationEvents
        for(let i = 0; i < events.length; i++)
        {
            if(events[i].categoryEvent == "aggregation")
            {
                parameter    =    new    Parameter(events[i].contextParam.code,
events[i].contextParam.name,               events[i].contextParam.description,
events[i].contextParam.dataType);
                let aggregationEvent = new AggregationEvent(events[i].code,
events[i].description,          parameter,          events[i].unixWindowStart,
events[i].unixWindowEnd);
                this._events.push(aggregationEvent);

                for(let pair of events[i].atomicEvent)
                {
                    let atomicEvent = this._events.find(event => event.Code ===
pair.event.code) as AtomicEvent;
                    let device = this._devices.find(device => device.Code ===
pair.component.code);

                    if(atomicEvent && device)
                        aggregationEvent.AddAtomicPair(atomicEvent, device);
                }
            }
        }

        //ComplexEvents
        for(let i = 0; i < events.length; i++)
        {
            if(events[i].categoryEvent == "complex")
            {
                parameter    =    new    Parameter(events[i].contextParam.code,
events[i].contextParam.name,               events[i].contextParam.description,
events[i].contextParam.dataType);
                this._events.push(new             ComplexEvent(events[i].code,
events[i].description, parameter));
            }
        }

        for(let i = 0; i < events.length; i++)
        {
            if(events[i].categoryEvent == "complex")
            {
                let complexEvent = this._events.find(event => event.Code ===
events[i].code) as ComplexEvent;

                for(let inbound of events[i].inboundEvent)
                {
                    let inboundEvent = this._events.find(event => event.Code
=== inbound.code);
                    if(inboundEvent)
                        complexEvent.AddInboundEvent(inboundEvent);
                }
            }
        }
```

```
        //ActiveRules
        for(let i = 0; i < activeRules.length; i++)
        {
            let    activeRule    =    new    ActiveRule(activeRules[i].code,
activeRules[i].description)
            this._activeRules.push(activeRule);
            for(let event of activeRules[i].event)
            {
                let    inboundEvent    =    this._events.find(foundEvent    =>
foundEvent.Code === event.code)
                if(inboundEvent)
                    activeRule.AddEvent(inboundEvent);
            }
        }
    }
}
```

Листинг В.10 – Файл active-rule.model.ts

```
import { Event, EventCategory } from "./events/event.model";

export class ActiveRule
{
    private _code: string = "";
    private _description: string = "";

    private _action: string = "";
    private _condition: string = "";
    private _events: Event[] = [];

    public get Code(): string
    {
        return this._code;
    }
    private set Code(value: string)
    {
        if(value)
            this._code = value;
    }

    public get Description(): string
    {
        return this._description;
    }
    private set Description(value: string | undefined)
    {
        if(value)
            this._description = value;
    }

    public get Action(): string
    {
        return this._action;
    }
    private set Action(value: string | undefined)
    {
        if(value)
            this._action = value;
    }

    public get Condition(): string
    {
        return this._condition;
```

```
    }
    private set Condition(value: string | undefined)
    {
        if(value)
            this._condition = value;
    }

    public get Events(): Event[]
    {
        return this._events;
    }

    constructor(code:   string,   description?:   string,   action?:   string,
condition?: string)
    {
        this.Code = code;
        this.Description = description;
        this.Action = action;
        this.Condition = condition;
    }

    AddEvent(event: Event)
    {
        if(event)
        {
            this._events.push(event);
        }
    }
}
```

Листинг В.11 – Файл event.model.ts

```
import { ActiveRule } from "../active-rule.model";
import { Device } from "../device.model";
import { Parameter } from "../parameter.model";


export class Event
{
    private _code: string = "";
    private _description: string = "";
    private _category: EventCategory = EventCategory.ATOMIC;

    private _parameter: Parameter | null = null;

    private _activeRules: ActiveRule[] = [];

    public get Code(): string
    {
        return this._code;
    }
    private set Code(value: string)
    {
        if(value)
            this._code = value;
    }

    public get Description(): string
    {
        return this._description;
    }
    private set Description(value: string | undefined)
    {
```

```
        if(value)
            this._description = value;
    }

    public get Category(): EventCategory
    {
        return this._category;
    }
    private set Category(value: EventCategory)
    {
        if(value)
            this._category = value;
    }

    public get Parameter(): Parameter | null
    {
        return this._parameter;
    }

    constructor(code: string, description?: string, category?: EventCategory,
param?: Parameter)
    {
        this.Code = code;
        this.Description = description;
        if(category)
            this.Category = category;
        this.SetParameter(param);
    }

    public SetParameter(param: Parameter | undefined)
    {
        if(param)
            this._parameter = param;
    }

    public AddActiveRule(activeRule: ActiveRule)
    {
        if(activeRule)
            this._activeRules.push(activeRule);
    }
}

export enum EventCategory
{
    ATOMIC,
    AGGREGATION,
    COMPLEX
}
```

Листинг В.12 – Файл compex-event.model.ts

```
import { Device } from "../device.model";
import { Parameter } from "../parameter.model";
import { Event, EventCategory } from "./event.model";

export class ComplexEvent extends Event
{
    private _inboundEvents: Event[] = [];
    // private _template: Event[] = [];

    constructor(code: string, description?: string, param?: Parameter)
    {
        super(code, description, EventCategory.COMPLEX, param)
```

```
    }

    public AddInboundEvent(event: Event)
    {
        if(event)
            this._inboundEvents.push(event);
    }
}
```

Листинг В.13 – Файл atomic-event.model.ts

```
import { Device } from "../device.model";
import { Parameter } from "../parameter.model";
import { Event, EventCategory } from "./event.model";

export class AtomicEvent extends Event
{
    private _parents: Device[] = [];

    public get Parents(): Device[]
    {
        return this._parents;
    }

    constructor(code: string, description?: string, param?: Parameter)
    {
        super(code, description, EventCategory.ATOMIC, param)
    }

    public AddParent(device: Device)
    {
        if(device)
            this._parents.push(device);
    }
}
```

Листинг В.14 – Файл aggregation-event.model.ts

```
import { Device } from "../device.model";
import { Parameter } from "../parameter.model";
import { AtomicEvent } from "./atomic-event.model";
import { Event, EventCategory } from "./event.model";

export class AggregationEvent extends Event
{
    private _timeWindowStart: string = "";
    private _timeWindowEnd: string = "";

    private _atomicEvents: {event: AtomicEvent, device: Device}[] = [];
//{device: Device, event: Event}

    public get TimeStart(): string
    {
        return this._timeWindowStart;
    }
    private set TimeStart(value: string)
    {
        this._timeWindowStart = value;
    }

    public get TimeEnd(): string
    {
```

```
        return this._timeWindowEnd;
    }
    private set TimeEnd(value: string)
    {
        this._timeWindowEnd = value;
    }

    constructor(code:  string,  description?:  string,  param?:  Parameter,
timeStart?: string, timeEnd?: string)
    {
        super(code, description, EventCategory.AGGREGATION, param);
        if(timeStart)
            this._timeWindowStart = timeStart;
        if(timeEnd)
            this._timeWindowEnd = timeEnd;
    }

    public AddAtomicPair(event: AtomicEvent, device: Device)
    {
        if(event && device)
            this._atomicEvents.push({event, device});
    }
}
```

Приложение Г

ЛИСТИНГ СЕРВИСОВ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ

(рекомендуемое)

Листинг Г.1 – Файл http-request.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({providedIn: 'root'})   //Сервис общий для проекта
export class HTTPRequestService
{
  //Импорт объекта сервиса для отправки http запросов
  constructor(private httpClient: HttpClient) {}
  /**
   * Функция запроса GET к серверу
   * @param url URL-адрес, к которому направляется запрос
   * @param additionalHeaders Дополнительные заголовки запроса
   * @returns Объект типа Observable, при подписке возвращает информацию от
сервера
  */
  public get(url: string, additionalHeaders?: HttpHeaders): Observable<any>
  {
    let headers = new HttpHeaders
    (
      {
        'Content-Type': 'application/json',
        'Accept': '*/*',
      }
    );

        //Добавление дополнительный заголовков к запросу
    if(additionalHeaders)
    {
      additionalHeaders.keys().forEach(key =>
      {
        let value = additionalHeaders.get(key);

        if(headers.has(key))
          headers = headers.set(key, value ? value : "");
        else
          headers = headers.append(key, value ? value : "");
      });
    }

    //Объект опций для ответа на запрос
    let options: Object =
    {
      headers: headers,
      observe: 'body',
      withCredentials: true
    };

    return this.httpClient.get<any>(url, options);
  }

  /**
   * Функция запроса POST к серверу
   * @param url URL-адрес, к которому направляется запрос
   * @param additionalHeaders Дополнительные заголовки запроса
   * @returns Объект типа Observable, при подписке возвращает информацию от
сервера
  */
  public post(url: string, object: any, additionalHeaders?: HttpHeaders):
Observable<any>
```

```
  {
    let headers = new HttpHeaders
    (
      {
        'Content-Type': 'application/json',
        'Accept': '*/*',
      }
    );

      //Добавление дополнительный заголовков к запросу
  if(additionalHeaders)
  {
    additionalHeaders.keys().forEach(key =>
    {
      let value = additionalHeaders.get(key);

      if(headers.has(key))
        headers = headers.set(key, value ? value : "");
      else
        headers = headers.append(key, value ? value : "");
    });
  }

      //Объект опций для ответа на запрос
    let options: Object =
    {
      headers: headers,
      observe: 'body',
      withCredentials: true
    };

    return this.httpClient.post<any>(url, object, options);
  }

  /**
   * Функция запроса PATCH к серверу
   * @param url URL-адрес, к которому направляется запрос
   * @param additionalHeaders Дополнительные заголовки запроса
   * @returns Объект типа Observable, при подписке возвращает информацию от
сервера
  */
  public patch(url: string, object: any, additionalHeaders?: HttpHeaders):
Observable<any>
  {
    let headers = new HttpHeaders
    (
      {
        'Content-Type': 'application/json',
        'Accept': '*/*',
      }
    );

  //Добавление дополнительный заголовков к запросу
  if(additionalHeaders)
  {
    additionalHeaders.keys().forEach(key =>
    {
      let value = additionalHeaders.get(key);

      if(headers.has(key))
        headers = headers.set(key, value ? value : "");
      else
        headers = headers.append(key, value ? value : "");
    });
```

```
  }

  //Объект опций для ответа на запрос
    let options: Object =
    {
      headers: headers,
      observe: 'body',
      withCredentials: true
    };

    return this.httpClient.patch<any>(url, object, options);
  }

  /**
   * Функция запроса PATCH к серверу
   * @param url URL-адрес, к которому направляется запрос
   * @param additionalHeaders Дополнительные заголовки запроса
   * @returns Объект типа Observable, при подписке возвращает информацию от
сервера
  */
   public delete(url: string, object: any, additionalHeaders?: HttpHeaders):
Observable<any>
   {
     let headers = new HttpHeaders
     (
       {
         'Content-Type': 'application/json',
         'Accept': '*/*',
       }
     );

   //Добавление дополнительный заголовков к запросу
   if(additionalHeaders)
   {
     additionalHeaders.keys().forEach(key =>
     {
       let value = additionalHeaders.get(key);

       if(headers.has(key))
         headers = headers.set(key, value ? value : "");
       else
         headers = headers.append(key, value ? value : "");
     });
   }

   //Объект опций для ответа на запрос
     let options: Object =
     {
       headers: headers,
       observe: 'body',
       withCredentials: true,
       body: object
     };

     return this.httpClient.delete<any>(url, options);
   }


  public GetAuthorizationHeader(): HttpHeaders
  {
    return new HttpHeaders
    (
      {
        'Authorization': 'Bearer ' + localStorage.getItem("accessToken")
```

```
      }
    );
  }
}
```

