

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

*Факультет*  
вычислительной техники

*Кафедра*  
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

**БАКАЛАВРСКАЯ РАБОТА**

на тему

**Программные средства управления журналами  
событий и хранилищами исторических данных  
в составе информационно-управляющей  
платформы**

Студент

\_\_\_\_\_  
(подпись, дата)

Якупов Х. А.

\_\_\_\_\_  
(ФИО полностью)

Руководитель

\_\_\_\_\_  
(подпись, дата)

Шибанов С.В.

\_\_\_\_\_  
(фамилия, инициалы)

Нормоконтролёр

\_\_\_\_\_  
(подпись, дата)

Такташкин Д.В.

\_\_\_\_\_  
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от \_\_\_\_\_ № \_\_\_\_\_)

Заведующий кафедрой

\_\_\_\_\_  
(подпись)

Козлов А.Ю.

\_\_\_\_\_  
(фамилия, инициалы)

Работа защищена с отметкой \_\_\_\_\_ (протокол заседания ГЭК от \_\_\_\_\_ № \_\_\_\_\_)

Секретарь ГЭК

\_\_\_\_\_  
(подпись)

Попова Н.А.

\_\_\_\_\_  
(фамилия, инициалы)

Пенза, 2025

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

Факультет  
Вычислительной техники

Кафедра  
МО и ПЭВМ

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

**ЗАДАНИЕ  
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ  
БАКАЛАВРА**

1. Студент Якупов Хаким Алиевич гр. 21ВИ1 факультета ФВТ  
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства управления журналами событий и хранилищами исторических данных в составе информационно-управляющей платформы  
Тема утверждена приказом ПГУ № \_\_\_\_\_ от " \_\_\_\_\_ " \_\_\_\_\_ 20 \_\_\_\_\_

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

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

1) Назначение: Программные средства управления журналами событий и хранилищами исторических данных в составе информационно-управляющей платформы

2) Среда выполнения: Windows 11

Языковые средства разработки: C#, KSQL, UML

Инструментальные средства: ASP.NET, Visual Studio, Winforms, Apache Kafka, Kafka Connect, ksqlDB, MongoDB, Redis

Технологии разработки: ООП, сервисно-ориентированный подход.

3) Основные функции разрабатываемого приложения:

a) Добавление логической переменной;

b) Сохранение исторических данных в долговременное хранилище;

c) Сохранение исторических данных в оперативное хранилище;

d) Просмотр актуальных данных из промежуточного хранилища.

## 5. Перечень подлежащих разработке вопросов

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

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

*3) Разработка;*

*4) Контроль качества программного обеспечения;*

*5) Оформление пояснительной записки.*

## 5. Календарный график выполнения работы

| №<br>п/п | Наименование этапов работы                       | Объем<br>работы | Срок<br>выполнения         | Подпись<br>руководителя |
|----------|--------------------------------------------------|-----------------|----------------------------|-------------------------|
| 1        | Анализ предметной области и требований к системе | 10%             | 08.04.2025 –<br>14.04.2025 |                         |
| 2        | Проектирование программных средств               | 20%             | 15.04.2025 –<br>25.04.2025 |                         |
| 3        | Реализация программных средств                   | 30%             | 26.04.2025 –<br>12.05.2025 |                         |
| 4        | Тестирование программных средств                 | 20%             | 13.05.2025 –<br>22.05.2025 |                         |
| 5        | Оформление пояснительной записки                 | 20%             | 23.05.2025 –<br>03.06.2025 |                         |

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы \_\_\_\_\_  
(подпись, дата)

Задание к исполнению принял студент \_\_\_\_\_  
(подпись, дата)

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

## Реферат

Пояснительная записка содержит 99 листов, 22 рисунка, 26 таблиц, 27 использованных источника и 3 приложения.

C#, RESTFul API, Winforms, UML, KSQL, ksqlDB, Apache Kafka, Kafka Connect, Visual Studio, Redis

Объект исследования – программные средства управления журналами событий и хранилищами данных в состав информационно-управляющей платформы.

Цель выпускной квалификационной работы – разработать программные средства управления журналами событий и хранилищами исторических данных в составе информационно-управляющей платформы.

Технологии разработки – разработка проводилась на языке программирования C#. В качестве среды разработки использовался редактор программного кода Microsoft Visual Studio.

Среда исполнения – Windows 11.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы программные средства управления журналами событий и хранилищами исторических данных.

Все требования к разработке приложения соблюдены.

|           |      |               |         |      |                                                                                                              |              |      |        |
|-----------|------|---------------|---------|------|--------------------------------------------------------------------------------------------------------------|--------------|------|--------|
|           |      |               |         |      | ПГУ 09.03.02 – 08BP211.32 ПЗ                                                                                 |              |      |        |
|           |      |               |         |      |                                                                                                              |              |      |        |
| Изм.      | Лист | № докум.      | Подпись | Дата | Программные средства управления журналами событий и хранилищами исторических данных<br>Пояснительная записка | Лит.         | Лист | Листов |
| Разраб.   |      | Якупов Х. А.  |         |      |                                                                                                              |              |      |        |
| Пров.     |      | Шибанов С.В.  |         |      |                                                                                                              |              | 4    | 99     |
|           |      |               |         |      |                                                                                                              | Группа 21ВИ1 |      |        |
| Н. Контр. |      | Такташкин Д.В |         |      |                                                                                                              |              |      |        |
| Утв.      |      | Козлов А.Ю.   |         |      |                                                                                                              |              |      |        |

## Содержание

|                                                                                                                                                        |    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Введение .....                                                                                                                                         | 7  |
| 1 Анализ предметной области и планирование процесса разработки программных средств управления журналами событий и хранилищами исторических данных..... | 9  |
| 1.1 Low-code платформа для развертывания, эксплуатации и сопровождения информационно–управляющих систем.....                                           | 9  |
| 1.1.1 Назначение и основные возможности платформы .....                                                                                                | 9  |
| 1.1.2 Концептуальная модель информационно-управляющей системы на основе платформы .....                                                                | 12 |
| 1.1.2.1 Внешние информационно–управляющие системы .....                                                                                                | 13 |
| 1.1.2.2 События.....                                                                                                                                   | 14 |
| 1.1.2.3 Активные правила .....                                                                                                                         | 15 |
| 1.1.3 Обобщенная программная архитектура платформы.....                                                                                                | 17 |
| 1.2 Журнализация и хранение данных в процессе функционирования информационно-управляющих систем .....                                                  | 20 |
| 1.2.1 Понятие и классы логических переменных как объектов хранимых данных .....                                                                        | 20 |
| 1.2.2 Источники и потребители данных логических переменных .....                                                                                       | 21 |
| 1.2.3 Бизнес-процессы управления хранимыми данными логических переменных .....                                                                         | 22 |
| 1.3 Сравнительный анализ современных решений для хранения данных в информационно-управляющих платформах .....                                          | 23 |
| 1.3.1 Обоснование сравнительного анализа .....                                                                                                         | 23 |
| 1.3.2 Обзор программных платформ для анализа .....                                                                                                     | 24 |
| 1.3.3 Результаты сравнительного анализа и выводы.....                                                                                                  | 27 |
| 1.4 Планирование процесса разработки программных средств .....                                                                                         | 29 |
| 1.4.1 Цели разработки и назначение программных средств .....                                                                                           | 29 |
| 1.4.2 План-график разработки и планирование бюджета.....                                                                                               | 29 |
| 2 Анализ требований и постановка задачи на разработку программных средств управления журналами событий и хранилищами исторических данных.....          | 35 |
| 2.1 Анализ функциональных требований.....                                                                                                              | 35 |

|       |                                                                                                           |    |
|-------|-----------------------------------------------------------------------------------------------------------|----|
| 2.2   | Анализ нефункциональных требований .....                                                                  | 40 |
| 2.3   | Анализ требований к пользовательскому интерфейсу панели управления.....                                   | 42 |
| 2.4   | Анализ требований к аппаратно-программному обеспечению .....                                              | 42 |
| 2.4.1 | Требования к аппаратному обеспечению .....                                                                | 42 |
| 2.4.2 | Требования к системному программному обеспечению .....                                                    | 43 |
| 2.4.3 | Требования к средствам разработки .....                                                                   | 44 |
| 2.5   | Постановка задачи на разработку программных средств .....                                                 | 45 |
| 3     | Разработка программных средств управления журналами событий и хранилищами исторических данных .....       | 47 |
| 3.1   | Разработка архитектуры развертывания программных средств .....                                            | 47 |
| 3.2   | Разработка RESTFul API-сервера для взаимодействия с приложениями информационно-управляющей платформы..... | 49 |
| 3.3   | Разработка структуры для описания логических переменных .....                                             | 55 |
| 3.3   | Разработка клиентского приложения.....                                                                    | 56 |
| 3.3.1 | Разработка интерфейса .....                                                                               | 56 |
| 3.3.2 | Разработка программных средств клиентского приложения .....                                               | 60 |
| 4     | Контроль качества программных средств взаимодействия с объектами мониторинга и управления .....           | 62 |
| 4.1   | Планирование процесса контроля качества приложения .....                                                  | 62 |
| 4.1.1 | Мероприятия по контролю качества .....                                                                    | 62 |
| 4.1.2 | Обоснование выбора модели тестирования .....                                                              | 63 |
| 4.1.3 | Обоснование выбора метрик программного кода.....                                                          | 64 |
| 4.2   | Тестирование приложения .....                                                                             | 65 |
| 4.2.1 | Функциональное тестирование .....                                                                         | 65 |
| 4.3   | Оценка качества программного кода .....                                                                   | 67 |
|       | Заключение .....                                                                                          | 71 |
|       | Список использованных источников .....                                                                    | 72 |
|       | Приложение А (справочное) Глоссарий .....                                                                 | 76 |
|       | Приложение Б (обязательное) Листинг RESTFul API.....                                                      | 78 |
|       | Приложение В (рекомендуемое) Тестирование .....                                                           | 92 |

## Введение

Информационно-управляющие системы (ИУС) играют ключевую роль в обеспечении эффективного функционирования технических и технологических объектов. Современные ИУС представляют собой комплекс программных и аппаратных средств, предназначенных для сбора, обработки, хранения, отображения и передачи информации, необходимой для принятия решений оператором. Одним из важнейших компонентов ИУС является операторский интерфейс, обеспечивающий визуализацию данных, мониторинг текущего состояния систем и взаимодействие с пользователем.

В условиях роста объёмов обрабатываемой информации, повышения требований к скорости реакции на события и необходимости удалённого управления возрастает значимость разработки адаптируемых и расширяемых программных решений для визуализации и управления данными. Программные средства, обеспечивающие оперативный доступ к событиям, отображение критических сообщений, сортировку и фильтрацию информации, а также возможность настройки интерфейса под задачи конкретного пользователя, становятся необходимыми в промышленности, энергетике, транспорте, а также в сфере умных зданий и интернета вещей.

Целью выпускной квалификационной работы (ВКР) является разработка программных средств управления журналами событий и хранилищами исторических данных в составе информационно-управляющей платформы (ИУП). ИУП предназначена для развертывания, интеграции и сопровождения ИУС Low-code и No-code для индустриального IoT в режиме Low-code/No-code.

Для достижения поставленной цели были определены и решены следующие задачи:

- 1) анализ предметной области и планирование процесса разработки программных средств;
- 2) анализ требований и постановка задачи на разработку;
- 3) разработка программных средств;
- 4) контроль качества программного обеспечения;

В результате выполнения ВКР были разработаны программные средства управления журналами событий и хранилищами данных.

## **1 Анализ предметной области и планирование процесса разработки программных средств управления журналами событий и хранилищами исторических данных**

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

#### **1.1.1 Назначение и основные возможности платформы**

Разработка информационно-управляющей платформы (ИУП) ведется в рамках инициативной научно-исследовательской работы (НИР) «Методы и алгоритмы повышения эффективности оперативной обработки событий в системах активных правил», выполняемой на кафедре «Математическое обеспечение и применение ЭВМ». ИУП предназначена для создания, внедрения и эксплуатации информационно-управляющих систем (ИУС) на базе технологии Интернета вещей (IoT).

В законченном варианте ИУП может применяться для комплексной разработки безопасных, надежных, масштабируемых и высокопроизводительных приложений в режиме low-code/no-code без написания кода (либо с минимальным объемом кодирования) на всех этапах разработки, внедрения и сопровождения ИУС.

Одним из главных преимуществ Low-code и No-code платформы является скорость разработки. Благодаря визуальному интерфейсу и готовым модулям разработчики могут сосредоточиться на функционале и бизнес-логике, не тратя время на написание кода с нуля.

Low-code и No-code платформа должна обеспечить снижение расходов на разработку, так как требуется меньше времени и усилий по сравнению с традиционными методами разработки. Средствами платформы можно создавать MVP-решения (минимально жизнеспособный продукт) или прототипы, что позволит проверять идеи на жизнеспособность и доработать первоначальные продукты до нужного уровня.

Платформа даст больше возможностей для кастомизации и масштабирования, так как поддерживает использование кода для настройки специфических функций. Это позволит компаниям создавать более сложные и функциональные приложения, которые могут адаптироваться к растущим требованиям.

Low-code/No-code платформа позволит инженерам и техническим специалистам с хорошим знанием индустрии, но небольшим опытом программирования, создавать и разворачивать как рыночно-ориентированные продукты, так и сервисы внутренней цифровизации предприятий. Это обеспечит привлечение команд с высокой бизнес-экспертизой к процессу создания программного обеспечения.

ИУП позволит разработчикам в интерактивном режиме формировать метаданные, описывающие аспекты организации и поведения дискретно-событийных реагирующих ИУС (ДСРИУС) [1]:

- 1) инфраструктуру – IoT-устройства (датчики и устройства управления);
- 2) обрабатываемые данные – входные и хранимые данные;
- 3) логику мониторинга – правила обнаружения событий во входных данных, поступающих от устройств наблюдения (датчиков);
- 4) логику управления – активные правила, описывающие реакцию на обнаружение событий.

На основе метаданных создается и настраивается программное обеспечение, непосредственно отвечающее за обработку входных данных, обнаружение событий и реагирование на обнаруженные события.

Платформа позволяет выполнять потоковую обработку больших объемов входных данных с высокой скоростью, чтобы обеспечивать оперативное обнаружение событий, мониторинг и управление.

Для реагирования на обнаруженные события в ИУП используются активные правила (Event-Condition-Action, ECA).

Обнаруженные события, состояния физических и логических объектов управления, другие данные сохраняются в памяти. Поддерживается три уровня хранения данных:

- 1) оперативный – хранение данных непосредственно в потоках;
- 2) промежуточный – хранение данных в промежуточном хранилище в формате key-value, в качестве которого используется in-memory система управления данными;
- 3) долговременный – хранение данных в долговременном хранилище, в качестве которого используется NoSQL СУБД.

Оперативный уровень хранения используется в процессе обработки данных при их поступлении в систему, на этапе обнаружения событий и в ходе исполнения активных правил.

Промежуточный уровень хранения используется в основном пользовательскими приложениями для быстрого доступа к архивным данным, а также для восстановления оперативных данных в случае их потери и различных сбоев.

Долговременный уровень хранения используется для архивирования и журнализации накопленных данных. Долговременные данные в дальнейшем планируется использовать для интеллектуального анализа, разбора нештатных ситуаций и пр.

Глубина хранения данных настраивается для каждого проекта отдельно.

Основными функциями информационно-управляющей платформы являются [2]:

- 1) определение и актуализация инфраструктуры датчиков и устройств управления;
- 2) конструирование и управление событиями и активными правилами;
- 3) обработка потоков данных от устройств мониторинга, первичная обработка простых и обнаружение сложных событий;
- 4) исполнение активных правил по отношению к объектам управления;
- 5) журнализация и анализа событий и активных правил;

6) аудит событий и активных правил;

7) настройка.

В процессе дальнейших исследований и разработки функционал информационно-управляющей платформы будет развиваться и совершенствоваться.

### 1.1.2 Концептуальная модель информационно-управляющей системы на основе платформы

В основе организации и функционирования информационно-управляющей платформы лежит концептуальная модель, включающая классы информационных объектов и отношений между ними [3] (рисунок 1.1).

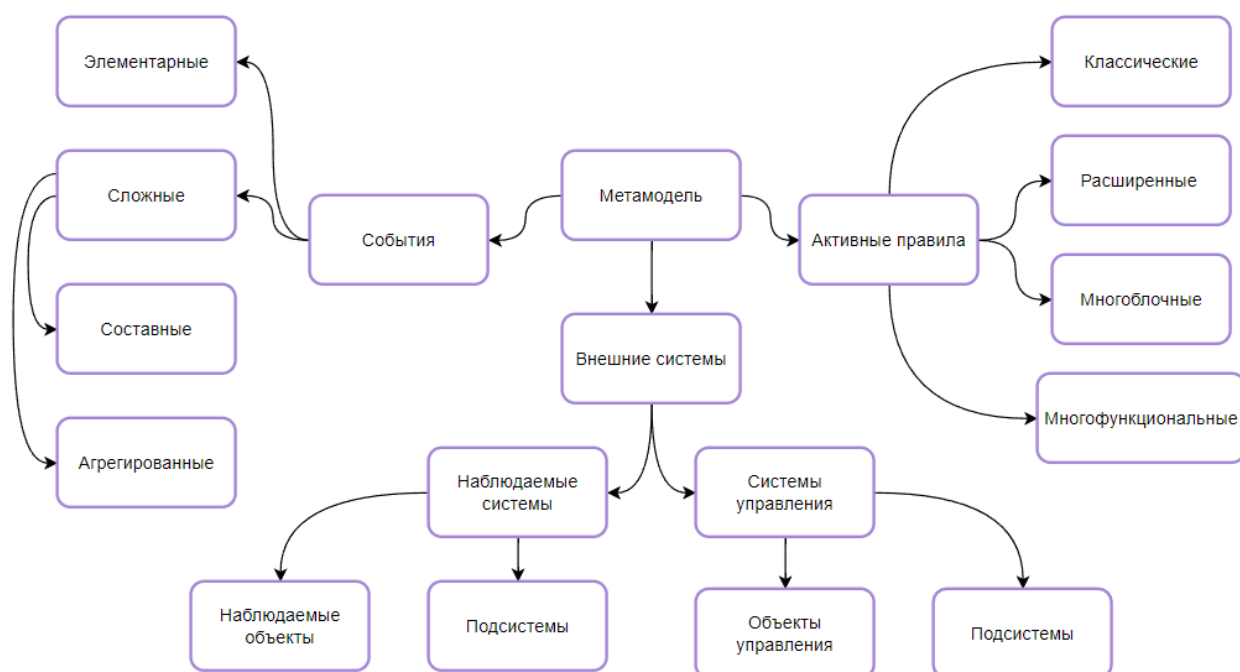


Рисунок 1.1 – Обобщенная концептуальная модель информационно-управляющей платформы

Концептуальная модель включает следующие классы объектов:

- 1) системы мониторинга и управления;
- 2) объекты мониторинга и объекты управления;
- 3) события, поступающие от объектов мониторинга;

4) активные правила для обработки событий и выработки управляющих воздействий;

5) другие виды объектов.

В таблице 1.1 представлен перечень основных классов объектов концептуальной модели и их краткое описание.

Таблица 1.1 – Перечень основных классов объектов концептуальной модели и их краткое описание

| Класс объектов                               | Описание                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Системы мониторинга и управления             | Это системы, предназначенные для наблюдения (мониторинга) за состоянием подсистем или объектов, а также для выработки решений и выполнения управляющих воздействий. Они действуют как связующее звено между объектами наблюдения и объектами управления, анализируя данные и реагируя на изменения среды.                                                                                        |
| Объекты мониторинга и объекты управления     | Объекты мониторинга – элементы системы, чьи параметры, состояния или поведение отслеживаются в реальном времени. Они предоставляют данные для анализа, например, датчики или системы с телеметрией. Объекты управления – элементы системы, на которые оказывают непосредственное влияние для изменения их состояния или поведения, например, исполнительные механизмы, устройства или программы. |
| События, поступающие от объектов мониторинга | Это данные или сигналы, которые указывают на изменения в состоянии объектов мониторинга. События включают информацию о возникновении аномалий, достижении пороговых значений, обновлении параметров, а также других значимых изменениях, требующих реакции.                                                                                                                                      |
| Активные правила                             | Это механизм или набор инструкций, определяющий, как система должна интерпретировать поступающие события и вырабатывать решения для управления. Активные правила обеспечивают автоматизацию процессов, включая анализ событий, выбор сценария реакции и выполнение управляющих действий.                                                                                                         |

В дальнейшем набор классов информационных объектов концептуальной модели может быть расширен и видоизменен.

#### 1.1.2.1 Внешние информационно–управляющие системы

Информационно-управляющая платформа может взаимодействовать с множеством внешних информационно-управляющих систем, которые могут подразделяться на системы мониторинга и системы управления.

Любая внешняя ИУС может иметь внутреннюю, иногда достаточно сложную структуру и состоять из множества подсистем. Каждая ИУС или ее

подсистема могут включать множество объектов мониторинга и объектов управления.

Объекты мониторинга являются источниками данных. Данные, поступающие от объектов мониторинга, интерпретируются как базовые (или первичные) события и образуют входной поток событий.

К объектам управления могут быть применены различные управляющие воздействия как реакция на возникающие события. Управляющие действия, применяемые к объектам управления, создают выходной поток действий.

### 1.1.2.2 События

Дискретно-событийные ИУС (ДСИУС) могут порождать, передавать и обрабатывать большое количество событий в единицу времени, поступающих от внешних источников в дискретные моменты времени, а также события, возникающие внутри системы в результате выполнения различных операций. События делятся на элементарные (простые) и комплексные (сложные), которые могут быть либо агрегированными, либо составными. (рисунок 1.2).



Рисунок 1.2 – Классификация событий

Элементарные события поступают от внешних наблюдаемых систем, объекты наблюдения которых отправляют различную информацию в определенные моменты времени. Элементарное событие описывается уникальным идентификатором события, контекстом события – набором

параметров, описывающих природу события, идентификатором источника события – объекта мониторинга, типом события, временем возникновения события.

Сложные события являются вычисляемыми событиями, которые фиксируются платформой в результате выполнения операций над элементарными и/или другими сложными событиями. Эти сложные события можно дополнительно классифицировать на агрегированные и составные события.

Агрегированное событие формируется на основе агрегирующего запроса к входному потоку элементарных событий.

Составное событие представляет собой комбинацию элементарных и/или сложных событий, с возможным использованием ограничений и логических операций. Контекст составного события определяется параметрами контекста входящих событий.

### **1.1.2.3      Активные правила**

В основе поведения информационно-управляющей платформы при реагировании на события, возникающие во внешних системах мониторинга и управления, лежит модель активных правил (Event-Condition-Action, ECA). Активное правило включает:

- 1) событийную часть, которая описывает тип события, инициирующего данное правило;
- 2) условную часть, представляющую собой логическое выражение;
- 3) активную часть, состоящую из операций над объектами управления.

Реакция на возникающие события вырабатывается в процессе исполнения активных правил, ассоциированных с типами событий. Активные правила можно классифицировать на классические и расширенные (рисунок 1.3). Расширенное активное правило может быть дополнительно классифицировано на альтернативное, многоблочное и многофункциональное правила.

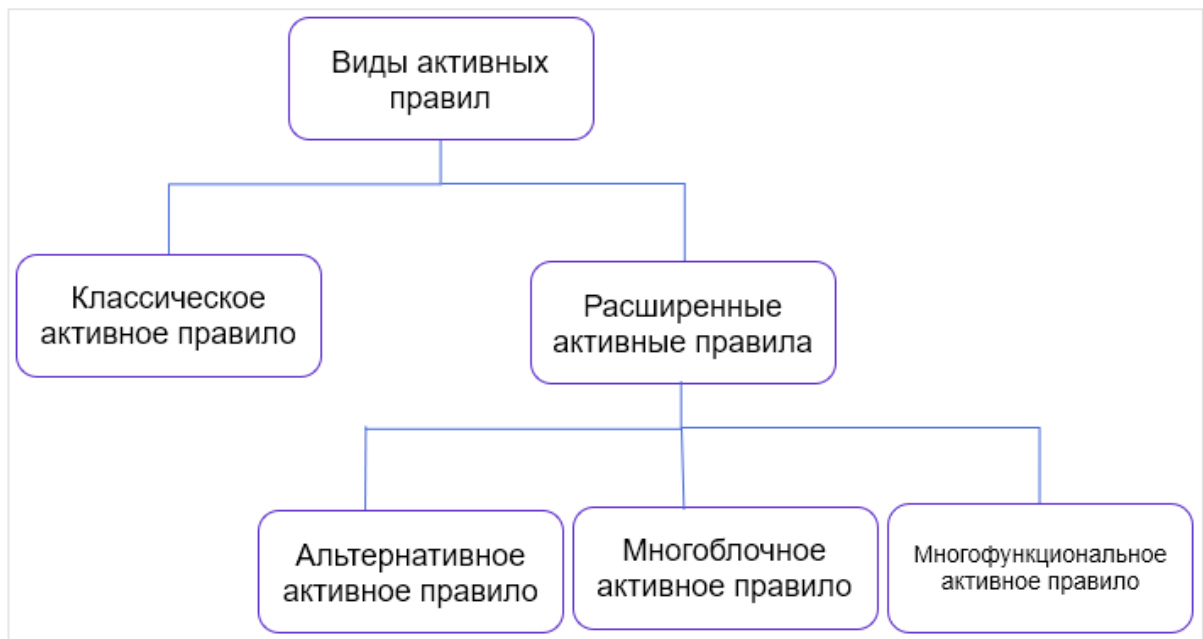


Рисунок 1.3 – Классификация активных правил

Классическое активное правило описывается кортежем:

$$r = \langle e, c, a \rangle,$$

где  $e$  – тип события, обрабатываемого системой;

$c$  – условие, логическое выражение над контекстом события, при истинности которого для экземпляра события выполняется исполнение активного правила;

$a$  – действие, набор операций, выполняемые над объектами управления в случае истинности условия  $c$ .

Появление расширенных активных правил обусловлено растущей сложностью бизнес-процессов и необходимостью повышения быстродействия систем, вследствие минимизации множества активных правил.

Альтернативное активное правило представляется как:

$$r = \langle e, c, \{a_{base}, a_{alt}\} \rangle,$$

где  $a_{base}$  – действие, выполняемое в случае истинности условия;

$a_{alt}$  – альтернативное действие, выполняемое в случае, если условие ложно.

Для описания реакции на сложное событие с множеством возможных результирующих событий применяется многоблочные активные правила, для

которых определены несколько вариантов обнаружения событий. Такие правила представляются как:

$$r = \langle \{e_1, c_1, a_1\}, \dots, \{e_n, c_n, a_n\} \rangle,$$

где  $e_1, \dots, e_n$  – множество результирующих событий;

$c_1, \dots, c_n$  – множество условий, соответствующих множеству результирующих событий;

$a_1, \dots, a_n$  – множество действий, соответствующих множеству условий.

Многофункциональное активное правило применяется в случае, когда в качестве реакции на возникновения события и истинности условия необходимо выполнить множество реагирующих воздействий. Многофункциональное правило представляется кортежем:

$$r = \langle e, c, A \rangle,$$

где  $A = \{a_1, \dots, a_k\}$  – множество экземпляров результирующих воздействий, применяемых к управляемым объектам.

### 1.1.3 Обобщенная программная архитектура платформы

Архитектурно информационно-управляющая платформа состоит из нескольких достаточно крупных программных компонент, которые взаимодействуют между собой по протоколу RESTful API (рисунок 1.4).

Программные средства управления метаданными реализованы в виде отдельного портала, который позволяет создавать логические элементы информационно-управляющих систем и сохранять их описание в центральном репозитории метаданных. Метаданные из центрального репозитория тиражируются в другие компоненты платформы и размещаются в локальных репозиториях.

Центральный репозиторий представляет собой глобальное хранилище метаданных объектов информационно-управляющих систем, разворачиваемых на платформе.

REST-сервер в составе программных средств управления метаданными реализует операции ведения и верификации метаданных в центральном

репозитории, также синхронизацию и тиражирование метаданных в локальные репозитории других компонент платформы.

Сервис потоковой обработки событий и исполнения активных правил предназначен для обработки данных, поступающих от IoT-устройств, обнаружения заданных событий и исполнения активных правил в ответ на возникновение событий. Сервис построен на основе микросервисной и событийно-ориентированной архитектуры.