Листинг Г.2 – Файл user-request.service.ts

```
import { HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { environment } from 'src/environments/environment';
import { HTTPRequestService } from '../http-request/http-request.service';

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

  constructor(public httpRequest: HTTPRequestService) { }

  public GetAuthorizationHeader(): HttpHeaders
  {
    return new HttpHeaders
    (
      {
        'Authorization': 'Bearer ' + localStorage.getItem("accessToken")
      }
    );
  }

  public RefreshRequest(): Observable<any>
  {
    return
this.httpRequest.post(`${environment.DOMEN}${environment.PORT}${environment.A
UTH}${environment.REFRESH}`,                                    {"refreshToken":
localStorage.getItem("refreshToken")}, this.GetAuthorizationHeader());
  }

  public LogoutRequest(): Observable<any>
  {
    return
this.httpRequest.post(`${environment.DOMEN}${environment.PORT}${environment.A
UTH}${environment.LOGOUT}`,                                     {"refreshToken":
localStorage.getItem("refreshToken")}, this.GetAuthorizationHeader());
  }

  public async Refresh(): Promise<boolean>
  {
    let hasRefreshed = false;

    await this.RefreshRequest().toPromise().then
    (
      (responce:any) =>
      {
        localStorage.removeItem("accessToken");
        localStorage.setItem("accessToken", responce["accessToken"]);

        localStorage.removeItem("refreshToken");
        localStorage.setItem("refreshToken", responce["refreshToken"]);

        hasRefreshed = true;
      }
    ).catch
```

```
    (
      (errorResponce:any) =>
      {
        localStorage.removeItem("accessToken");
        localStorage.removeItem("refreshToken");

        hasRefreshed = false;
      }
    );

    return hasRefreshed;
  }

  public async Logout(): Promise<any>
  {
    await this.LogoutRequest().toPromise().then
    (
      (responce:any) =>
      {
        localStorage.removeItem("accessToken");
        localStorage.removeItem("refreshToken");
      }
    ).catch
    (
      (errorResponce:any) =>
      {
        console.log(errorResponce);
      }
    );
  }

  public async MakeRequest(requestFunction: RequestFunction, data?: any,
httpRequest?: HTTPRequestService): Promise<any>
  {
    let requestService: HTTPRequestService;
    if(httpRequest == undefined)
    {
      requestService = this.httpRequest
    }
    else
    {
      requestService = httpRequest;
    }

    let _callback: {isError: boolean, responce: any} = {isError: false,
responce: {}};

    await requestFunction(requestService, data).toPromise()
    .then
    (
      (responce:any) =>
      {
        _callback.responce = responce;
      }
    )
    .catch
    (
      async (error:any) =>
      {
        if(error.error.statusCode == 403)
        {
          await this.Refresh().then
          (
            async (hasRefreshed: any) =>
```

```
            {
              if(hasRefreshed)
              {
                await requestFunction(requestService, data).toPromise()
                  .then((responce:any) =>
                  {
                    _callback.responce = responce
                  })
                  .catch((errorResponce: any) =>
                  {
                    _callback.isError = true;
                    _callback = errorResponce;
                  });
              }
              else
              {
                _callback.isError = true;
              }
            }
          );
        }
        else
        {
          _callback.isError = true;
          _callback.responce = error;
        }
      }
    );

    return _callback;
  }
}

export type RequestFunction = (httpRequest: HTTPRequestService, data?: any) =>
Observable<any>;
```

Листинг Г.3 – Файл auth.service.ts

```
import { Injectable } from '@angular/core';
import { HTTPRequestService } from 'src/app/core/services/http-request/http-
request.service';
import { environment } from 'src/environments/environment';

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

  constructor(private httpRequest: HTTPRequestService) { }

  public get isAuthorized(): boolean
  {
    return (localStorage.getItem("accessToken") != null) ? true : false;
  }

  public login(email:string, password:string )
  {
    return this.httpRequest.post
    (

`${environment.DOMEN}${environment.PORT}${environment.AUTH}${environment.LOGI
N}`,
      {"email": email, "password" : password}
    );
  }
```

```
  public registration(email:string, password:string)
  {
    return this.httpRequest.post
    (

`${environment.DOMEN}${environment.PORT}${environment.AUTH}${environment.REGI
STRATION}`,
      {"email": email, "password" : password}
    );
  }
}
```

Листинг Г.4 – Файл administrator-request.service.ts

```
import { Injectable } from '@angular/core';
import { async, Observable } from 'rxjs';
import { HTTPRequestService } from 'src/app/core/services/http-request/http-
request.service';
import { RequestFunction, UserRequestService } from
'src/app/core/services/user-request/user-request.service';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class AdministratorRequestService extends UserRequestService
{
  constructor(public httpRequest: HTTPRequestService)
  {
    super(httpRequest);
  }

  public GetUsersRequest(httpRequest: HTTPRequestService): Observable<any>
  {
    return
httpRequest.get(`${environment.DOMEN}${environment.PORT}${environment.USER}`,
httpRequest.GetAuthorizationHeader());
  }

  public GetUserRequest(httpRequest: HTTPRequestService, userData: any):
Observable<any>
  {
    return
httpRequest.get(`${environment.DOMEN}${environment.PORT}${environment.USER}/$
{userData.code}`, httpRequest.GetAuthorizationHeader());
  }

  public GetRolesRequest(httpRequest: HTTPRequestService): Observable<any>
  {
    return
httpRequest.get(`${environment.DOMEN}${environment.PORT}${environment.USER_RO
LE}`, httpRequest.GetAuthorizationHeader());
  }

  public RegisterUserRequest(httpRequest: HTTPRequestService, userData: any):
Observable<any>
  {
    return httpRequest.post
    (

`${environment.DOMEN}${environment.PORT}${environment.AUTH}${environment.REGI
STRATION}`,
```

```
      userData, httpRequest.GetAuthorizationHeader()
    );
  }

  public SetRoleRequest(httpRequest: HTTPRequestService, userData: any):
Observable<any>
  {
    return httpRequest.post
    (

`${environment.DOMEN}${environment.PORT}${environment.USER}${environment.ROLE
}`,
      userData, httpRequest.GetAuthorizationHeader()
    );
  }

  public RemoveRoleRequest(httpRequest: HTTPRequestService, userData: any):
Observable<any>
  {
    return httpRequest.delete
    (

`${environment.DOMEN}${environment.PORT}${environment.USER}${environment.ROLE
}`,
      userData, httpRequest.GetAuthorizationHeader()
    );
  }

  public DeleteUserRequest(httpRequest: HTTPRequestService, userData: any):
Observable<any>
  {
    return httpRequest.delete
    (
      `${environment.DOMEN}${environment.PORT}${environment.USER}`,
      userData, httpRequest.GetAuthorizationHeader()
    );
  }

  public UpdateUserRequest(httpRequest: HTTPRequestService, userData: any):
Observable<any>
  {
    return httpRequest.patch
    (
      `${environment.DOMEN}${environment.PORT}${environment.USER}`, userData,
httpRequest.GetAuthorizationHeader()
    );
  }
}
```

Листинг Г.5 – Файл developer-request.service.ts

```
import { HttpHeaders } from '@angular/common/http';
import { Injectable, Injector } from '@angular/core';
import { Observable } from 'rxjs/internal/Observable';
import { HTTPRequestService } from 'src/app/core/services/http-request/http-
request.service';
import { UserRequestService } from 'src/app/core/services/user-request/user-
request.service';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
```

```
export class DeveloperRequestService extends UserRequestService {

  constructor(httpRequest: HTTPRequestService)
  {
    super(httpRequest);
  }

  public GetSystemsRequest(httpRequest: HTTPRequestService): Observable<any>
  {
    return
httpRequest.get(`${environment.DOMEN}${environment.PORT}${environment.SYSTEM}
`, httpRequest.GetAuthorizationHeader());
  }

  public GetDevicesRequest(httpRequest: HTTPRequestService): Observable<any>
  {
    return
httpRequest.get(`${environment.DOMEN}${environment.PORT}${environment.DEVICE}
`, httpRequest.GetAuthorizationHeader());
  }

  public GetEventsRequest(httpRequest: HTTPRequestService): Observable<any>
  {
    return
httpRequest.get(`${environment.DOMEN}${environment.PORT}${environment.EVENT}`
, httpRequest.GetAuthorizationHeader());
  }

  public        GetActiveRulesRequest(httpRequest:        HTTPRequestService):
Observable<any>
  {
    return
httpRequest.get(`${environment.DOMEN}${environment.PORT}${environment.ACTIVE_
RULE}`, httpRequest.GetAuthorizationHeader());
  }
}
```

Листинг Г.6 – Файл options-request.service.ts

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { HTTPRequestService } from 'src/app/core/services/http-request/http-
request.service';
import { UserRequestService } from 'src/app/core/services/user-request/user-
request.service';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class OptionsRequestService extends UserRequestService
{
  constructor(httpRequest: HTTPRequestService)
  {
    super(httpRequest);
  }

  public   ChangePasswordRequest(httpRequest:   HTTPRequestService,   userData:
any): Observable<any>
  {
    return httpRequest.post
    (
```

```
`${environment.DOMEN}${environment.PORT}${environment.AUTH}${environment.CHAN
GE_PASSWORD}`,
      userData, httpRequest.GetAuthorizationHeader()
    );
  }
}
```

Приложение Д

ДИАГРАММЫ ФУНКЦИОНАЛЬНЫХ БЛОКОВ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

(рекомендуемое)

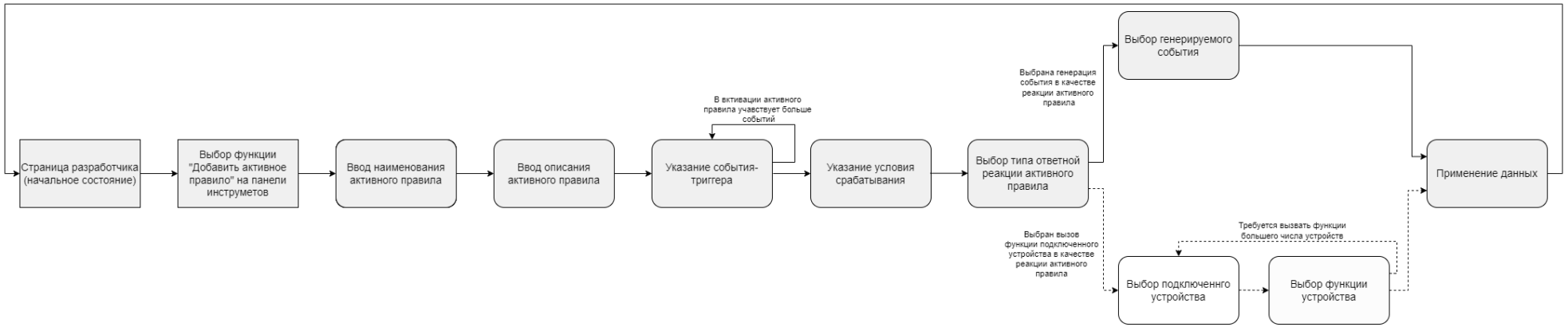


Рисунок Д.1 – Диаграмма функциональных блоков пользовательского интерфейса при реализации функции «Создать активное правило»

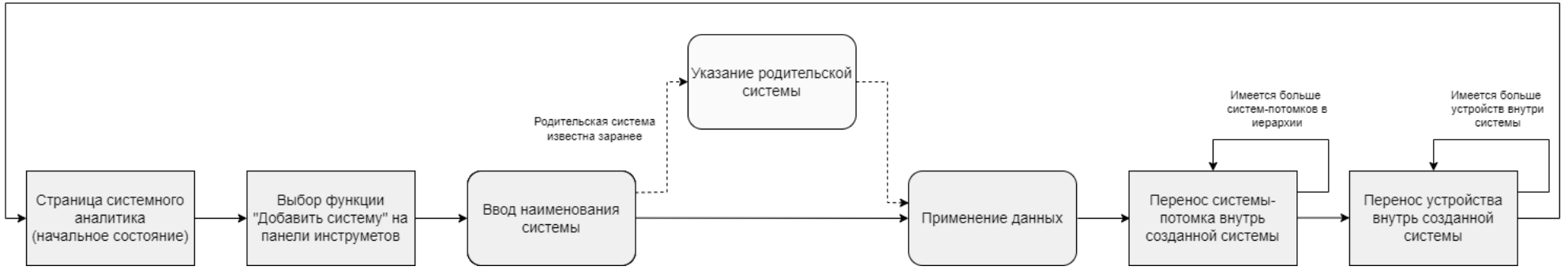


Рисунок Д.2 – Диаграмма функциональных блоков пользовательского интерфейса при реализации функции «Добавить и наполнить систему»

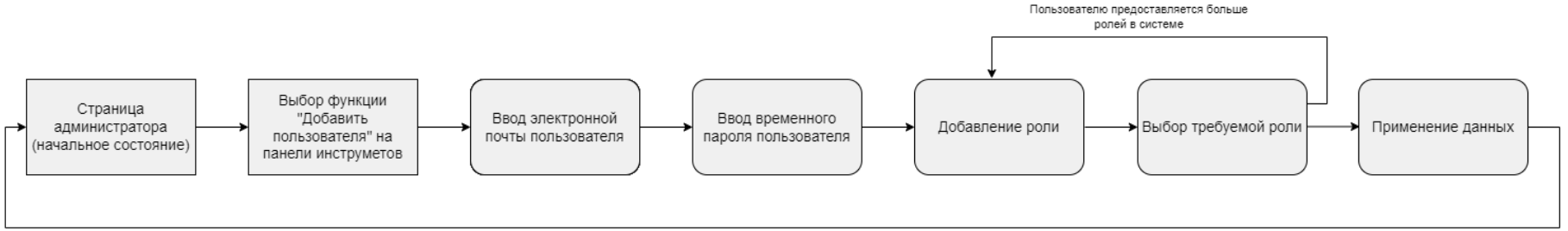


Рисунок Д.3 – Диаграмма функциональных блоков пользовательского интерфейса при реализации функции «Добавить пользователя»

Приложение Е

ЛИСТИНГ МОДУЛЬНЫХ ТЕСТОВ

(рекомендуемое)

Листинг Е.1 – Файл login.component.spec.ts

```
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { FormControl, FormGroup, FormsModule, NgForm, ReactiveFormsModule }
from '@angular/forms';
import { RouterTestingModule } from '@angular/router/testing';

import { LoginComponent } from './login.component';

xdescribe('LoginComponent', () => {
  let component: LoginComponent;
  let fixture: ComponentFixture<LoginComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ LoginComponent ],
      imports:
      [
        HttpClientModule,
        RouterTestingModule,
        FormsModule,
        ReactiveFormsModule
      ],
      providers:
      [
        HttpClient
      ]})
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(LoginComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('Должен быть создан', () => {
    expect(component).toBeTruthy();
  });

  it('Должен отобразить форму', () => {
    expect(fixture.nativeElement.querySelector('#login')).toBeTruthy();
  });

  it('Должен считать значения с полей формы', async () =>
  {
    localStorage.clear();
    let data =
    {
      email: "kirill.tambovtsev.00@mail.ru",
      password: "tttt"
    }

    fixture.detectChanges();
    await fixture.whenStable().then(async () =>
    {
      const emailInput: HTMLInputElement =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const passwordInput: HTMLInputElement =
fixture.debugElement.nativeElement.querySelector("#password-field");
      emailInput.value = data.email;
```

```
      passwordInput.value = data.password;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));

      expect(component.form.value.email).toEqual(data.email);
      expect(component.form.value.password).toEqual(data.password);
    });
  });

  it('Должен записать в localStorage полученный accessToken при верно введенных
значениях', async () =>
  {
    localStorage.clear();
    let data =
    {
      email: "kirill.tambovtsev.00@mail.ru",
      password: "tttt"
    }

    fixture.detectChanges();
    await fixture.whenStable().then(async () =>
    {
      const             emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const            passwordInput:            HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#password-field");
      emailInput.value = data.email;
      passwordInput.value = data.password;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));

      await component.login();
      expect(localStorage.getItem('accessToken')).toBeFalsy();
    });
  });

  it('Должен выдать ошибку при вводе неверных данных', async () =>
  {
    localStorage.clear();
    fixture.detectChanges();

      let data =
      {
        email: "qwerty",
        password: "22"
      }

      const             emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const            passwordInput:            HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#password-field");
      emailInput.value = data.email;
      passwordInput.value = data.password;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));

      await component.login();
      fixture.detectChanges();
      expect(fixture.debugElement.nativeElement.querySelector("#error-
field")).not.toBeTruthy();
  });
```

```
});
```

Листинг Е.2 – Файл registration.component.spec.ts

```
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { RouterTestingModule } from '@angular/router/testing';

import { RegistrationComponent } from './registration.component';

xdescribe('RegistrationComponent', () => {
  let component: RegistrationComponent;
  let fixture: ComponentFixture<RegistrationComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ RegistrationComponent ],
      imports:
      [
        HttpClientModule,
        RouterTestingModule,
        FormsModule,
        ReactiveFormsModule
      ],
      providers:
      [
        HttpClient
      ]})
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(RegistrationComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('Должен быть создан', () => {
    expect(component).toBeTruthy();
  });

  it('Должен считать значения с полей формы', async () =>
  {
    localStorage.clear();
    let data =
    {
      email: "kirill.tambovtsev.00@mail.ru",
      password: "tttt",
      repeatedPassword: "tttt"
    }

    fixture.detectChanges();
    await fixture.whenStable().then(async () =>
    {
      const             emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const            passwordInput:            HTMLInputElement            =
fixture.debugElement.nativeElement.querySelector("#password-field");
      const         repeatedPasswordInput:         HTMLInputElement         =
fixture.debugElement.nativeElement.querySelector("#repeatedPassword-field");
      emailInput.value = data.email;
```

```
      passwordInput.value = data.password;
      repeatedPasswordInput.value = data.repeatedPassword;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));
      repeatedPasswordInput.dispatchEvent(new Event('input'));

      expect(component.form.value.email).toEqual(data.email);
      expect(component.form.value.password).toEqual(data.password);

expect(component.form.value.repeatedPassword).toEqual(data.repeatedPassword);
    });
  });

  it('Должен записать в localStorage полученный accessToken при верно введенных
значениях', async () =>
  {
    localStorage.clear();
    let data =
    {
      email: "kirill.tambovtsev.00@mail.ru",
      password: "tttt",
      repeatedPassword: "tttt"
    }

    fixture.detectChanges();
    await fixture.whenStable().then(async () =>
    {
      const             emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const            passwordInput:            HTMLInputElement            =
fixture.debugElement.nativeElement.querySelector("#password-field");
      const          repeatedPasswordInput:          HTMLInputElement          =
fixture.debugElement.nativeElement.querySelector("#repeatedPassword-field");
      emailInput.value = data.email;
      passwordInput.value = data.password;
      repeatedPasswordInput.value = data.repeatedPassword;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));
      repeatedPasswordInput.dispatchEvent(new Event('input'));

      await component.registration();
      expect(localStorage.getItem('accessToken')).toBeFalsy();
    });
  });

  it('Должен выдать ошибку при вводе неверных данных', async () =>
  {
    localStorage.clear();
    fixture.detectChanges();

      let data =
      {
        email: "qwerty",
        password: "22",
        repeatedPassword: "11"
      }

      const             emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const            passwordInput:            HTMLInputElement            =
fixture.debugElement.nativeElement.querySelector("#password-field");
```

```
      const repeatedPasswordInput: HTMLInputElement =
fixture.debugElement.nativeElement.querySelector("#repeatedPassword-field");
      emailInput.value = data.email;
      passwordInput.value = data.password;
      repeatedPasswordInput.value = data.repeatedPassword;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));
      repeatedPasswordInput.dispatchEvent(new Event('input'));

      await component.registration();
      fixture.detectChanges();
      expect(fixture.debugElement.nativeElement.querySelector("#error-
field")).not.toBeTruthy();
  });
});
```

Листинг Е.3 – Файл authorization.component.spec.ts

```
import { CUSTOM_ELEMENTS_SCHEMA, NO_ERRORS_SCHEMA } from '@angular/core';
import { ComponentFixture, TestBed } from '@angular/core/testing';

import { AuthorizationComponent } from './authorization.component';

xdescribe('AuthorizationComponent', () => {
  let component: AuthorizationComponent;
  let fixture: ComponentFixture<AuthorizationComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations:
      [
        AuthorizationComponent,
      ],
      schemas:
      [
        CUSTOM_ELEMENTS_SCHEMA,
        NO_ERRORS_SCHEMA
      ]
    })
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(AuthorizationComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('Должен быть создан', () => {
    expect(component).toBeTruthy();
  });

  it('Должен отобразить форму', () => {
    expect(fixture.nativeElement.querySelector('.container')).toBeTruthy();
  });

  it('Должен переключить вкладку входа на вкладку регистрации', () => {
    component.openLoginTab(false);
    fixture.detectChanges();
    expect(fixture.nativeElement.querySelector('#reg-tab-
content')).toBeTruthy();
```