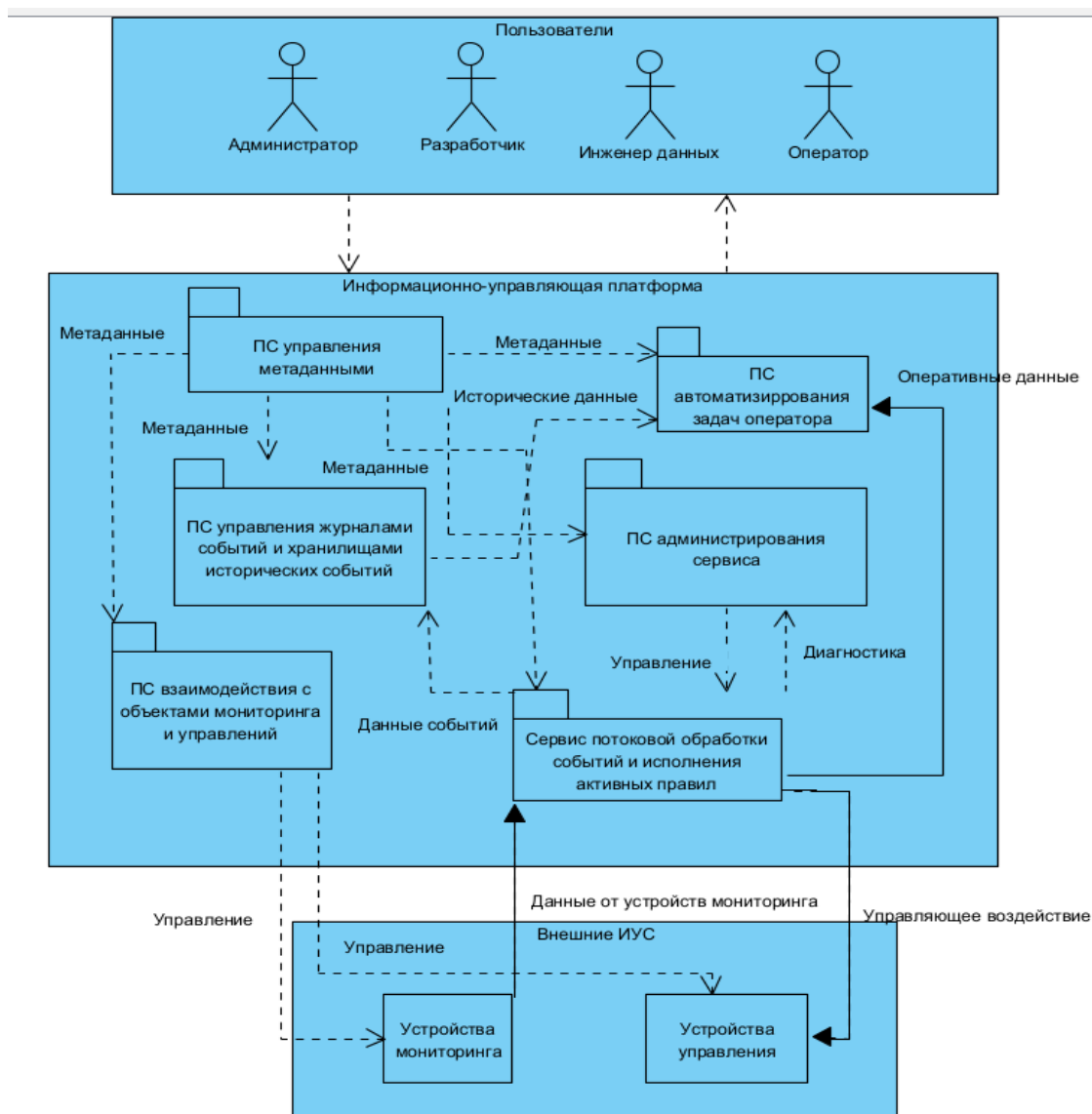


Рисунок 1.4 – Обобщенная программная архитектура информационно-управляющей платформы

Микросервисная архитектура представляет собой подход к разработке единого приложения в виде набора небольших сервисов, работающих в

отдельных процессах, и взаимодействующих с применением упрощенных механизмов. Событийно-ориентированная архитектура состоит из независимых и специализированных компонентов обработки событий, которые асинхронно получают и обрабатывают события. Использование микросервисной и событийно-ориентированной архитектуры предполагает независимое масштабирование сервисов, высокую отказоустойчивость, ускоренную разработку.

Программные средства взаимодействия с объектами мониторинга и управления реализуют основные операции с IoT-устройствами – создание и развертывание устройств, контроль работоспособности, моделирование их поведения. В настоящее время поддерживается взаимодействие по протоколу MQTT через облачный сервис Yandex IoT Cloud. Для непосредственного управления IoT-устройствами и моделирования их поведения применяется платформа NODE-Red.

Программные средства управления журналами событий и хранилищами исторических данных предназначены для организации оперативного, промежуточного и долговременного хранения данных об обнаруженных событиях, состояниях физических и логических объектов управления и другие. Сохраненные и накопленные данные доступны для просмотра и анализа.

В состав платформы входят пользовательские приложения с графическим интерфейсом, ориентированные на различные классы пользователей. Для разработчиков предназначен портал управления метаданными, для операторов и диспетчеров – приложение для автоматизации задач оператора, для администратора устройств – панель управления устройствами, для инженера данных – панель управления данными.

## 1.2 Журнализация и хранение данных в процессе функционирования информационно-управляющих систем

### 1.2.1 Понятие и классы логических переменных как объектов хранимых данных

Логическая переменная в контексте функционирования информационно-управляющих систем представляет собой абстрактный объект, предназначенный для фиксации определённого состояния или характеристики наблюдаемого объекта (устройства, процесса, параметра и т. д.).

Были выделены следующие свойства логических переменных:

1) уникальный идентификатор (Id) – каждая логическая переменная должна однозначно интерпретироваться;

2) тип переменной (TypeVariableId) – переменная может храниться в различных вариантах (атом, структура или ряд);

3) название (Name) – уникальное наименование переменной, по которой возможен доступ к таблице данных;

4) описание (Description) – характеристика, поясняющая суть переменной и её назначение;

5) запрос (Query) – запрос, предназначенный для создания таблицы.

Благодаря своей гибкой структуре переменные могут относиться к разным классам информационных объектов. Примеры классов представлены на таблице 1.2.

Таблица 1.2 – Примеры классов информационных объектов

| Тип данных              | Описание                                                                    | Пример                                        |
|-------------------------|-----------------------------------------------------------------------------|-----------------------------------------------|
| Данные наблюдений       | Фактические значения, поступившие от датчиков или измерительных устройств   | Температура окружающей среды                  |
| События                 | Логические переменные, формирующиеся путем анализа первичных показаний      | Повышение порогового уровня температуры       |
| Агрегированное значение | Промежуточные или конечные выводы, рассчитанные на основании входных данных | Среднее значение потребления энергии за месяц |

## Продолжение таблицы 1.2

|                       |                                                                       |                                         |
|-----------------------|-----------------------------------------------------------------------|-----------------------------------------|
| Управляющие команды   | Сигналы, передаваемые управляемым устройствам для изменения состояния | Изменение скорости вращения вентилятора |
| Действия пользователя | Информация о действиях пользователей внутри системы                   | Авторизация пользователя                |

При журнализации и хранении данных структура логической переменной позволяет без особых усилий сохранять данные и получать доступ к ним.

### 1.2.2 Источники и потребители данных логических переменных

Источники данных отвечают за предоставление первоначальных сведений, на основе которых формируются логические переменные. Они делятся на несколько категорий:

- 1) датчики и приборы;
- 2) логические каналы и журналы;
- 3) интерфейсы прикладных программ.

К первой группе относятся: физические устройства и сенсоры, собирающие реальные физические данные (температура, влажность, вибрация и т. д.).

Ко второй: журналы операционной системы, логи приложений и т. д.

К последней: интерфейсы приложений, используемые для сбора данных с облачных сервисов, web-служб и прочих онлайн-ресурсов, а также графические интерфейсы (дашборды, панели мониторинга).

Потребители данных используют логические переменные для выполнения задач анализа, контроля и принятия решений. Основными классами потребителей являются: аналитические системы, системы управления, а также пользовательские интерфейсы.

Грамотная организация и поддержка взаимодействия между классами источников и потребителей повышают эффективность управления, безопасность и стабильность системы.

### 1.2.3 Бизнес-процессы управления хранимыми данными логических переменных

Для описания бизнес-процессов управления хранимыми данными логических переменных была выбрана диаграмма деятельности. Диаграмма деятельности – это технология, позволяющая описывать логику процедур, бизнес-процессы и потоки работ. Во многих случаях они напоминают блок-схемы, но принципиальная разница между диаграммами деятельности и нотацией блок-схем заключается в том, что первые поддерживают параллельные процессы [5].

Диаграмма для управления хранимыми данными логических переменных представлена на рисунке 1.5.

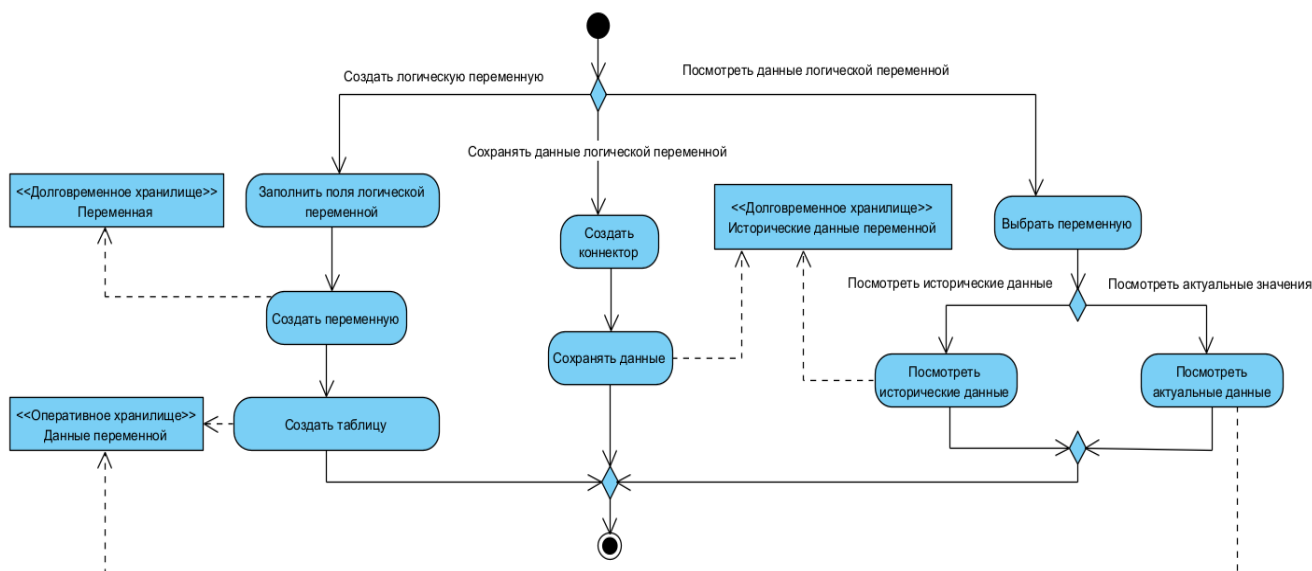


Рисунок 1.5 – Диаграмма деятельности управления хранимыми данными логических переменных

На диаграмме изображены следующие бизнес-процессы:

- 1) «Создать логическую переменную»;
- 2) «Сохранять данные логической переменной»;
- 3) «Посмотреть данные логической переменной».

В бизнес-процессе «Создать логическую переменную» происходит следующее: пользователь заполняет поля логической переменной и она

сохраняться в долговременное хранилище, затем на основе переменной создается таблица в оперативном хранилище.

При бизнес-процессе «Сохранять данные логической переменной» сначала создается коннектор, с помощью которого данные сохраняются в долговременном хранилище.

В «Посмотреть данные исторической переменной» пользователь выбирает посмотреть исторические данные или посмотреть актуальные значения.

Единственным участником бизнес-процессов является пользователь «Инженер данных».

### **1.3 Сравнительный анализ современных решений для хранения данных в информационно-управляющих платформах**

#### **1.3.1 Обоснование сравнительного анализа**

Сравнительный анализ – это исследование, при котором оцениваются и сравниваются характеристики программ по критериям.

Основными целями сравнительного анализа являются:

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

При проведении сравнительного анализа современных решений для хранения данных в информационно-управляющих платформах требуется:

- 1) выбрать платформы для анализа;
- 2) определить критерии сравнения;
- 3) изучить выбранные платформы;
- 4) сделать выводы по анализу.

Для сравнительного анализа были выбраны следующие платформы: OrangeScada, MasterSCADA и Aggregate.

На таблице 1.3 представлены некоторые характеристики анализируемых платформ.

Таблица 1.3 – Характеристики анализируемых платформ

|                              |                                         |                                           |                                            |
|------------------------------|-----------------------------------------|-------------------------------------------|--------------------------------------------|
| Параметр                     | OrangeScada                             | MasterSCADA                               | Aggregate                                  |
| Страна разработки            | Россия                                  | Россия                                    | Россия                                     |
| Архитектура                  | Распределенный клиент-сервер            | Клиент-сервер                             | Клиент-сервер, web-клиенты                 |
| Цена и лицензия              | Коммерческая, лицензия за точку доступа | Цена за лицензию, пакетные предложения    | Лицензия за устройство или модуль          |
| Документирование и поддержка | Детальная документация, курсы обучения  | Руководства пользователя, форумы, саппорт | Официальная документация, форум сообщества |

Для анализа данных платформ были выбраны следующие критерии, приведенные на таблице 1.4.

Таблица 1.4 – Критерии сравнения

| Критерий                                       | Характеристика                                                                       |
|------------------------------------------------|--------------------------------------------------------------------------------------|
| Хранение данных в виде объектов или переменных | Возможность хранить данные в виде переменных                                         |
| Хранение исторических данных                   | Возможность хранить исторические данные                                              |
| Обработка исторических данных                  | Возможность обрабатывать большие объемы данных                                       |
| Получение актуального значения                 | Получение значения в реальном времени                                                |
| Использование in-memory СУБД                   | Быстрота доступа к данным                                                            |
| Масштабируемость                               | Способность системы адаптироваться к увеличению объема данных                        |
| Интеграция                                     | Способность системы взаимодействовать с внешними источниками данных или приложениями |

Общая шкала оценки критериев представлена на таблице 1.5.

Таблица 1.5 – Общая шкала оценки критериев

| Оценка | Интерпретация                                           |
|--------|---------------------------------------------------------|
| 1      | Низкий уровень реализации, серьезные ограничения        |
| 2      | Средний уровень, существуют компромиссы или ограничения |
| 3      | Высокое качество реализации, полный охват.              |

Таким образом, для сравнительного анализа были выбраны аналоги платформ и критерии сравнения.

### 1.3.2 Обзор программных платформ для анализа

OrangeScada – программное обеспечение, работающее на операционных системах Windows, предназначенное для разработки и внедрения систем диспетчеризации различного инженерного оборудования. Ее можно

использовать в качестве классической SCADA-системы, когда все компоненты: сервер и клиент располагаются локально на одном компьютере, так и в качестве web-ориентированной SCADA с возможностью удаленного доступа к данным через локальную сеть или Интернет.

Панели управления OrangeScada представлены на рисунке 1.6.