```
    expect(fixture.nativeElement.querySelector('#login-tab-
content')).not.toBeTruthy();
  });
});
```

Листинг Е.4 – Файл auth.spec.ts

```
import { HttpClient, HttpClientModule, HttpHeaders } from
'@angular/common/http';
import { TestBed } from '@angular/core/testing';
import { Observable } from 'rxjs/internal/Observable';
import { HTTPRequestService } from 'src/app/core/services/http-request/http-
request.service';

import { AuthService } from './auth.service';

fdescribe('AuthService', () => {

  let service: AuthService;
  const mockHTTPRequestService =
  {
    get: () => new Observable(),
    post: () => new Observable()
  };

  beforeEach(() => {
    TestBed.configureTestingModule
    ({
      imports:
      [
        HttpClientModule
      ],
      providers:
      [
        HttpClient,
        {provide: HTTPRequestService, useValue: mockHTTPRequestService}
      ]
    });

    service = TestBed.inject(AuthService);
  });

  it('Должен быть создан', () => {
    expect(service).toBeTruthy();
  });

  it('Должен возвращать ответ на POST - запрос для входа', () => {
    let response = service.login("", "");
    expect(response).toBeInstanceOf(Observable);
  });

  it('Должен возвращать ответ на POST - запрос для регистрации', () => {
    let response = service.registration("", "");
    expect(response).toBeInstanceOf(Observable);
  });

});
```

Приложение Ж

ЛИСТИНГ ИНТЕГРАЦИОННЫХ ТЕСТОВ

(рекомендуемое)

Листинг Ж.1 – Файл registration.component.int.spec.ts

```
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { RouterTestingModule } from '@angular/router/testing';

import { RegistrationComponent } from './registration.component';

fdescribe('RegistrationComponent (интеграционные тесты)', () => {
  let component: RegistrationComponent;
  let fixture: ComponentFixture<RegistrationComponent>;

  let mockRouter =
  {
    navigate: jasmine.createSpy('navigate')
  };

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ RegistrationComponent ],
      imports:
      [
        HttpClientModule,
        RouterTestingModule,
        FormsModule,
        ReactiveFormsModule
      ],
      providers:
      [
        HttpClient
      ]})
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(RegistrationComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('Должен обратиться к серверу и записать в localStorage полученный
accessToken при верно введенных значениях', async () =>
  {
    localStorage.clear();
    let data =
    {
      email: "kirill.tambovtsev.00@mail.ru",
      password: "tttt",
      repeatedPassword: "tttt"
    }

    fixture.detectChanges();
    await fixture.whenStable().then(async () =>
    {
      const             emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const            passwordInput:            HTMLInputElement            =
fixture.debugElement.nativeElement.querySelector("#password-field");
      const         repeatedPasswordInput:         HTMLInputElement         =
fixture.debugElement.nativeElement.querySelector("#repeatedPassword-field");
```

```
      emailInput.value = data.email;
      passwordInput.value = data.password;
      repeatedPasswordInput.value = data.repeatedPassword;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));
      repeatedPasswordInput.dispatchEvent(new Event('input'));

      await component.registration();
      expect(localStorage.getItem('accessToken')).toBeFalsy();
    });
  });

  it('Должен перейти на страницу /workspace при получении accessToken', async
() =>
  {
    localStorage.clear();
    let data =
    {
      email: "kirill.tambovtsev.00@mail.ru",
      password: "tttt",
      repeatedPassword: "tttt"
    }

    const              emailInput:              HTMLInputElement              =
fixture.debugElement.nativeElement.querySelector("#email-field");
    const             passwordInput:             HTMLInputElement              =
fixture.debugElement.nativeElement.querySelector("#password-field");
    emailInput.value = data.email;
    passwordInput.value = data.password;

    emailInput.dispatchEvent(new Event('input'));
    passwordInput.dispatchEvent(new Event('input'));

    await component.registration();
    expect(localStorage.getItem('accessToken')).toBeFalsy();

    // expect(mockRouter.navigate).toHaveBeenCalledWith(['/workspace']);
  });

  it('Должен выдать ошибку при вводе неверных данных', async () =>
  {
    localStorage.clear();
    fixture.detectChanges();

      let data =
      {
        email: "qwerty",
        password: "22",
        repeatedPassword: "11"
      }

      const              emailInput:              HTMLInputElement              =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const             passwordInput:             HTMLInputElement              =
fixture.debugElement.nativeElement.querySelector("#password-field");
      const          repeatedPasswordInput:          HTMLInputElement          =
fixture.debugElement.nativeElement.querySelector("#repeatedPassword-field");
      emailInput.value = data.email;
      passwordInput.value = data.password;
      repeatedPasswordInput.value = data.repeatedPassword;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));
```

```
      repeatedPasswordInput.dispatchEvent(new Event('input'));

      await component.registration();
      fixture.detectChanges();
      expect(fixture.debugElement.nativeElement.querySelector("#error-
field")).not.toBeTruthy();
  });
});
```

Листинг Ж.2 – Файл login.component.int.spec.ts

```
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { DebugElement, NO_ERRORS_SCHEMA } from '@angular/core';
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { FormControl, FormGroup, FormsModule, NgForm, ReactiveFormsModule }
from '@angular/forms';
import { Router } from '@angular/router';
import { RouterTestingModule } from '@angular/router/testing';

import { LoginComponent } from './login.component';


fdescribe('LoginComponent (интеграционные тесты)', () => {
  let component: LoginComponent;
  let fixture: ComponentFixture<LoginComponent>;

  let mockRouter =
  {
    navigate: jasmine.createSpy('navigate')
  };

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ LoginComponent ],
      imports:
      [
        HttpClientModule,
        RouterTestingModule,
        FormsModule,
        ReactiveFormsModule
      ],
      providers:
      [
        HttpClient,
        {
          provide: Router, useValue: mockRouter
        }
      ],
      schemas: [ NO_ERRORS_SCHEMA ]})
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(LoginComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });


  it('Должен обратиться к серверу и записать в localStorage полученный
accessToken при верно введенных значениях', async () =>
  {
    localStorage.clear();
```

```
    let data =
    {
      email: "kirill.tambovtsev.00@mail.ru",
      password: "tttt"
    }

    fixture.detectChanges();
    await fixture.whenStable().then(async () =>
    {
      const             emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
      const            passwordInput:            HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#password-field");
      emailInput.value = data.email;
      passwordInput.value = data.password;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));

      await component.login();
      expect(localStorage.getItem('accessToken')).toBeFalsy();
    });
  });

  it('Должен перейти на страницу /workspace при получении accessToken', async
() =>
  {
    localStorage.clear();
    let data =
    {
      email: "kirill.tambovtsev.00@mail.ru",
      password: "tttt"
    }

    const              emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
    const             passwordInput:            HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#password-field");
    emailInput.value = data.email;
    passwordInput.value = data.password;

    emailInput.dispatchEvent(new Event('input'));
    passwordInput.dispatchEvent(new Event('input'));

    await component.login();
    expect(localStorage.getItem('accessToken')).toBeFalsy();

    // expect(mockRouter.navigate).toHaveBeenCalledWith(['/workspace']);
  });

  it('Должен обратиться к серверу и выдать ошибку при вводе неверных данных',
async () =>
  {
    localStorage.clear();
    fixture.detectChanges();

      let data =
      {
        email: "qwerty",
        password: "22"
      }

      const             emailInput:             HTMLInputElement             =
fixture.debugElement.nativeElement.querySelector("#email-field");
```

```
      const            passwordInput:            HTMLInputElement            =
fixture.debugElement.nativeElement.querySelector("#password-field");
      emailInput.value = data.email;
      passwordInput.value = data.password;

      emailInput.dispatchEvent(new Event('input'));
      passwordInput.dispatchEvent(new Event('input'));