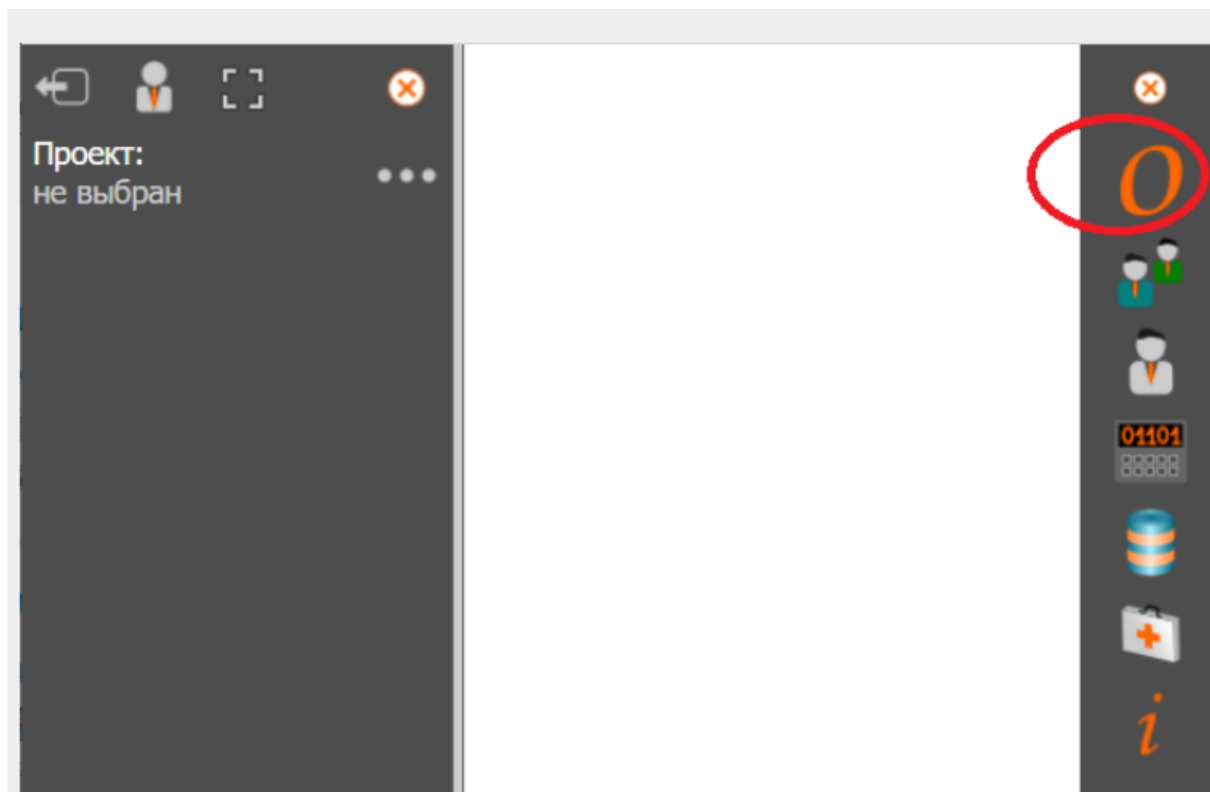


Рисунок 1.6 – Панели управления OrangeScada

На рисунки изображены:

- 1) панель слева – навигационная, для быстрого перехода между документами проекта;
- 2) панель справа содержит инструменты для разработки [6].

MasterScada – платформа, в которой есть инструменты по созданию крупных распределенных систем с возможностью использования технологий Интернета, повышены удобство и гибкость, расширены возможности использования различных аппаратных платформ и операционных систем, увеличено число поддерживаемых уровней систем управления и реализована миграция функционала между уровнями.

Графический интерфейс оператора создается в редакторе HMI, который входит в состав среды разработки MasterSCADA 4D, в режиме исполнения созданные окна открываются в клиенте визуализации. Графический интерфейс редактора представлен на рисунке 1.7.

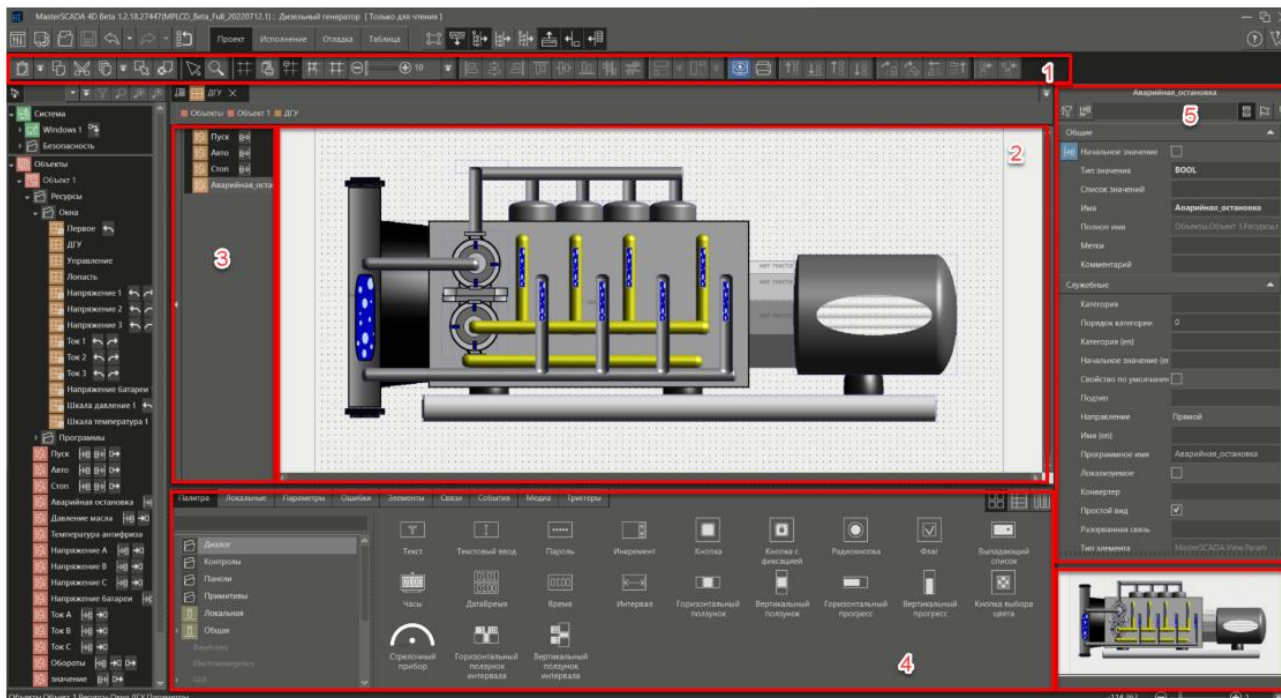


Рисунок 1.7 – Графический редактор HMI

Интерфейс редактора HMI состоит из следующих областей:

- 1) панель инструментов;
- 2) рабочая область редактора HMI – в этой рабочей области создается внешний вид окна (мнемосхемы), в котором оно будет отображаться в клиенте визуализации в режиме исполнения ;
- 3) клеммники;
- 4) легенда;
- 5) панель свойств;
- 6) миникарта – полное уменьшенное изображение окна (мнемосхемы), на котором прямоугольной рамкой выделена зона, отображаемая в рабочей области редактора HMI.

Путем перемещения рамки по миникарте выбирается требуемая для редактирования зона. Используется для работы с мнемосхемами, размеры которых превышают размеры рабочей области редактора HMI) [7].

AggreGate – это платформа low-code разработки, используемая для создания и внедрения приложений, продуктов, решений, сервисов IoT и цифрового предприятия, а также для работы с ними.

За создание пользовательского интерфейса отвечает Конструктор Web UI. Конструктор помогает создавать веб-интерфейсы решений, сервисов и продуктов на базе платформы Aggregate [8].

Пример панели управления Aggregate, созданной с помощью данного инструмента, представлена на рисунке 1.8.

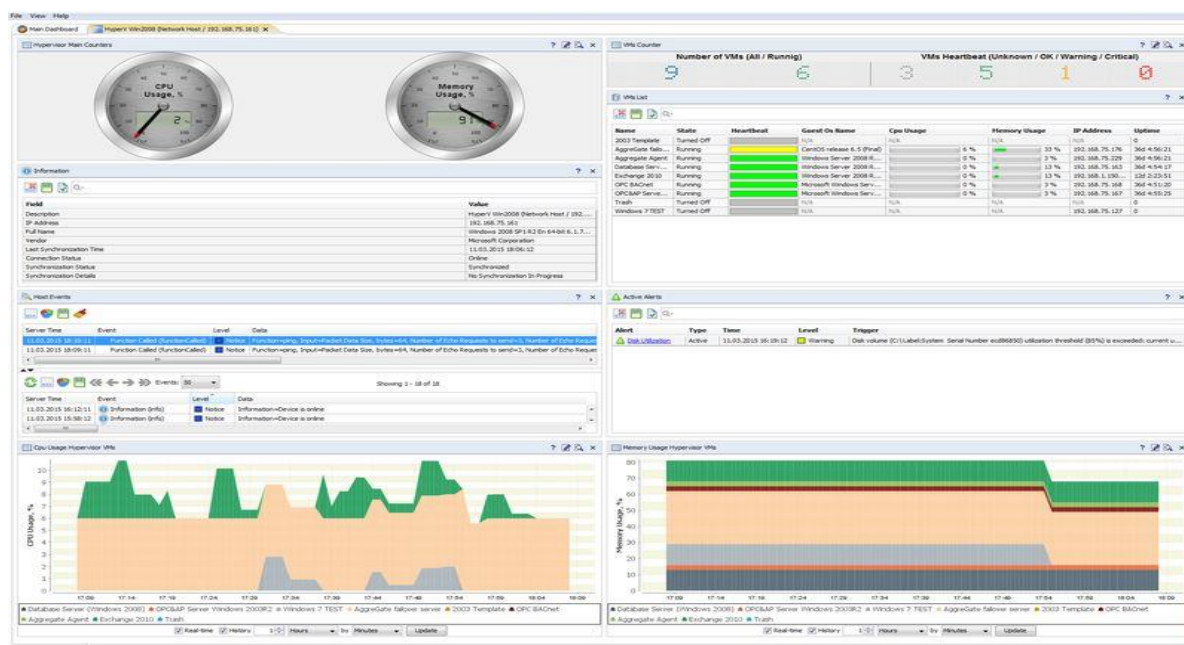


Рисунок 1.8 – Панель управления Aggregate

Таким образом, в целях сравнительного анализа был выполнен обзор аналогичных платформ и рассмотрены их способы создания пользовательского интерфейса.

### 1.3.3 Результаты сравнительного анализа и выводы

Результаты сравнительного анализа представлены на таблице 1.6.

Таблица 1.6 – Результаты сравнительного анализа

| № п/п | Критерий                                       | OrangeScada | MasterScada | Aggregate |
|-------|------------------------------------------------|-------------|-------------|-----------|
| 1     | Хранение данных в виде объектов или переменных | 3           | 2           | 3         |
| 2     | Хранение исторических данных                   | 3           | 3           | 3         |
| 3     | Обработка исторических данных                  | 3           | 1           | 2         |
| 4     | Получение актуального значения                 | 3           | 2           | 3         |
| 5     | Использование in-memory СУБД                   | 2           | 1           | 2         |
| 6     | Масштабируемость                               | 3           | 2           | 2         |
| 7     | Интеграция                                     | 3           | 2           | 3         |

По большинству важных критериев (хранение данных, работа с историческими данными, получение актуального значения, масштабируемость) OrangeScada уверенно опережает остальных участников. MasterSCADA заметно отстаёт по возможностям обработки данных и масштабированию, а Aggregate немного уступает OrangeSCADA, особенно в вопросах обработки больших объемов данных и производительности.

Таким образом, можно сделать вывод о том, что во многих информационно-управляющих системах хранение данных происходит в виде объектов или переменных, что говорит о полезности и удобности данного подхода.

Для обеспечения базового решения необходимо реализовать стандартные функции:

- 1) сохранение данных в виде переменных;
- 2) хранение исторических данных;
- 3) обработка исторических данных;
- 4) получение актуального значения.

В качестве дополнительных функций необходимого для обеспечения конкурентноспособности необходимо использование in-memory СУБД, обеспечение масштабируемости и высокого уровня интеграции.

## **1.4 Планирование процесса разработки программных средств**

### **1.4.1 Цели разработки и назначение программных средств**

Разрабатываемые программные средства управления журналами событий и хранилищами исторических данных должны обеспечивать пользователя набором удобных инструментов для управления данными.

Основные цели разработки:

- 1) реализация хранения данных в виде переменных;
- 2) разработка хранения и передачи данных от потоков в различные уровни хранилищ данных;
- 3) интеграция сервиса со сторонними клиентскими приложения.

Кроме того, после окончания основного процесса разработки и публикации сервиса управления, требуется поддерживать его функционал, проверять на неисправности, проводить анализ работы сервиса с помощью сбора статистики, а также внедрять новый функционал при необходимости, для упрощения работы сервиса.

### **1.4.2 План-график разработки и планирование бюджета**

Планирование процесса разработки включает формирование плана выполнения работ, который отражает порядок действий при реализации проекта с учетом необходимых ресурсов. Основной задачей планирования является определение целей и методов их достижения.

Для демонстрации плана разработки подходит диаграмма Ганта, отображающая длительность и сроки выполнения этапов разработки на шкале времени, а также требуемые для их реализации ресурсы. В качестве инструмента управления проектом была выбрана программа Microsoft Project. Дальнейший расчет будет производиться для ПС взаимодействия с объектами мониторинга и управления.

В таблице 1.7 приведены результаты определения задач проекта, типов, длительности выполнения и задачи, предшествующие им.

Таблица 1.7 – Задачи проекта

| №  | Задача                                                                                              | Тип    | Длительность | Предшественники |
|----|-----------------------------------------------------------------------------------------------------|--------|--------------|-----------------|
| 1  | 2                                                                                                   | 3      | 4            | 5               |
| 1  | Начало проекта                                                                                      | Веха   | 1            |                 |
| 2  | Анализ предметной области и постановка задачи на разработку                                         | Фаза   | 10           |                 |
| 3  | Анализ назначения и возможностей ПС                                                                 | Задача | 1            | 1               |
| 4  | Анализ архитектуры ПС                                                                               | Задача | 1            | 3               |
| 5  | Анализ классификации логических переменных как объектов хранимых данных                             | Задача | 1            | 4               |
| 6  | Анализ источников и потребителей данных логических переменных                                       | Задача | 1            | 5               |
| 7  | Анализ бизнес-процессов управления хранимыми данными логических переменных                          | Задача | 2            | 6               |
| 8  | Сравнительный анализ аналогов                                                                       | Задача | 2            | 7               |
| 9  | Постановка задач на разработку ПС                                                                   | Задача | 2            | 8               |
| 10 | Анализ предметной области и постановка задачи на разработку завершены                               | Веха   | 0            | 9               |
| 11 | Разработка требований к ПС управления журналами событий и хранилищами исторических данных           | Фаза   | 6            |                 |
| 12 | Анализ функциональных требований                                                                    | Задача | 1            | 9               |
| 13 | Анализ нефункциональных требований                                                                  | Задача | 1            | 12              |
| 14 | Анализ требований к пользовательскому интерфейсу клиентского приложения                             | Задача | 1            | 13              |
| 15 | Анализ требования к аппаратному обеспечению                                                         | Задача | 1            | 14              |
| 16 | Анализ требований к системному программному обеспечению                                             | Задача | 1            | 15              |
| 17 | Анализ требований к средствам разработки                                                            | Задача | 1            | 16              |
| 18 | Разработка требований к ПС управления журналами событий и хранилищами исторических данных завершена | Веха   | 0            | 17              |
| 19 | Разработка ПС управления журналами событий и хранилищами исторических данных                        | Фаза   | 23           |                 |
| 20 | Разработка архитектуры развертывания программных средств                                            | Задача | 4            | 18              |
| 21 | Разработка серверной части ПС                                                                       | Задача | 5            | 19              |
| 22 | Разработка клиентской части ПС                                                                      | Задача | 5            | 20              |

Продолжение таблицы 1.7

| 1  | 2                                                                                   | 3      | 4  | 5  |
|----|-------------------------------------------------------------------------------------|--------|----|----|
| 23 | Разработка ПС для взаимодействия с приложениями информационно-управляющей платформы | Задача | 5  | 21 |
| 24 | Разработка ПС завершена                                                             | Веха   | 0  | 22 |
| 25 | Контроль качества ПС                                                                | Фаза   | 10 |    |
| 26 | Выбор методов тестирования                                                          | Задача | 2  | 24 |
| 27 | Тестирование ПС                                                                     | Задача | 5  | 25 |
| 28 | Оценка качества программного кода                                                   | Задача | 3  | 26 |
| 29 | Контроль качества ПС завершен                                                       | Веха   | 0  | 27 |
| 30 | Завершение работы над проектом                                                      | Фаза   | 5  |    |
| 31 | Оформление документации                                                             | Задача | 4  | 28 |
| 32 | Сдача проекта                                                                       | Задача | 1  | 29 |
| 33 | Проект завершен                                                                     | Веха   | 0  | 30 |

После составления плана выполнения работ необходимо оценить ресурсы проекта и распределить их по задачам. Информация об имеющихся в проекте ресурсах и их распределение представлены в таблицах 1.8 и 1.9 соответственно.

Таблица 1.8 – Ресурсы проекта

| Название ресурса | Тип ресурса  | Затраты            |                     | Начисление       |
|------------------|--------------|--------------------|---------------------|------------------|
|                  |              | Стандартная ставка | Ставка сверхурочных |                  |
| Программист      | Трудовой     | 600 р/ч            | 700 р/ч             | Пропорциональное |
| Аналитик         | Трудовой     | 500 р/ч            | 800 р/ч             | Пропорциональное |
| Тестировщик      | Трудовой     | 500 р/ч            | 700 р/ч             | Пропорциональное |
| Компьютер        | Материальный |                    |                     | Пропорциональное |
| Интернет         | Затраты      |                    |                     | Пропорциональное |
| Печать           | Затраты      |                    |                     | Пропорциональное |

Таблица 1.9 – Распределение ресурсов по задачам

| Название задачи                                                         | Ресурсы            | Единицы (затраты) |
|-------------------------------------------------------------------------|--------------------|-------------------|
| 1                                                                       | 2                  | 3                 |
| Начало проекта                                                          | Компьютер          | 1                 |
| Анализ предметной области и постановка задачи на разработку             |                    |                   |
| Анализ назначения и возможностей ПС                                     | Аналитик           | 100%              |
| Анализ архитектуры ПС                                                   | Аналитик           | 100%              |
| Анализ классификации логических переменных как объектов хранимых данных | Аналитик, Интернет | 100%              |

Продолжение таблицы 1.9

| 1                                                                                                   | 2                     | 3    |
|-----------------------------------------------------------------------------------------------------|-----------------------|------|
| Анализ источников и потребителей данных логических переменных                                       | Аналитик, Интернет    | 100% |
| Анализ бизнес-процессов управления хранимыми данными логических переменных                          | Аналитик              | 100% |
| Сравнительный анализ аналогов                                                                       | Аналитик, Интернет    | 100% |
| Постановка задач на разработку ПС                                                                   | Аналитик, программист | 100% |
| Анализ предметной области и постановка задачи на разработку завершены                               |                       |      |
| Разработка требований к ПС управления журналами событий и хранилищами исторических данных           |                       |      |
| Анализ функциональных требований                                                                    | Аналитик              | 100% |
| Анализ нефункциональных требований                                                                  | Аналитик              | 100% |
| Анализ требований к пользовательскому интерфейсу клиентского приложения                             | Аналитик              | 100% |
| Анализ требования к аппаратному обеспечению                                                         | Аналитик, программист | 100% |
| Анализ требований к системному программному обеспечению                                             | Аналитик, программист | 100% |
| Анализ требований к средствам разработки                                                            | Аналитик, программист | 100% |
| Разработка требований к ПС управления журналами событий и хранилищами исторических данных завершена |                       |      |
| Разработка архитектуры развертывания программных средств                                            | Программист           | 100% |
| Разработка ПС серверной части                                                                       | Программист           | 100% |
| Разработка ПС клиентской части                                                                      | Программист           | 100% |
| Разработка ПС для взаимодействия с приложениями информационно-управляющей платформы                 | Программист           | 100% |
| Разработка ПС завершена                                                                             |                       |      |
| Контроль качества ПС                                                                                |                       |      |
| Выбор методов тестирования                                                                          | Тестировщик           | 100% |
| Выбор методов тестирования                                                                          | Тестировщик           | 100% |
| Тестирование ПС                                                                                     | Тестировщик           | 100% |
| Оценка качества программного кода                                                                   | Тестировщик           | 100% |
| Контроль качества ПС                                                                                |                       |      |
| Завершение работы над проектом                                                                      |                       |      |
| Оформление документации                                                                             | Аналитик, печать      | 100% |
| Сдача проекта                                                                                       |                       |      |
| Проект завершен                                                                                     |                       |      |

На рисунке 1.10 показана диаграмма Ганта, которая наглядно отображает план выполнения задач проекта и распределенные для них ресурсы.

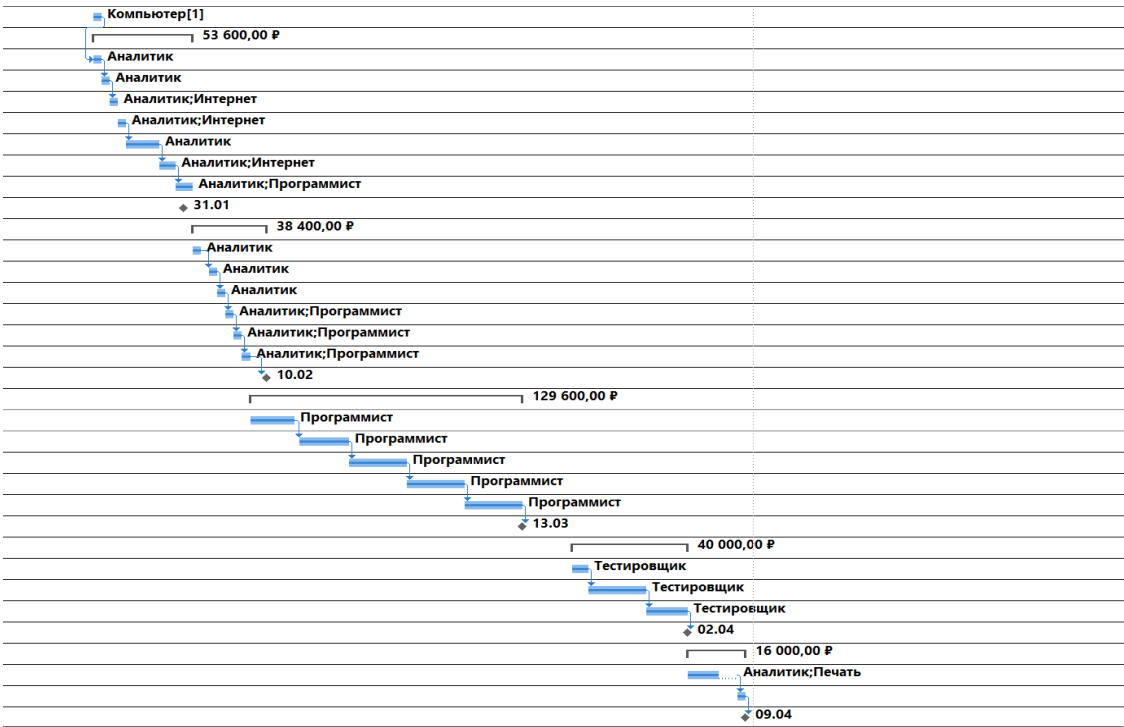


Рисунок 1.9 – Диаграмма Ганта

В Microsoft Project можно сформировать различные отчеты, среди которых наиболее распространенными являются обзор о затратах ресурсов и обзор проекта. Обзор о затратах ресурсов отражает стоимость ресурсов для создания проекта. Обзор по проекту показывает прогресс и длительность задач по проекту. На рисунках 1.11 и 1.12 представлены обзор о затратах ресурсов и обзор по проекту соответственно.

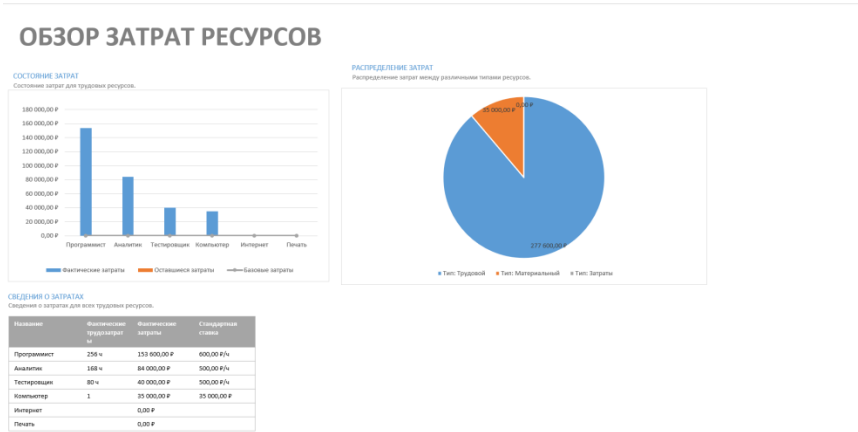


Рисунок 1.10 – Обзор затрат ресурсов проекта

## ОБЗОР ПРОЕКТА

### % ЗАВЕРШЕНИЯ

Состояние всех задач верхнего уровня. Чтобы просмотреть состояние вложенных задач, щелкните диаграмму и обновите уровень структуры в списке полей.

ПН 20.01.25 - СР 09.04.25

% ЗАВЕРШЕНИЯ

100%

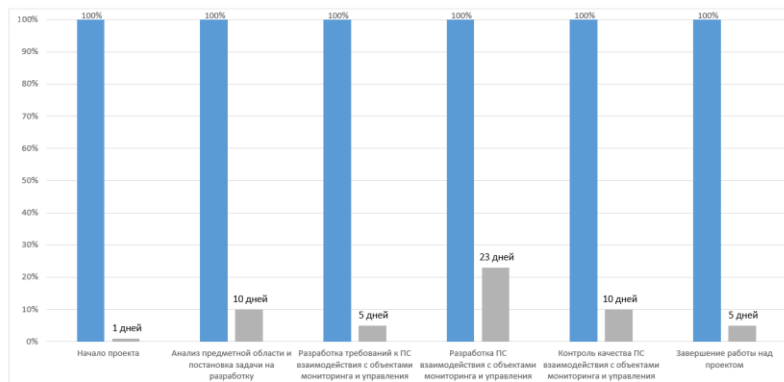


Рисунок 1.11 – Обзор проекта

Построенный обзор о затратах ресурсов показал, что общая стоимость разработки ПС взаимодействия с объектами мониторинга и управления составила 312600 рублей. Исходя из обзора по проекту, следует, что в ходе реализации проекта было выполнено 34 задачи. Проект был выполнен на 100% за 54 дня.

## 2 Анализ требований и постановка задачи на разработку программных средств взаимодействия с объектами мониторинга и управления

### 2.1 Анализ функциональных требований

Функциональные требования описывают прикладные требования к системе, то есть то, что система должна делать. Они определяют все функции, которые программа или продукт должны выполнять в зависимости от того, какие действия совершает пользователь [9].

Основные виды функциональных требования представлены на таблице 2.1.

Таблица 2.1 – Типы функциональных требований

| № | Тип требований                       | Описание                                                                      |
|---|--------------------------------------|-------------------------------------------------------------------------------|
| 1 | Требования к обработке данных        | Ввод/вывод данных, их хранение, обработка и передача.                         |
| 2 | Требования к интерфейсу пользователя | Графический интерфейс, средства ввода-вывода, обратная связь с пользователем. |
| 3 | Требования к безопасности            | Меры защиты данных, аутентификация, контроль доступа.                         |
| 4 | Требования к производительности      | Скорость обработки запросов, нагрузка на систему, масштабируемость.           |
| 5 | Требования к интеграции              | Взаимодействие с внешними приложениями и сервисами.                           |

Диаграмма вариантов использования – это одна из многих диаграмм, используемых в методологии UML. Она применяется для анализа функциональных требований системы и описывает взаимодействие пользователей (или внешних объектов) с системой посредством сценариев действий («вариантов использования»).

Результаты анализа функциональных требований представлены в виде диаграммы вариантов использования на рисунке 2.1.

Диаграмма вариантов использования была сделана с помощью программы Visual Paradigm.