      await component.login();
      fixture.detectChanges();
      expect(fixture.debugElement.nativeElement.querySelector("#error-
field")).not.toBeTruthy();
  });
});
```

Приложение И

РЕЗУЛЬТАТЫ КОНФИГУРАЦИОННОГО ТЕСТИРОВАНИЯ

(рекомендуемое)

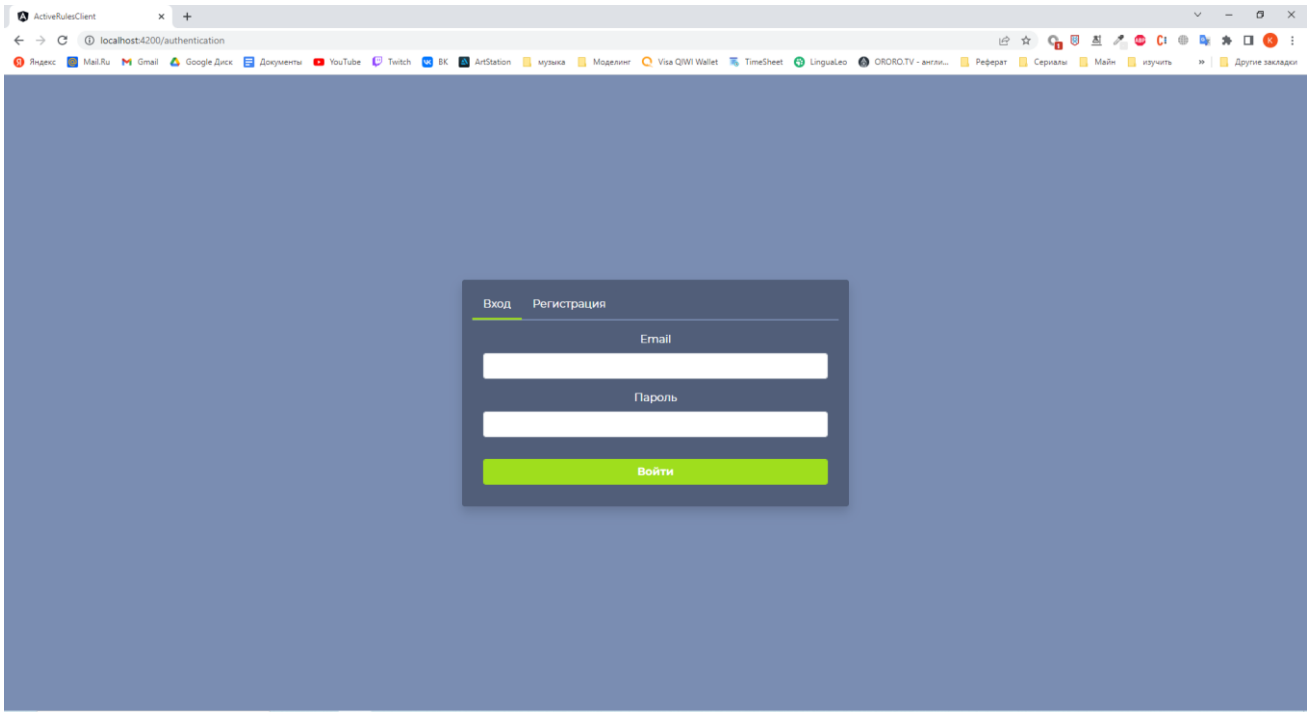


Рисунок И.1 – Результаты конфигурационного тестирования №1

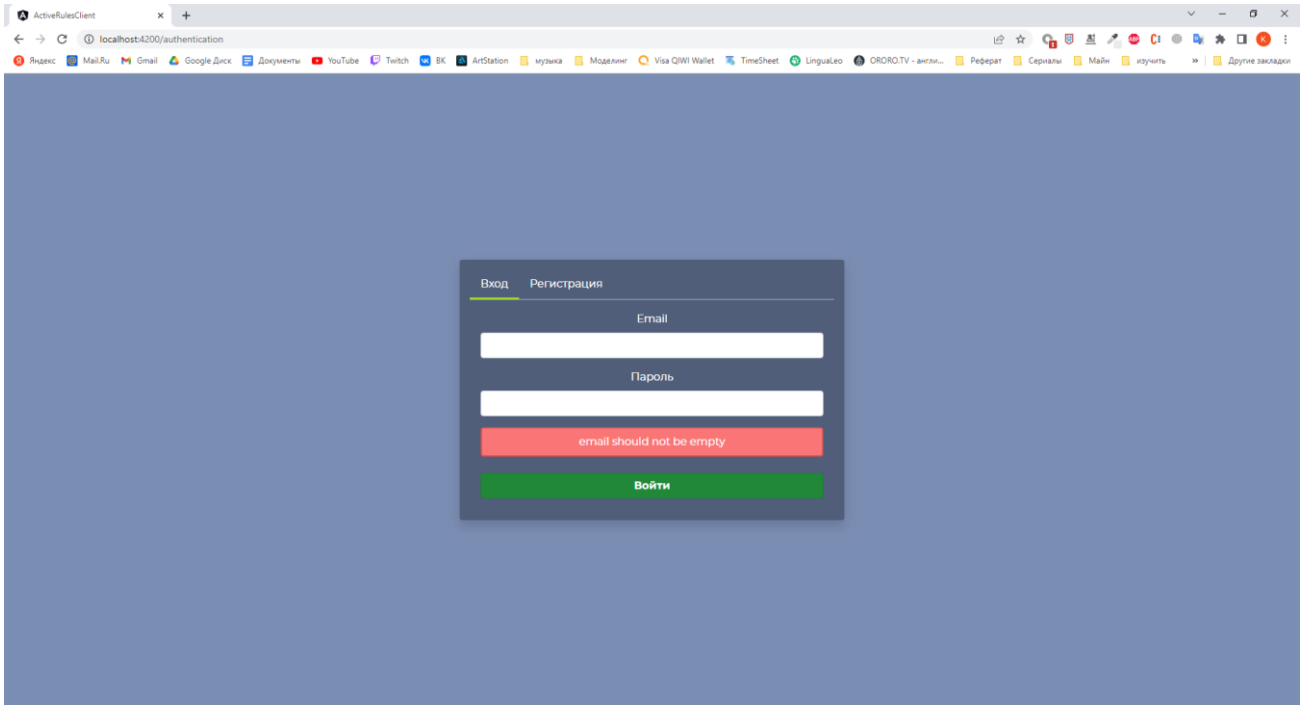


Рисунок И.2 – Результаты конфигурационного тестирования №2