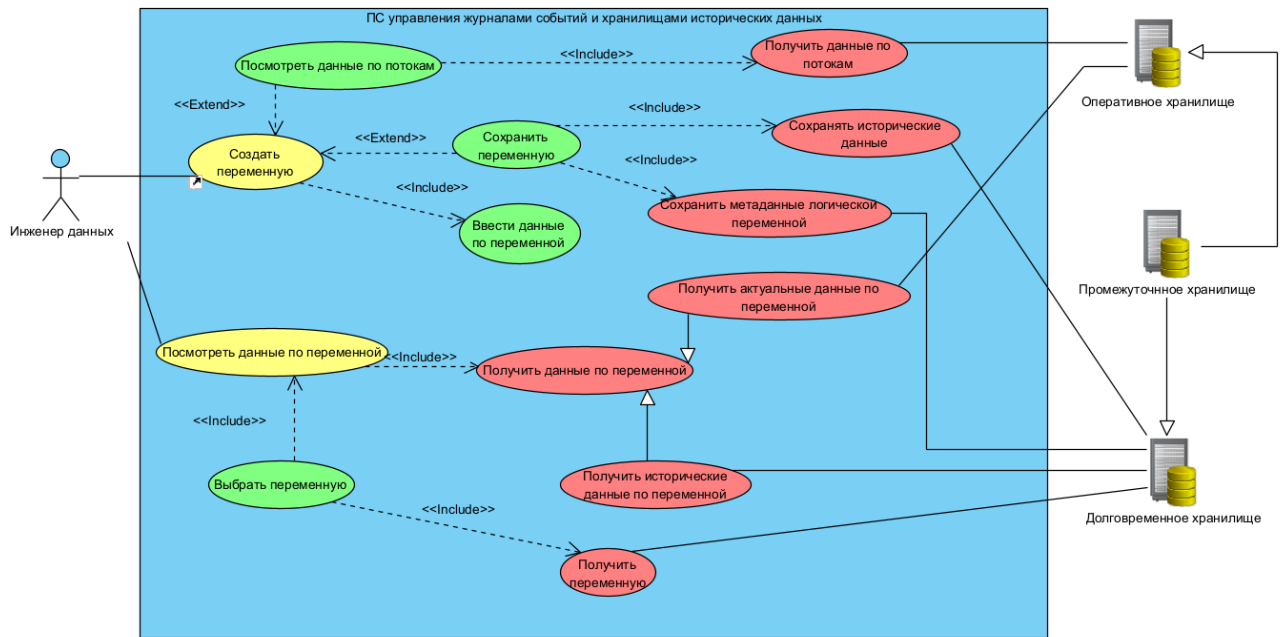


Рисунок 2.1 – Диаграмма вариантов использования

На диаграмме вариантов использования изображены следующие элементы:

- 1) актер – представляет собой некую роль, которую пользователь играет по отношению к системе;
- 2) вариант использования или прецедент (use case) – это единица поведения системы, результат которой значим для некоторого заинтересованного лица;
- 3) отношение – связь между вариантами использования или между актерами [10].

На таблице 2.2 представлена спецификация актеров диаграммы вариантов использования.

Таблица 2.2 – Спецификация актеров диаграммы вариантов использования

| Актер                    | Роль          | Описание                                                        |
|--------------------------|---------------|-----------------------------------------------------------------|
| Инженер данных           | Пользователь  | Взаимодействие с системой в целях использования хранилищ данных |
| Оперативное хранилище    | Объект данных | Предоставляет актуальные данные системе                         |
| Промежуточное хранилище  | Объект данных | Предоставляет данные, хранящиеся в оперативной памяти, системе  |
| Долговременное хранилище | Объект данных | Предоставляет исторические данные системе                       |

Варианты использования были поделены по цвету:

- 1) желтого цвета – уровень бизнес-процессов;
- 2) зеленого цвета – уровень бизнес-операций;
- 3) красного цвета – уровень операций над данными.

На таблице 2.3 представлена краткая спецификация вариантов использования.

Таблица 2.3 – Краткая спецификация вариантов использования

| № п/п | Вариант использования                      | Уровень            | Краткие спецификации                                                                                               |
|-------|--------------------------------------------|--------------------|--------------------------------------------------------------------------------------------------------------------|
| 1     | Создать переменную                         | Бизнес-процесс     | Инженер данных создает переменную                                                                                  |
| 2     | Посмотреть данные по переменной            | Бизнес-процесс     | Инженер данных смотрит данные по переменным                                                                        |
| 3     | Ввести данные по переменной                | Бизнес-операция    | Инженер данных вводит данные для создания переменной                                                               |
| 4     | Сохранить переменную                       | Бизнес-операция    | Инженер данных сохраняет метаданные логической переменной и сохраняет данные переменной в Долговременное хранилище |
| 5     | Посмотреть данные по потокам               | Бизнес-операция    | Инженер данных для написания KSQL – запроса смотреть схему данных потока                                           |
| 6     | Получить данные по переменной              | Операция с данными | Из Оперативного хранилища или Долговременного хранилища или из Промежуточного хранилища запрашиваются данные       |
| 7     | Получить актуальные данные по переменной   | Операция с данными | Из Оперативного хранилища или из Промежуточного хранилища запрашиваются данные                                     |
| 8     | Получить исторические данные по переменной | Операция с данными | Из Долговременного хранилища или из Промежуточного хранилища запрашиваются данные                                  |

## Продолжение таблицы 2.3

|    |                     |                    |                                                                                |
|----|---------------------|--------------------|--------------------------------------------------------------------------------|
| 9  | Выбрать переменную  | Бизнес-операция    | Инженер данных выбирает переменную из списка                                   |
| 10 | Получить переменную | Операция с данными | Из Долговременного хранилища или Промежуточного хранилища запрашиваются данные |

Для варианта использования «Создать переменную» была создан следующий сценарий, изображенный на рисунке 2.2.

1. Пользователь заполняет имя переменной
2. Пользователь выбирает тип переменной
3. Пользователь вводит описание переменной
4. Пользователь пишет запрос
  - 4.1. **if** Данные по потокам неизвестны
    - 4.1.1.  Получить данные по потокам
    - end if**
5. Пользователь нажимает кнопку "Сохранить переменную"
6. Система выполняет запрос
7. **if** Запрос неправильный ИЛИ не удалось создать коннектор
  - 7.1. Система выводит сообщение об ошибке
  - end if**
8. Система выводит сообщение об успешном создании переменной

Рисунок 2.2 – Сценарий варианта использования «Создать переменную»

Пользователь взаимодействует с системой для создания новой переменной. Процесс включает несколько шагов:

- 1) ввод имени переменной.
- 2) выбор типа переменной.
- 3) ввод описания переменной.
- 4) отправка запроса на создание переменной.
- 5) сохранение переменной.

Для данного сценария варианта использования была создана диаграмма деятельности, изображенная на рисунке 2.3.

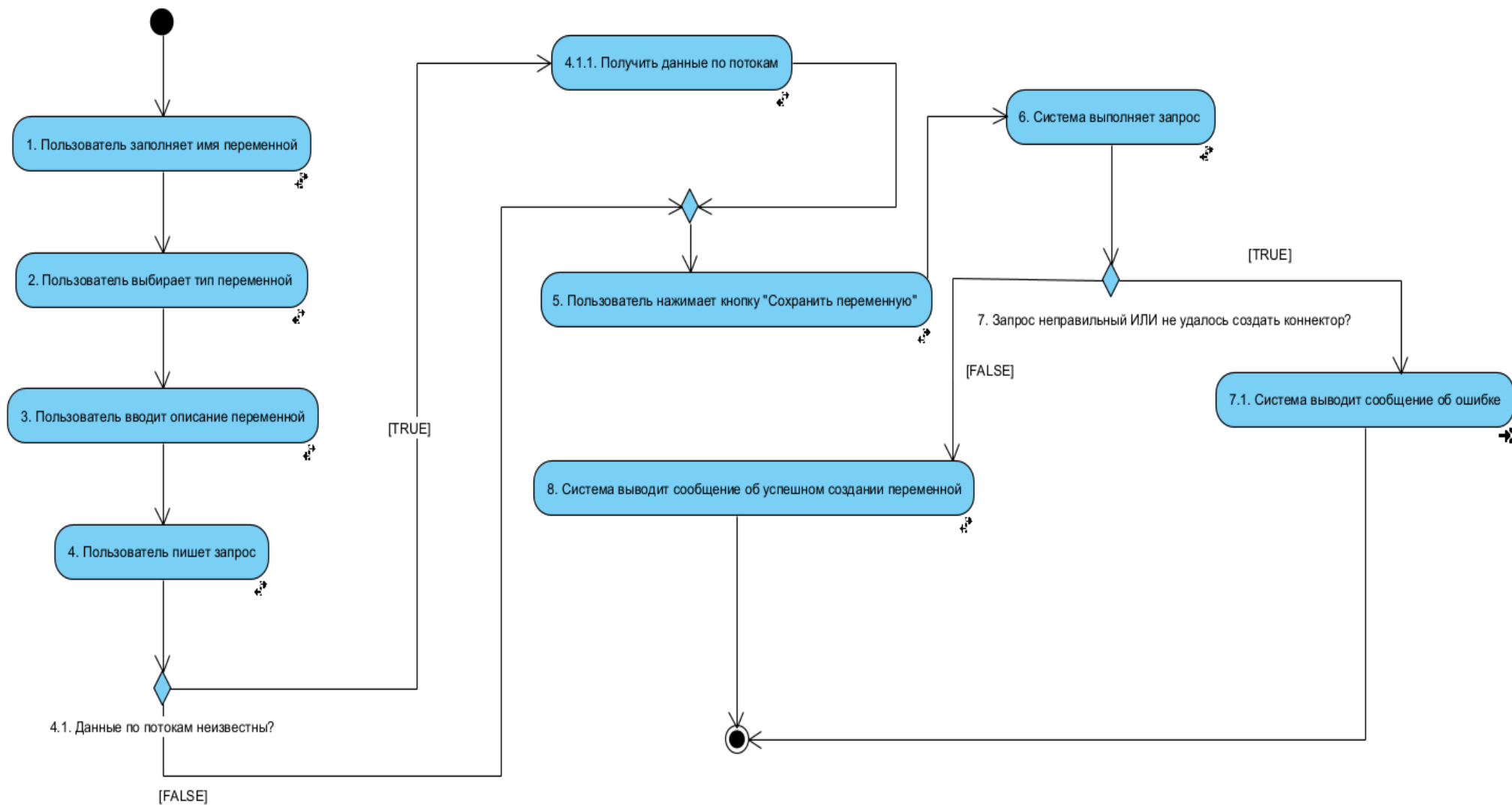


Рисунок 2.3 – Диаграмма деятельности сценария варианта использования

В разрабатываемой системе инженер данных выполняет полный цикл по созданию переменной, просмотра актуальных значений и просмотра исторических значений.

## **2.2 Анализ нефункциональных требований**

Нефункциональные требования – описание свойства или особенности, которыми должна обладать система, или ограничение, которое должна соблюдать система.

Нефункциональные требования:

- 1) производительность;
- 2) масштабируемость;
- 3) надежность;
- 4) безопасность;
- 5) удобство использования;
- 6) сопровождаемость;
- 7) совместимость;
- 8) конфиденциальность;
- 9) энергоэффективность.

Производительность – это способность системы обрабатывать большое количество запросов, за минимальное время отклика системы.

Масштабируемость – это возможность приложения поддерживать увеличение количества пользователей и объёма данных без значительных потерь производительности.

Надёжность – это способность системы обеспечивать высокий уровень отказоустойчивости и доступности, минимизируя вероятность простоев или сбоев.

Безопасность – это гарантия того, что приложение защищает данные от несанкционированного доступа и соответствует современным стандартам безопасности.

Удобство использования – это интуитивно понятный интерфейс системы, доступный для пользователей с разным уровнем подготовки.

Сопровождаемость – это возможность легко обновлять и поддерживать программное обеспечение, обеспечивая минимальные затраты ресурсов на усовершенствование и исправление ошибок.

Совместимость – это способность приложения работать на всех основных платформах и интегрироваться с другими системами и сервисами.

Конфиденциальность – это строгое соблюдение правил конфиденциальности для гарантии защиты личных и корпоративных данных.

Энергоэффективность – это оптимальное использование ресурсов оборудования, снижающее потребление энергии и уменьшающее нагрузку на инфраструктуру [11].

При анализе нефункциональных требований были рассмотрены требования к доступности, надежности, производительности, масштабируемости и удобству использования (таблица 2.4).

Таблица 2.4 – Нефункциональные требования

| Критерий               | Характеристика                                                                                      |
|------------------------|-----------------------------------------------------------------------------------------------------|
| Доступность            | Программные средства должны быть доступны 95% времени, за исключением времени обслуживания          |
| Надежность             | Программные средства должны обеспечивать восстановление после сбоя не более чем на 2 часа           |
| Производительность     | Исторические данные не должны загружаться более 10 секунд                                           |
| Масштабируемость       | Программные средства должны обеспечивать до 20 потоков, сохраняя текущий уровень производительности |
| Удобство использования | Интерфейс должен быть интуитивно понятен                                                            |

Данный набор нефункциональных требований позволит разработать надежные программные средства управления журналами событий и хранилищами исторических данных.

## **2.3 Анализ требований к пользовательскому интерфейсу панели управления**

Клиентское приложение должно иметь удобный интерфейс для взаимодействия с серверной частью.

Приложение должно иметь следующие формы:

- 1) добавление переменной;
- 2) просмотр исторических данных;
- 3) просмотр данных потоков;
- 4) просмотра данных переменной.

Форма для добавления переменной должна иметь текстовые поля для ввода информации и кнопка для просмотра данных потоков.

На форме для просмотра исторических данных должна быть таблица для отображения исторических данных, текстовые поля для настройки количества записей и комбинированный список для выбора переменной, по которой будут просмотрены данные.

Для формы просмотра данных потоков должны быть использованы комбинированный список для выбора потока и текстовые поля для вывода информации. Для просмотра данных переменной должны быть использованы текстовые поля.

## **2.4 Анализ требований к аппаратно-программному обеспечению**

### **2.4.1 Требования к аппаратному обеспечению**

Требования к аппаратному обеспечению (hardware requirements) определяют минимальные и рекомендуемые характеристики оборудования, необходимые для корректной работы программного обеспечения. Эти требования включают в себя параметры процессора, оперативной памяти, дискового пространства, видеокарты и других компонентов.

Минимальные требования – это базовые характеристики оборудования, которые гарантируют, что программное обеспечение будет работать, хотя возможно с ограничениями по производительности.

Рекомендуемые требования – это оптимальные характеристики оборудования, которые обеспечивают наилучшую производительность и стабильность работы программного обеспечения.

В таблице 2.5 представлены минимальные системные требования, на таблице 2.6 рекомендуемые системные требования для комфортной работы программных средств.

Таблица 2.5 – Минимальные требования к аппаратному обеспечению

| Оборудования                            | Минимальные требования              |
|-----------------------------------------|-------------------------------------|
| Процессор                               | 6 ядер с тактовой частотой 2.10 ГГц |
| Оперативная память                      | От 8 ГБ                             |
| Объем свободной памяти на жестком диске | От 50 ГБ                            |

Таблица 2.6 – Рекомендуемые требования к аппаратному обеспечению

| Оборудования                            | Минимальные требования              |
|-----------------------------------------|-------------------------------------|
| Процессор                               | 8 ядер с тактовой частотой 2.10 ГГц |
| Оперативная память                      | 16 ГБ                               |
| Объем свободной памяти на жестком диске | 100 ГБ                              |

В результате анализа требований к аппаратному обеспечению можно сделать вывод, что программные средства требовательны к памяти.

#### 2.4.2 Требования к системному программному обеспечению

Системное программное обеспечение (СПО) – это набор программ, которые обеспечивают функционирование компьютера и управление его ресурсами. Оно включает в себя операционные системы, драйверы устройств, утилиты и другие программы, которые обеспечивают взаимодействие между аппаратным обеспечением и прикладным программным обеспечением. Системное ПО играет ключевую роль в обеспечении стабильности и производительности компьютерной системы.

Системное программное обеспечения для развертывание приложения отображено в таблице 2.7.

Таблица 2.7 – Системное программное обеспечение для развертывания

| Компонент            | Значение                                                                                                                                          |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Операционная система | Windows 11                                                                                                                                        |
| Docker               | Программное обеспечение с открытым исходным кодом для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации |
| Visual Studio        | Интегрированная среда разработки для создания и отладки rest api и windows forms на с#.                                                           |

Такие требования к программному обеспечению позволяют развернуть программные средства управления журналами событий и хранилищами исторических данных.

### 2.4.3 Требования к средствам разработки

На основе анализа требований были определены технологии и средства разработки для реализации программных средств управления журналами событий и хранилищами исторических данных.

Для разработки клиентской части использовалась платформа Windows Forms – это технология пользовательского интерфейса для .NET, представляющая собой набор управляемых библиотек, которые упрощают выполнение стандартных задач, таких как чтение из файловой системы и запись в нее. С помощью среды разработки, такой как Visual Studio, можно создавать интеллектуальные клиентские приложения Windows Forms, которые отображают информацию, запрашивают ввод пользователя и взаимодействуют с удаленными компьютерами по сети [12]. Для написания программного кода использовался язык программирования C#.

В качестве промежуточного хранилища использовалась платформа Apache Kafka. Kafka – это отказоустойчивая система обмена сообщениями по типу «публикация/подписка». Отдельный узел Kafka называется брокером (broker), а несколько серверов Kafka образуют кластер (cluster). Записываемые генераторами (producers) сообщения Kafka хранит в топиках (topics). Потребители (consumers) подписываются на топики и обращаются к Kafka,

чтобы узнать, появились ли новые сообщения в топиках, на которые они подписаны [13].

Для создания таблиц переменных с помощью SQL – запросов используется `ksqlDB` – база данных потоковой передачи событий [14].

Для создания автоматической передачи данных из топиков, связанных с таблицами, в долговременное хранилище используется `Kafka Connect` – это инструмент для масштабируемой и надежной потоковой передачи данных между `Apache Kafka` и другими системами обработки данных [15].

Для развертывания данных средств и упаковки в единый контейнер используется `Docker`.

В качестве долговременного хранилища используется `MongoDB` – документоориентированная СУБД с открытым исходным кодом, которая хранит данные в JSON – формате.

Для оперативного хранения данных была выбрана `Redis` – NoSQL СУБД, хранящая данные в оперативной памяти.

Взаимодействие клиентской части с вышеперечисленными средствами реализуется с помощью серверной части. Для её создания использовался язык программирования `C#` на платформе `ASP.NET Core`. Серверная часть представляет из себя `RESTFul API` – сервис.

## **2.5 Постановка задачи на разработку программных средств**

Основная цель разрабатываемых программных средств управления журналами событий и хранилищами исторических данных – обеспечение сохранения и удобного просмотра исторических данных.

Основные требования к программным средствам:

- 1) клиентская часть должна обеспечивать удобное создание переменных для сохранения данных;
- 2) клиентская часть должна обеспечивать возможность просмотра актуального значения переменной;
- 3) клиентская часть должна предоставлять исторические данные;

4) серверная часть должна предоставлять запрашиваемые данные для клиентской части.

Программные средства должны обладать высокой надежностью и производительностью для обеспечения качественной обработки данных.

### 3 Разработка программных средств управления журналами событий и хранилищами исторических данных

#### 3.1 Разработка архитектуры развертывания программных средств

В результате анализа требований и постановки задачи к разработке программного обеспечения была определена архитектура программных средств управления журналами событий и хранилищами исторических данных.

Для описания архитектуры развертывания программных средств использовалась диаграмма развертывания. Диаграмма развертывания – это диаграмма, которая показывает конфигурацию узлов обработки во время выполнения и компонентов, которые на них работают [16].

Диаграмма развертывания программных средств представлена на рисунке 3.1.

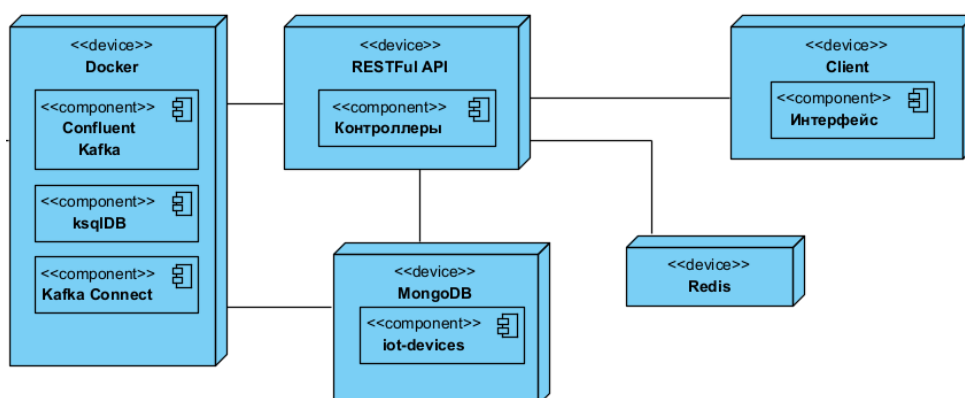


Рисунок 3.1 – Диаграмма развертывания программных средств

На диаграмме развертывания можно выделить следующие узлы устройств: «Docker», «RESTful API», «Winforms», «MongoDB», «Redis» и «Client». В «Docker» можно выделить следующие компоненты:

- 1) «Kafka Confluent»;
- 2) «ksqlDB»;
- 3) «Kafka Connect».

На основе этих данных с помощью ksqlDB создаются потоки, т. к. переменные можно создавать только на основе них.

«Kafka Connect» нужен для обеспечения передачи исторических данных в базу данных «MongoDB» коллекцию «iot-devices». Для передачи необходимо зарегистрировать коннектор, который формируется путем post – запроса к «RestFul Api».

«Client» требуется для удобного взаимодействия с «RESTFul API» через «Интерфейс».

При загрузке данных из «Kafka Confluent» или «MongoDB» данные временно сохраняются в «Redis», которая позволяет получить те же самые данные быстрее.

В «RESTful API» был реализован паттерн «Репозиторий», упрощающую абстракцию хранения данных, которая позволяет устранить связанность слоя модели и слоя данных [17].

Для более глубокого понимания архитектуры была сформирована диаграмма классов для «RESTFul API», изображенная на рисунке 3.2.

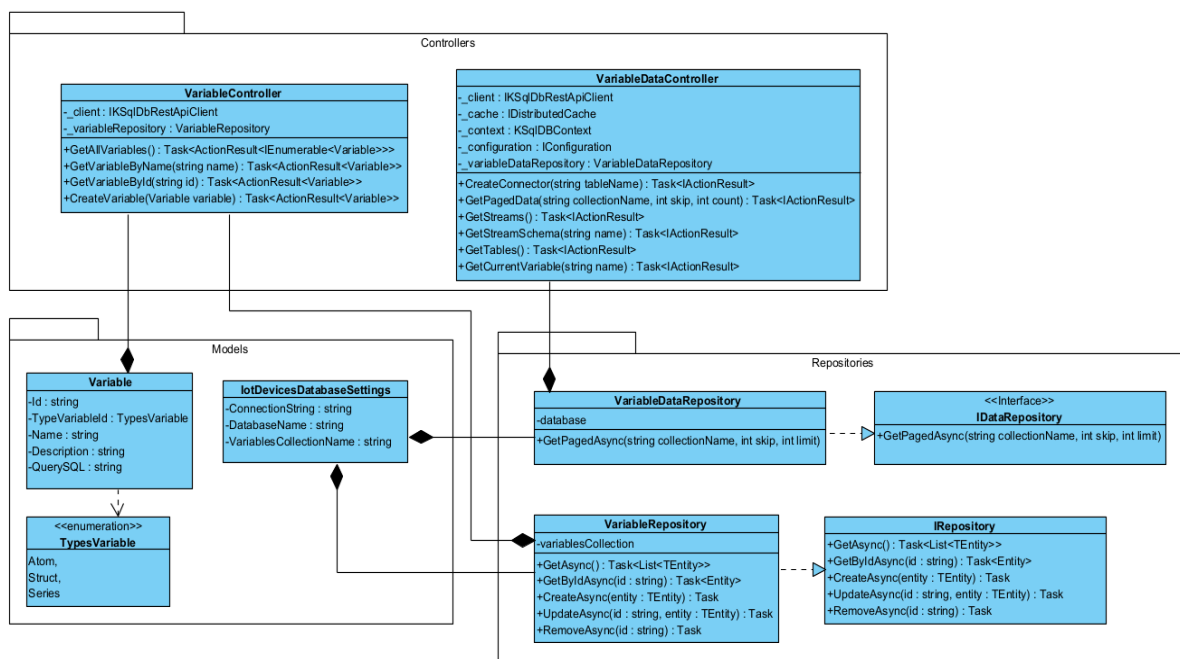


Рисунок 3.2 – Диаграмма классов «RESTFul Api»

Диаграмма разделена на несколько пакетов: «Controllers», «Models» и «Repositories».

Спецификация пакета «Controllers» представлена на таблице 3.1.

Таблица 3.1 – Спецификация пакета «Controllers»

| Класс контроллера      | Характеристика                                                                                                                                            |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| VariableController     | Отвечает за создания переменной, получение всех переменных, получения переменной по имени, получения переменной по id                                     |
| VariableDataController | Предназначен для создания коннектора, получения исторических данных, получения списка потоков, получения списка таблиц и получения актуальной переменной. |

Спецификация пакета «Controllers» представлена на таблице 3.2.

Таблица 3.2 – Спецификация пакета Models

| Модели                     | Характеристика                                                                        |
|----------------------------|---------------------------------------------------------------------------------------|
| Variable                   | Класс, который описывает логические переменные                                        |
| TypesVariable              | Перечисление, которое содержит значения типов переменной: «Atom», «Struct» и «Series» |
| IotDevicesDatabaseSettings | Класс, который содержит параметры подключения к MongoDB.                              |

Спецификация пакета Repositories представлена на таблице 3.3.

Таблица 3.3 – Спецификация пакета Repositories

| Репозиторий            | Характеристика                                                    |
|------------------------|-------------------------------------------------------------------|
| IDataRepository        | Интерфейс, который описывает поведение для VariableDataRepository |
| VariableDataRepository | Класс, реализующий поведение интерфейса IData                     |
| IRepository            | Интерфейс, описывающий поведение для классов репозитория          |
| VariableRepository     | Класс, реализующий поведение IDataRepository                      |

Таким образом, программные средства управления журналами событий построены на микросервисной архитектуре, состоящей из отдельных компонентов, каждый из которых выполняет определенную задачу. Весь код программы представлен в приложение Б.

### **3.2 Разработка RESTFul API-сервера для взаимодействия с приложениями информационно-управляющей платформы**

REST API (Representational State Transfer Application Programming Interface) – это архитектурный стиль взаимодействия между клиентом и сервером через HTTP. Он определяет принципы построения API, обеспечивая стандартизированный и эффективный обмен данными между различными системами [18].

За обработку HTTP-запросов в RESTful API отвечают контроллеры, имеющие определенный набор методов.

Контроллер VariableController обрабатывает методы, изображенные на рисунке 3.3.

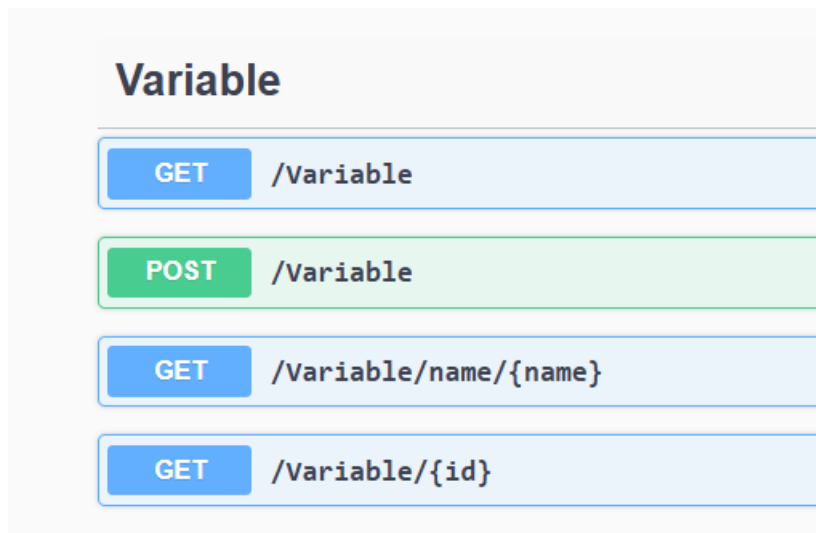


Рисунок 3.3 - Поддерживаемые методы контроллера VariableController

На таблице 3.4 описана более подробная спецификация методов VariableController.

Таблица 3.4 – Методы контроллера VariableController

| Метод                    | Название          | Входные данные | Описание                                             |
|--------------------------|-------------------|----------------|------------------------------------------------------|
| GET/Variable             | GetAllVariables   |                | отвечает за получения всех переменных из базы данных |
| POST/Variable            | CreateVariable    | variable       | добавляет переменную в базу данных                   |
| GET/Variable/name/{name} | GetVariableByName | name           | получает переменную по имени                         |
| GET/Variable/{id}        | GetVariableById   | id             | отвечает за получение переменной по id.              |

Контроллер «VariableDataController» поддерживает методы, представленные на рисунке 3.4.

| VariableData |                                                |
|--------------|------------------------------------------------|
| POST         | /api/VariableData/createConnector/{tableName}  |
| GET          | /api/VariableData/historyData/{collectionName} |
| GET          | /api/VariableData/streamsList                  |
| GET          | /api/VariableData/streamSchema/{name}          |
| GET          | /api/VariableData/tablesList                   |
| GET          | /api/VariableData/currentValue/{name}          |

Рисунок 3.4 – Поддерживаемые методы контроллера VariableDataController

На таблице 3.5 описана более подробная спецификация методов VariableDataController.