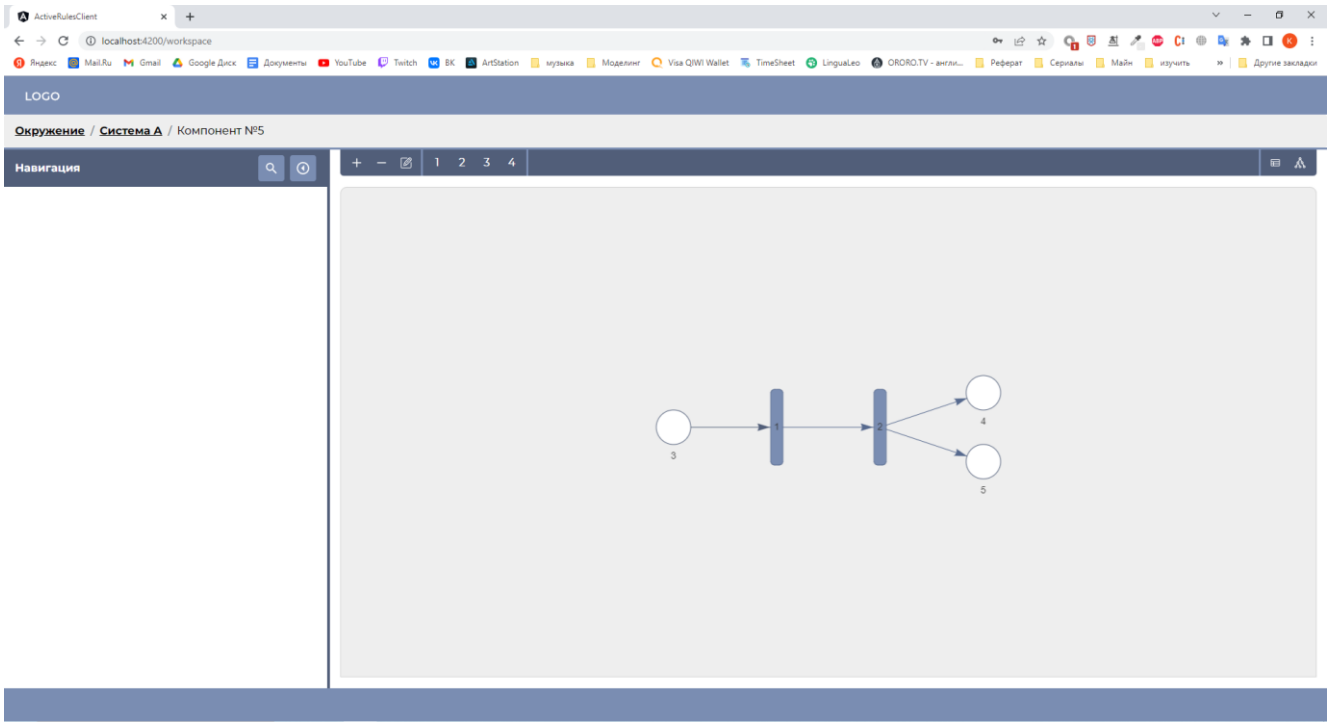


Рисунок И.3 – Результаты конфигурационного тестирования №3

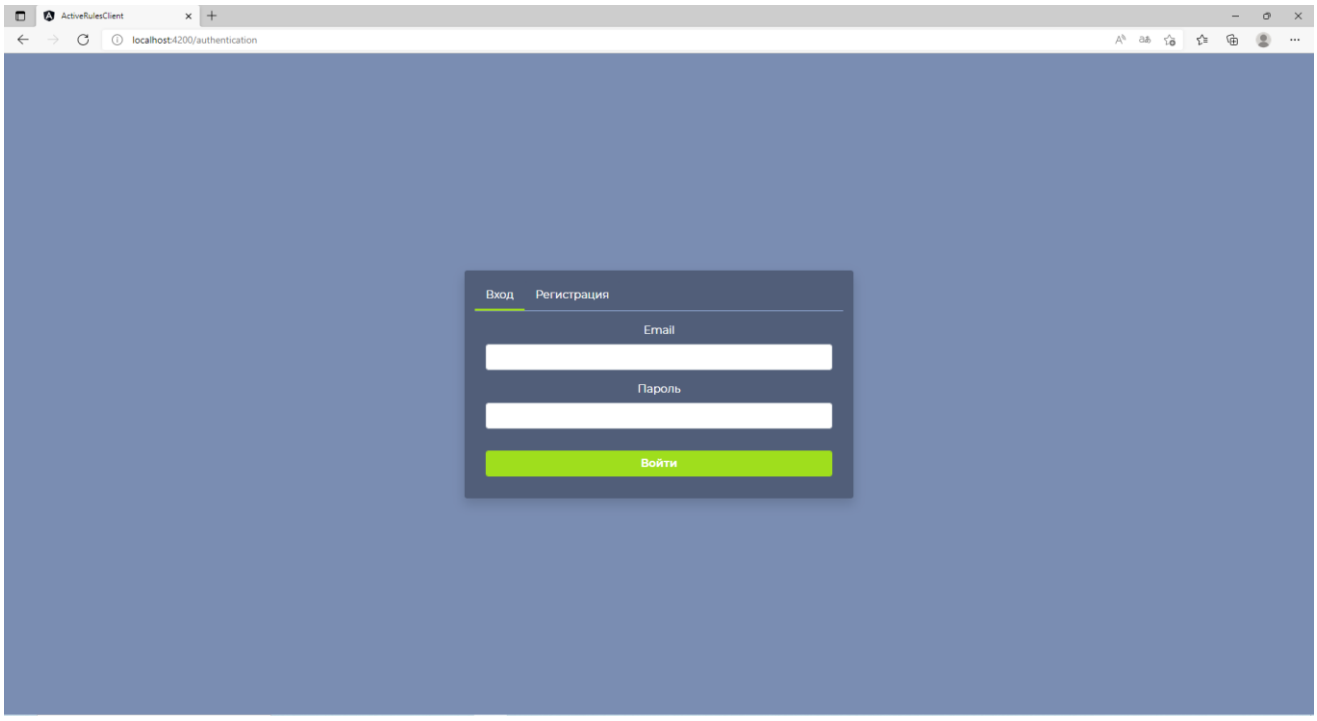


Рисунок И.4 – Результаты конфигурационного тестирования №4

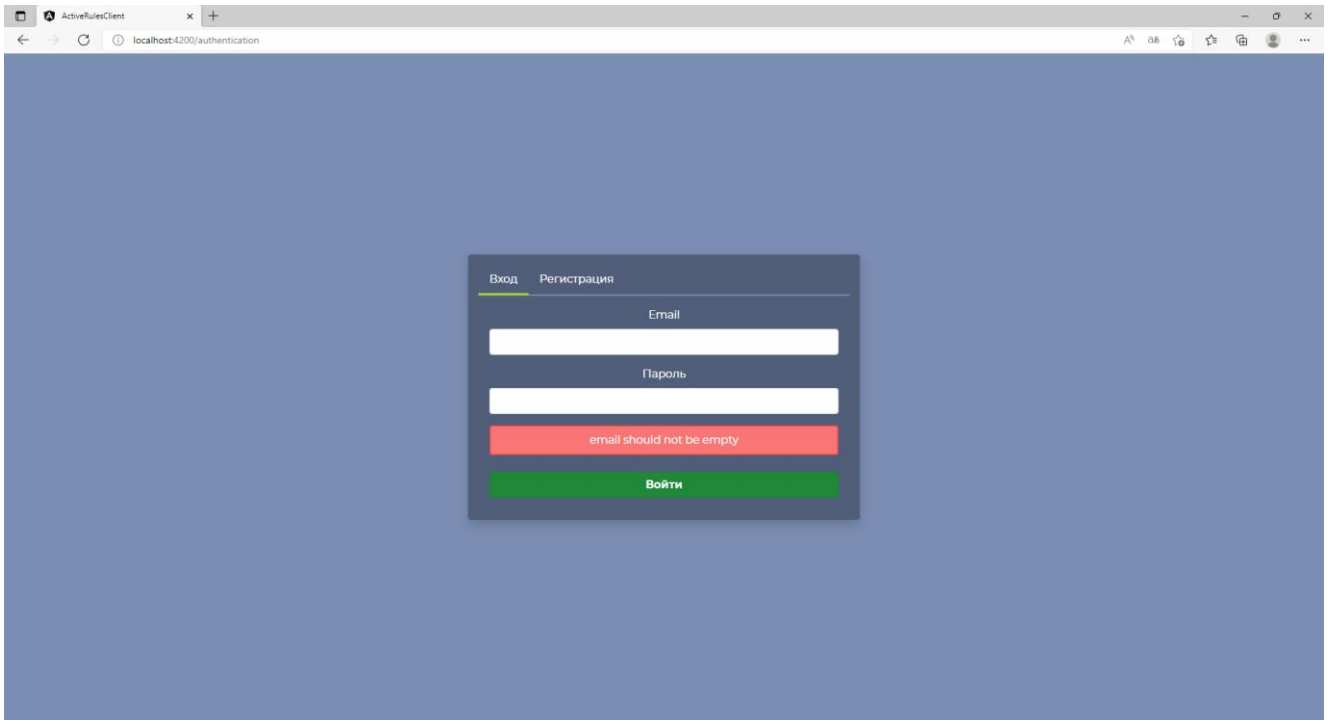


Рисунок И.5 – Результаты конфигурационного тестирования №5

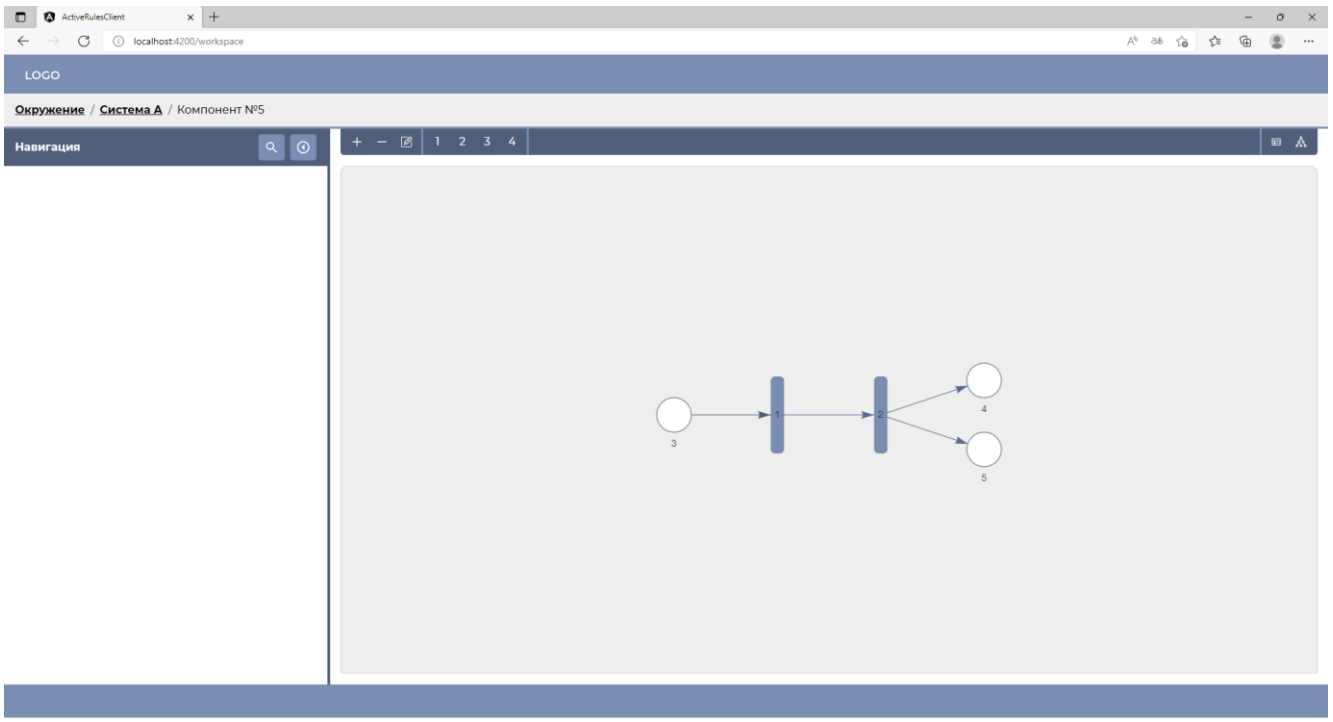


Рисунок И.6 – Результаты конфигурационного тестирования №6

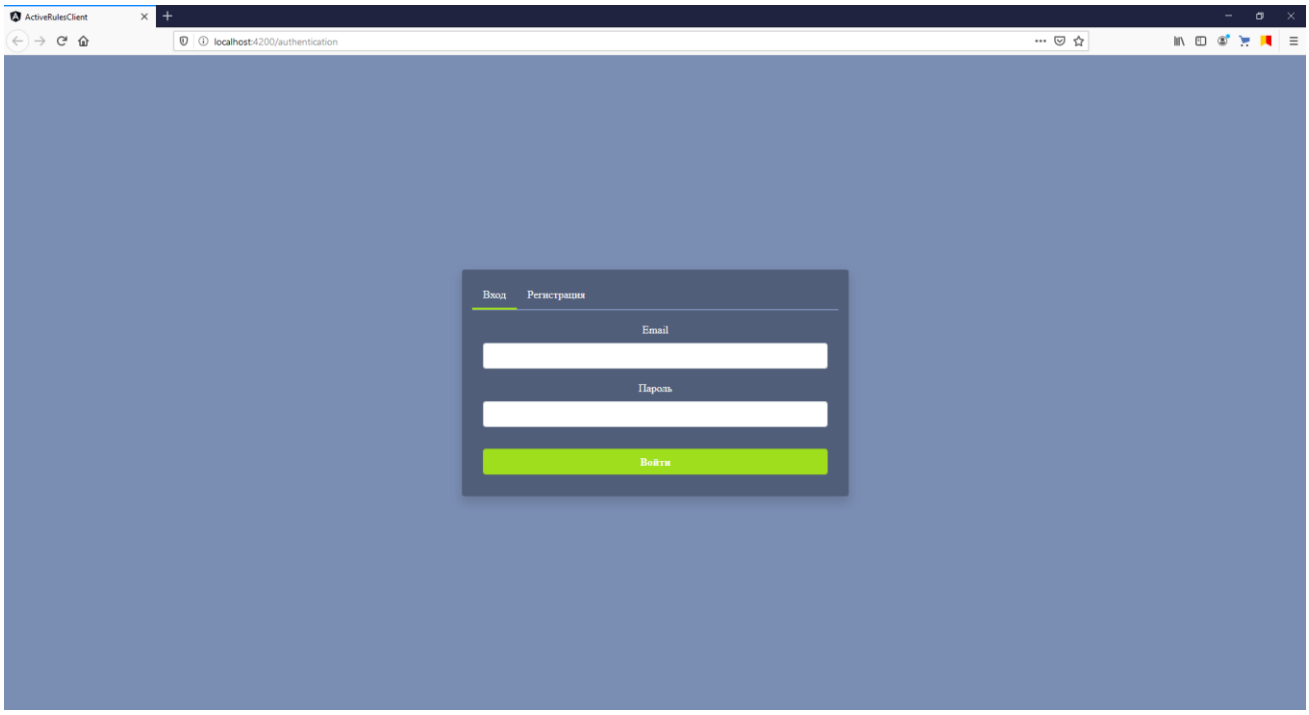


Рисунок И.7 – Результаты конфигурационного тестирования №7

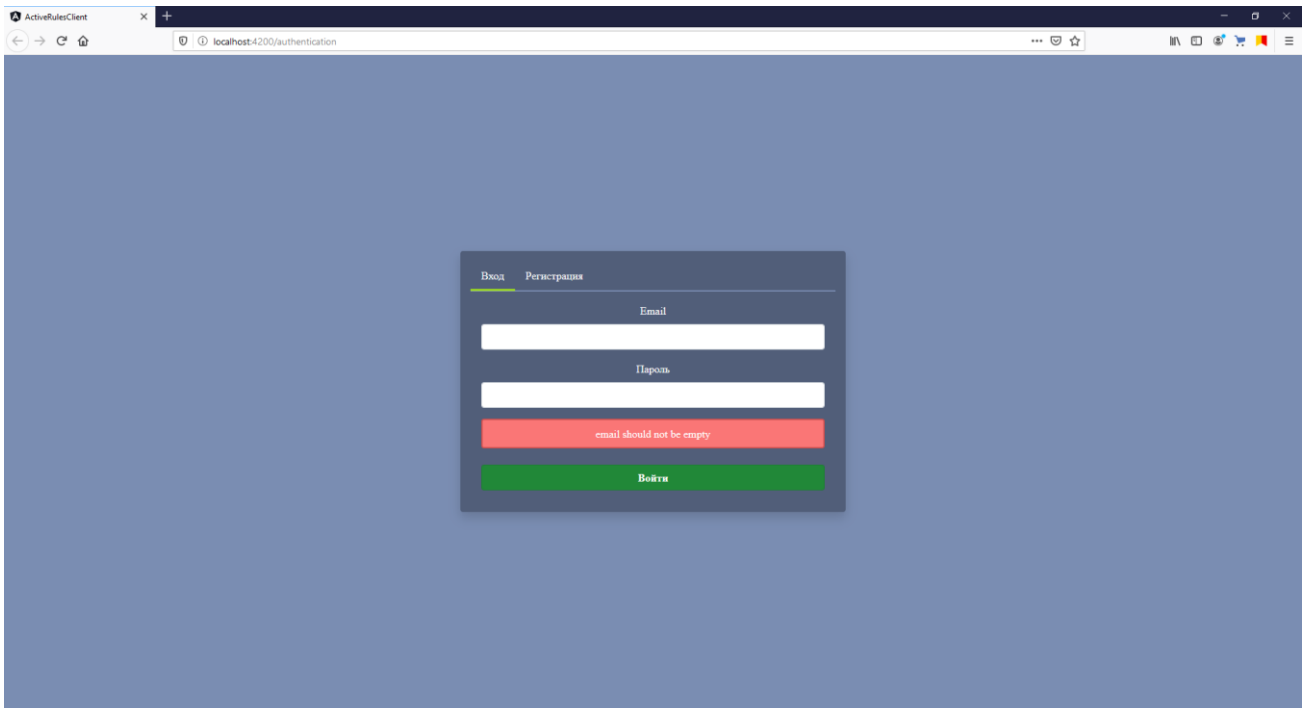


Рисунок И.8 – Результаты конфигурационного тестирования №8

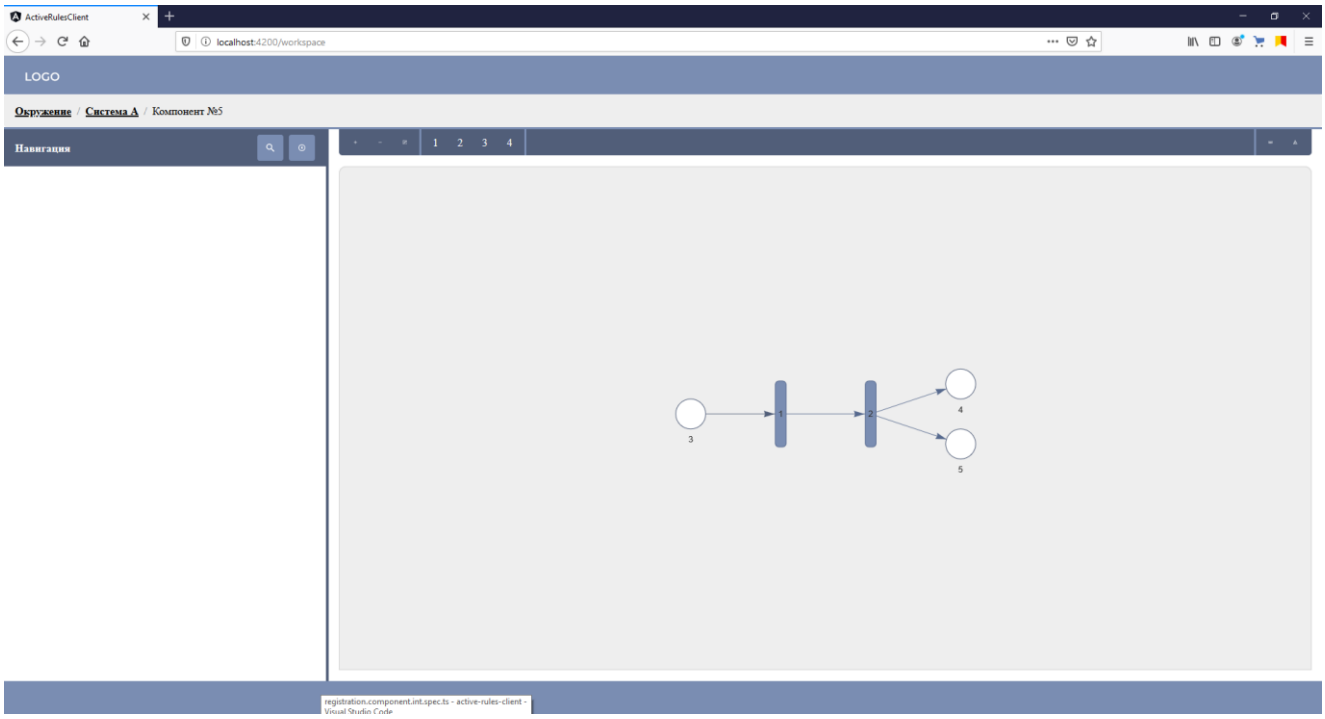


Рисунок И.9 – Результаты конфигурационного тестирования №9

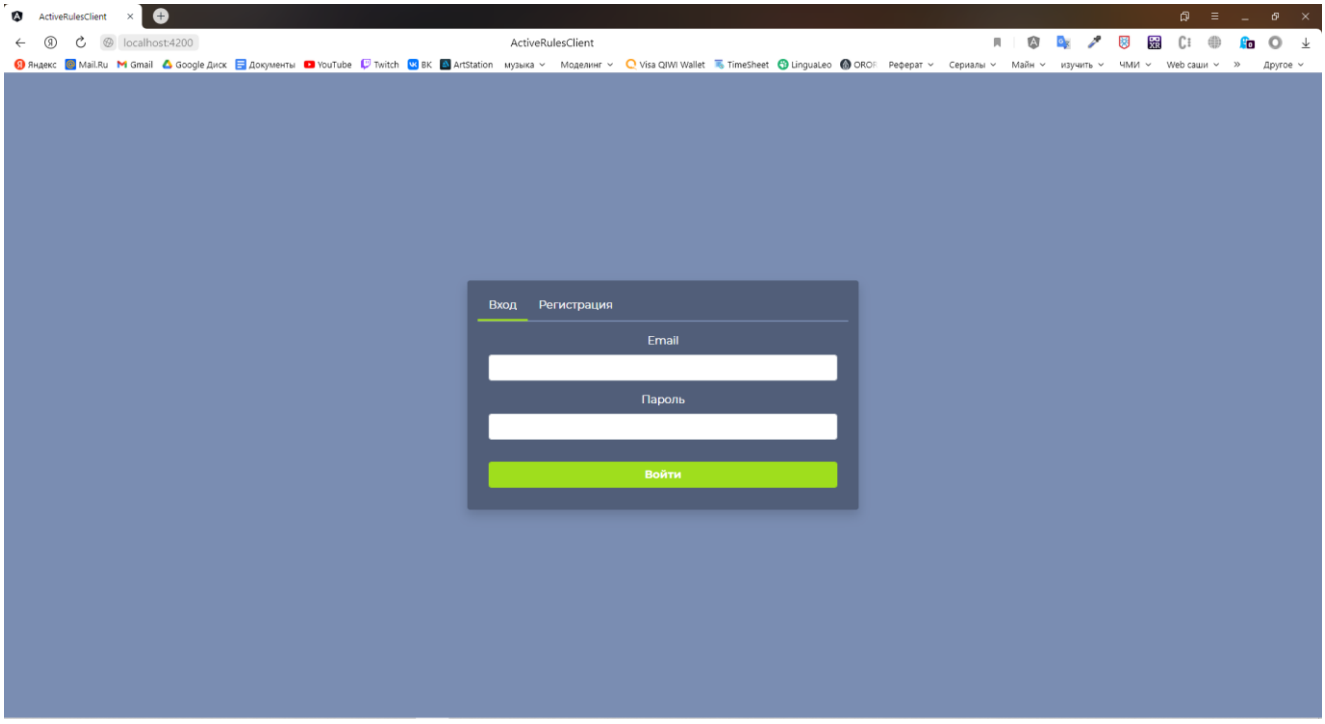


Рисунок И.10 – Результаты конфигурационного тестирования №10

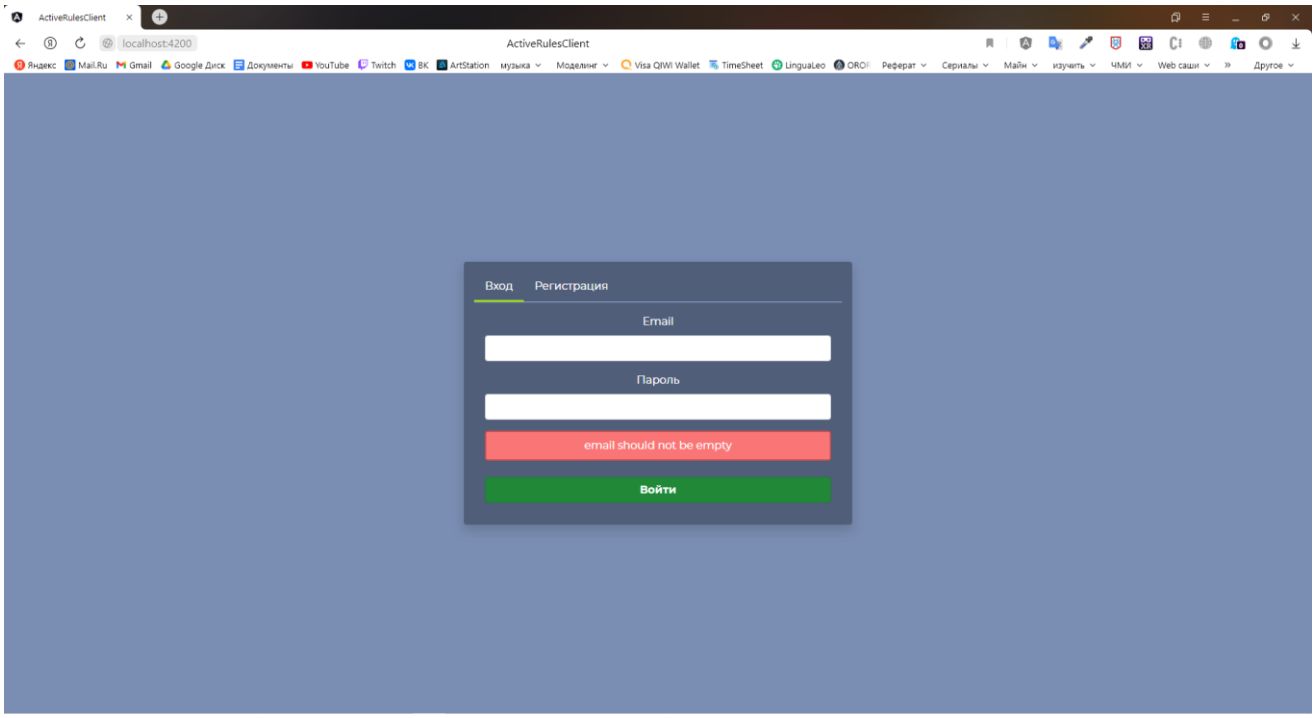


Рисунок И.11 – Результаты конфигурационного тестирования №11

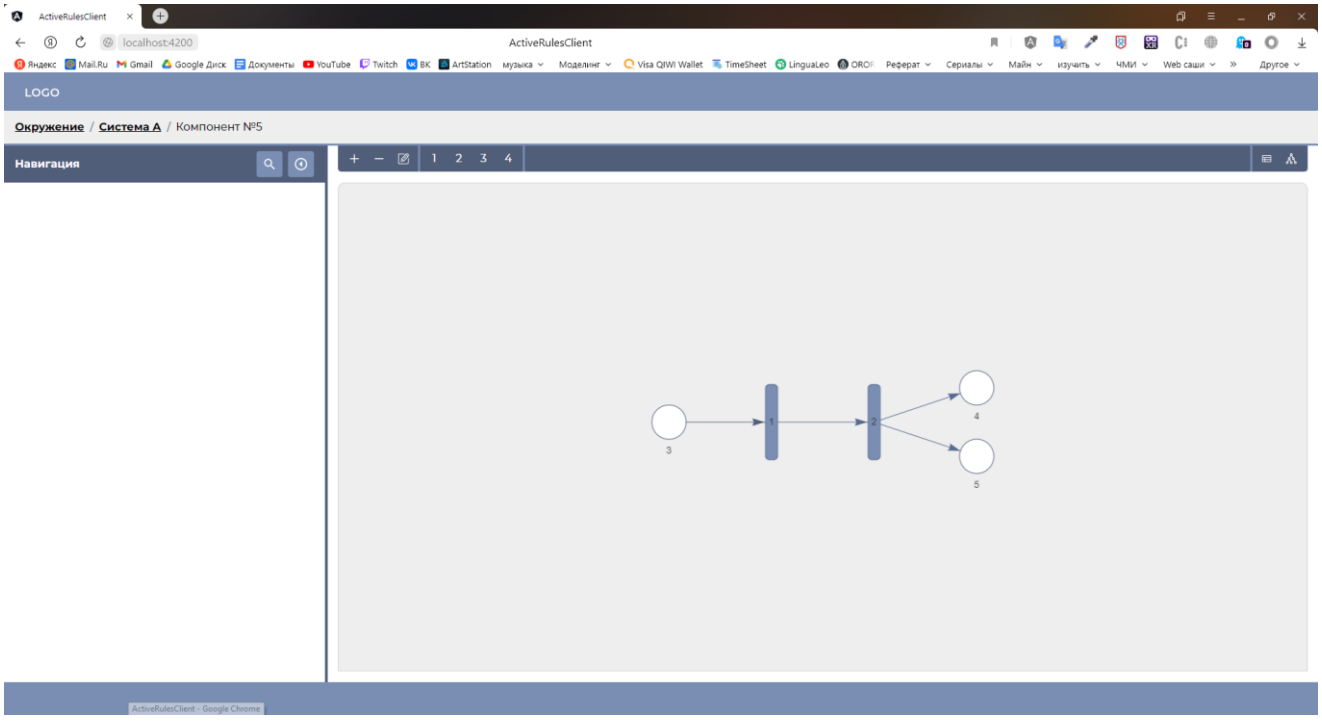


Рисунок И.12 – Результаты конфигурационного тестирования №12

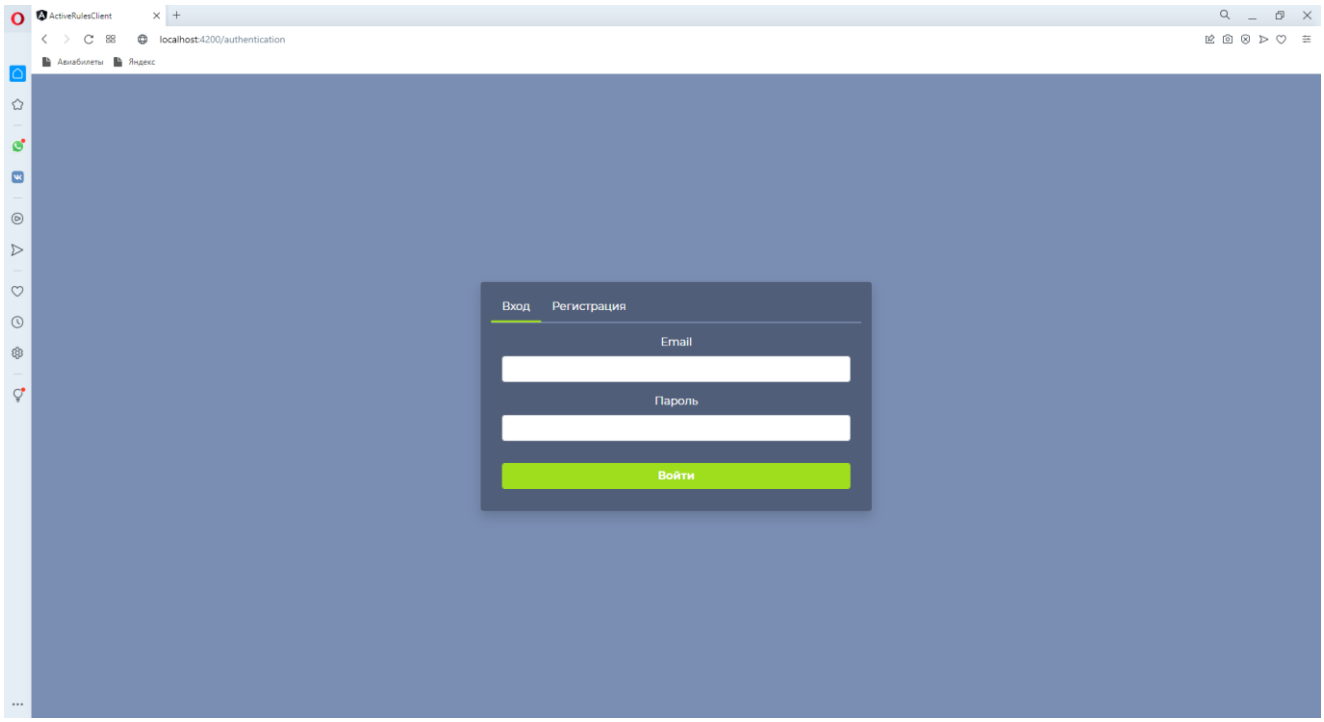


Рисунок И.13 – Результаты конфигурационного тестирования №13

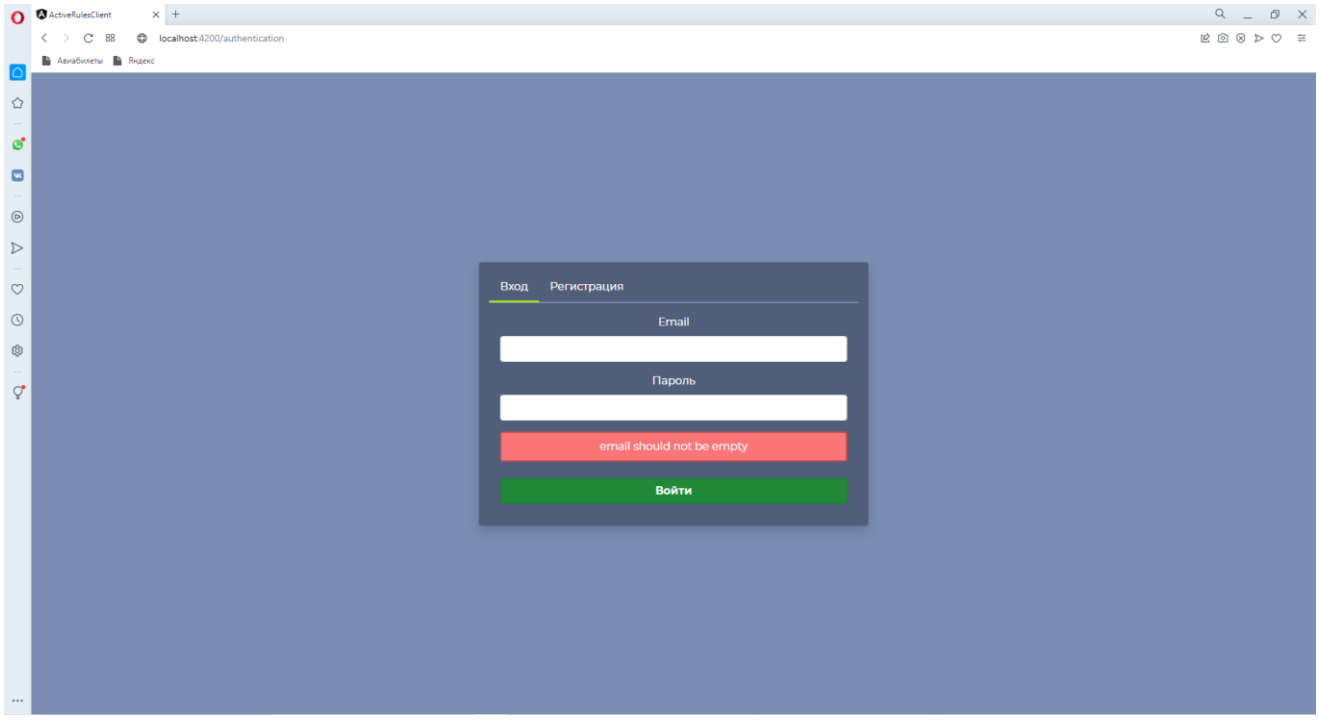


Рисунок И.14 – Результаты конфигурационного тестирования №14

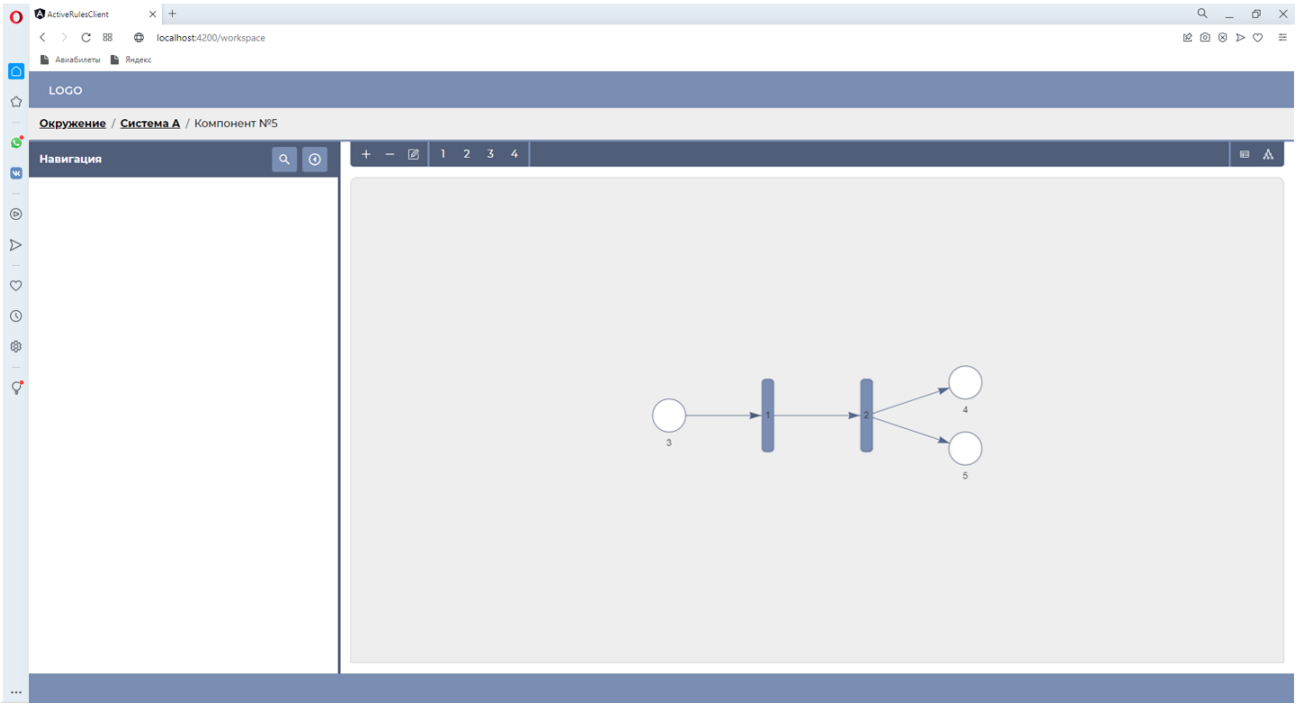


Рисунок И.15 – Результаты конфигурационного тестирования №15