Таблица 3.5 – Методы контроллера VariableDataController

| Метод                                             | Название        | Входные данные        | Описание                                                               |
|---------------------------------------------------|-----------------|-----------------------|------------------------------------------------------------------------|
| POST/api/VariableData/createConnector/{tableName} | CreateConnector | tableName             | Создает коннектор для исторических данных переменной                   |
| GET/api/VariableData/historyData/{collectionName} | GetPagedData    | collectionName, limit | Отвечает за получения исторических данных по переменной из базы данных |
| GET/api/VariableData/streamsList                  | GetStreams      |                       | Получает список потоков                                                |
| GET/api/VariableData/streamSchema/{name}          | GetStreamSchema | id                    | Отвечает за получение переменной по id.                                |
| GET/api/VariableData/tablesList                   | GetTables       |                       | Отвечает за получения списка таблицы                                   |
| GET/api/VariableData/currentValue/{name}          | GetCurrentValue | name                  | Получает актуальное значение переменной или потока по названию         |

Данный набор http – методов позволяет в полной мере взаимодействовать клиенту с хранилищами данных и выполнять полный набор функций приложения.

Код метода POST/api/VariableData/createConnector/{tableName} для создания коннектора представлен на листинге 3.1.

### Листинг 3.1 – Код метода POST/api/VariableData/createConnector/{tableName}

```
[HttpPost("createConnector/{tableName}")]
public async Task<IActionResult> CreateConnector(string tableName)
{
    try
    {
        var tables = await _client.GetTablesAsync();
        if (!tables[0].Tables.Any(t => t.Name.Equals(tableName,
StringComparison.OrdinalIgnoreCase)))
        {
            return BadRequest($"Table '{tableName}' does not exist
in ksqlDB");
        }
        var connectorName = $"connector_{tableName.ToLower()}";
        var kafkaConnectUrl = _configuration["KafkaConnect:Url"];
        var mongoUrl = _configuration["KafkaConnect:MongoUrl"];
        if (string.IsNullOrEmpty(kafkaConnectUrl) &&
string.IsNullOrEmpty(mongoUrl))
        {
            return StatusCode(500, "Kafka Connect URLs is not
configured");
        }
        using var httpClient = new HttpClient
        {
            BaseAddress = new Uri(kafkaConnectUrl)
        };
        var connectorConfig = new
        {
            name = connectorName,
            config = new Dictionary<string, string>
            {
                ["connector.class"] =
"com.mongodb.kafka.connect.MongoSinkConnector",
                ["topics"] = tableName,
                ["connection.uri"] = mongoUrl,
                ["database"] = "iot-devices",
                ["collection"] = tableName,
                ["writes.mode"] = "upsert_document",
                ["change.data.capture.handler"] = "",
                ["key.converter"] =
"org.apache.kafka.connect.storage.StringConverter",
                ["value.converter"] =
"org.apache.kafka.connect.json.JsonConverter",
                ["value.converter.schemas.enable"] = "false"
            }
        };
    }
}
```

```

        var response = await
httpClient.PostAsJsonAsync("/connectors", connectorConfig);
        if (!response.IsSuccessStatusCode)
        {
            var error = await
response.Content.ReadAsStringAsync();
            return BadRequest(new
            {
                Success = false, Error = error, ConnectorName =
connectorName
            });
        }
        var result = await response.Content.ReadAsStringAsync();
        return Ok(new
        {
            Success = true,
            ConnectorName = connectorName,
            Result = JsonSerializer.Deserialize<object>(result)
        });
    }
    catch (Exception ex)
    {
        return StatusCode(500, new
        {
            Success = false, Error = ex.Message, ex.StackTrace
        });
    }
}

```

Метод «CreateConnect (string tableName)» принимает в качестве аргумента имя таблицы. Затем идёт проверка существования таблицы. Если таблица существует, происходит проверка на правильную конфигурацию URL-адресов Kafka Connect и MongoDB. В случае правильной конфигурации происходит настройка конфигурации для коннектора. В нем указаны следующие характеристики:

- 1) name – имя коннектора;
- 2) connector.class – класс коннектора;
- 3) topics – тема Kafka, соответствующая таблице;
- 4) connection.uri – URL-адрес MongoDB;
- 5) iot-devices – имя базы данных;
- 6) collection – коллекция в MongoDB;
- 7) writes.mode – режим записи;
- 8) key.converter – преобразователь ключа;

9) `value.converter` – преобразователь значения.

Имя коннектора создается путем интерполирования строки с именем таблицы. Класс коннектора управляет интеграцией: будут ли сохраняться (source) или выгружаться данные (sink) в/из Kafka. В данном случае используется стандартный sink-коннектор для выгрузки данных из топика в коллекцию MongoDB: «com.mongodb.kafka.connect.MongoSinkConnector».

Типы режимов записи:

1) `upsert_document` – для вставки и обновления данных, в случае совпадения ключа;

2) `insert` – для вставки документов, но может вызвать ошибку, если в коллекции уже есть документ с аналогичным ключом;

3) `replace_document` – используется для постоянной замены старого документа новым;

4) `merge_document` – объединяет данные старого документа с новым;

5) `update_document` – для обновления документов с аналогичным ключом.

Был выбран режим `upsert_document`, так как нам необходимо хранить исторические данные и сохранять постоянное движение данных.

Преобразователь ключа необходим для преобразования ключа сообщения Kafka в формат, который может обработать коннектор. Для аналогичных целей служит и преобразователь значения.

Код метода `GET/api/VariableData/currentValue/{name}` представлен на листинге 3.2.

Листинг 3.2 – Код метода `GET/api/VariableData/currentValue/{name}`

```
[HttpGet("currentValue/{name}")]
public async Task<IActionResult> GetCurrentValue(string name)
{
    try
    {
        string cacheKey = $"elem: {name}";

        var redisValue = await _cache.GetAsync(cacheKey);
        if (redisValue != null)
            return
Ok(JsonSerializer.Deserialize<object>(redisValue));
```

```

        string ksQuery = $"select * from `{name}`";

        var result = await
            _context.ExecutePullQuery<object>(ksQuery);
        if (result == null)
        {
            return NoContent();
        }

        var serializedData = JsonSerializer.Serialize(result);
        var bytes = Encoding.UTF8.GetBytes(serializedData);

        DistributedCacheEntryOptions options = new
            DistributedCacheEntryOptions
            {
                AbsoluteExpirationRelativeToNow =
                    TimeSpan.FromSeconds(5)
            };
        await _cache.SetAsync(cacheKey, bytes, options);

        return Ok(result);
    }
    catch (Exception ex)
    {
        return
            StatusCode(StatusCode.Status500InternalServerError,
                new { Error = $"Failed to get value: {ex.Message}"
            });
    }
}

```

Метод «GetCurrentValue (string name)» принимает в качестве аргумента наименование потока или таблицы. С помощью переменной cacheKey определяется, что именно нужно получить из Redis. Если значение полученной по cacheKey не равно null, то возвращается результат, иначе происходит запрос к ksqlDB, для получения актуального значения. Затем данное значение записывается в Redis.

Таким образом, было разработано RESTFull API – сервера для взаимодействия с информационно-управляющей системой.

Весь код программы представлен в приложение Б.

### 3.3 Разработка структуры для описания логических переменных

Структура для описания логической переменной представлена на листинге 3.3 в виде класса Variable.

### Листинг 3.3 – Структура описания логической переменной

```
public partial class Variable
{
    [JsonIgnore]
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string? Id { get; set; }

    public VariableTypes TypeVariableId { get; set; }

    public string Name { get; set; } = null!;

    public string Description { get; set; } = null!;

    public string QueryKsql { get; set; } = null!;
}
```

Класс переменной состоит из следующих полей:

- 1) Id – отвечает за уникальность записи;
- 2) TypeVariableId – тип переменной;
- 3) Name – название переменной;
- 4) Description – описание переменной;
- 5) QueryKsql – запрос на KSQL.

На основе данного класса можно создать множество переменных. Схемы таблиц определяются не самой переменной, а полем QueryKsql.

## 3.3 Разработка клиентского приложения

### 3.3.1 Разработка интерфейса

Клиентское приложение должно быть удобным в использовании и соответствовать функциональным требованиям, корректно посылать и обрабатывать запросы от RESTFul API – сервера.

На таблице 3.6 перечислены классы графических элементов Winforms, которые использовались при разработке пользовательского интерфейса.

Таблица 3.6 – Классы графических элементов

| Класс    | Описание                                                    |
|----------|-------------------------------------------------------------|
| Button   | Класс для создания элемента кнопки.                         |
| ComboBox | Класс для создания элемента «список выбора».                |
| Form     | Класс для создания объекта окна формы.                      |
| GroupBox | Класс для создания элемента, используемого при группировке. |

### Продолжение таблицы 3.6

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| Label         | Класс для создания надписи.                                                       |
| ListBox       | Класс для создания объекта элемента, представляющего собой раскрывающийся список. |
| Panel         | Класс для создания объекта панели – контейнера для других объектов [3].           |
| DataGridView  | Класс для создания таблиц.                                                        |
| ToolStripMenu | Класс для создания меню.                                                          |
| NumericUpDown | Класс для создания регулятора числовых значений                                   |

В результате разработки пользовательского интерфейса было создано 4 формы:

- 1) «Панель управления – Главная»;
- 2) «Панель управления – Создание переменной»;
- 3) «Данные потоков»;
- 4) «Просмотр исторических данных».

Форма «Панель управления – Главная» показана на рисунке 3.5.

Панель управления - Создание переменной

Название переменной: temperature\_current\_temp\_sensor\_1

Тип переменной: Atom

Описание: Текущее значение температуры по temp\_sensor\_1

KSQL-запрос: CREATE TABLE `temperature\_current\_temp\_sensor\_1` AS  
SELECT  
`deviceid` AS `device\_id`,  
LATEST\_BY\_OFFSET(`value`) AS `last\_value`,  
LATEST\_BY\_OFFSET(`createdAt`) AS `last\_update`  
FROM `temperature\_data\_stream`  
WHERE `deviceid` = 'temp-sensor-1' GROUP BY `deviceid` EMIT CHANGES;

Создать переменную

Посмотреть данные потоков

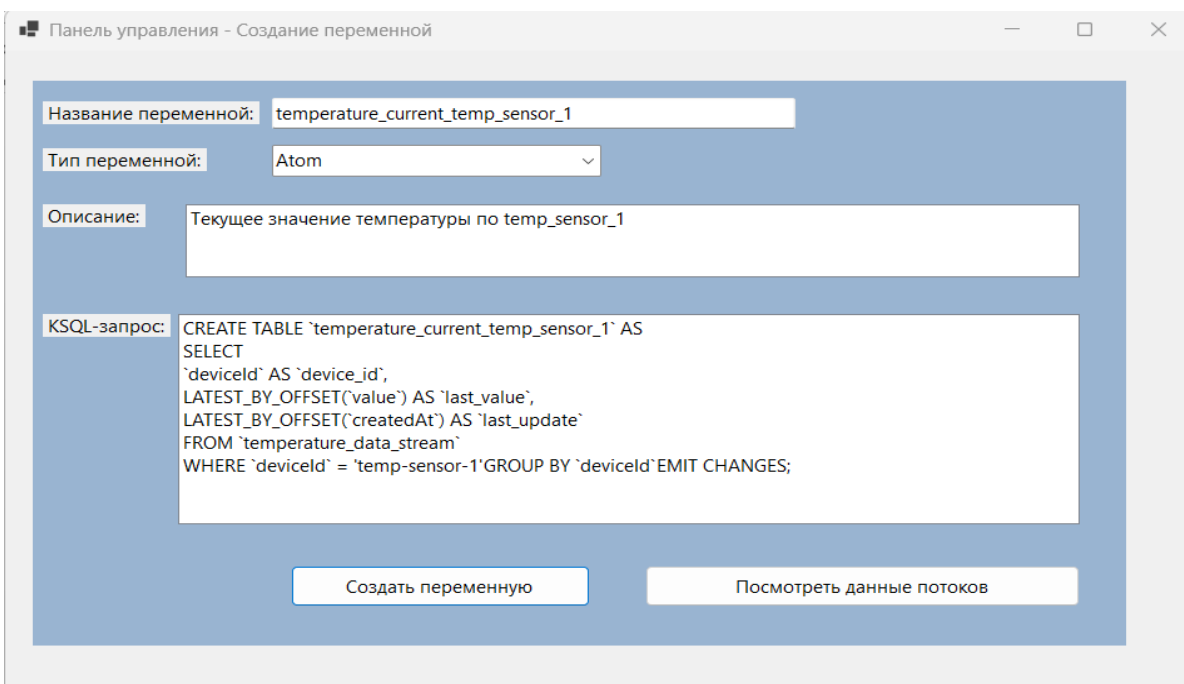
Рисунок 3.5 – Форма «Панель управления – Главная»

На данной форме представлена визуальная информация по переменной, а также можно получить исторические данные по переменной, нажав на соответствующую кнопку. Для вывода информации об имени переменной и типе переменной использовался графический элемент ComboBox. Элемент ListBox

использовался для вывода данных об описании, KSQL-запросе и актуальном значении переменной.

На рисунке 3.6 изображена форма «Панель управления – Создание переменной». На данной форме мы можем создать переменную, после того, как введем все значения полей переменной.

Для помощи в написании KSQL – запроса можно посмотреть данные потоков с помощью нажатия на кнопку «Посмотреть данные потоков». В этот раз для поля «Название переменной» использовался графический элемент TextBox.



Панель управления - Создание переменной

Название переменной: temperature\_current\_temp\_sensor\_1

Тип переменной: Atom

Описание: Текущее значение температуры по temp\_sensor\_1

KSQL-запрос: CREATE TABLE `temperature\_current\_temp\_sensor\_1` AS  
SELECT  
`deviceId` AS `device\_id`,  
LATEST\_BY\_OFFSET(`value`) AS `last\_value`,  
LATEST\_BY\_OFFSET(`createdAt`) AS `last\_update`  
FROM `temperature\_data\_stream`  
WHERE `deviceId` = `temp-sensor-1` GROUP BY `deviceId` EMIT CHANGES;

Создать переменную

Посмотреть данные потоков

Рисунок 3.6 – Форма «Панель управления – Создание переменной»

На рисунке 3.7 представлена форма «Данные потоков», которая необходима для пользователя при создании переменной. Она позволяет выбрать поток с помощью элемента ComboBox, просматривать схему потока и последнее значение из потока.

Поток:

Схема потока

```
CREATE STREAM `temperature_data_stream` (
  `deviceid` STRING,
  `createdAt` STRING,
  `value` INTEGER
) WITH (
  CLEANUP_POLICY = 'delete',
  KAFKA_TOPIC='temperature_sensors',
  KEY_FORMAT='KAFKA',
  VALUE_FORMAT='JSON'
);
```

Текущее значение

```
deviceid: temp-sensor-2,
createdAt: 2025-06-21T15:18:57+03:00,
value: 18
```

Рисунок 3.7 – Форма «Данные потоков»

На рисунке 3.8 изображена форма «Панель управления – Просмотр исторических данных».

Журнал исторических данных

| last_value | last_update               |
|------------|---------------------------|
| 20         | 2025-06-23T10:34:08+03:00 |
| 25         | 2025-06-24T10:34:08+03:00 |
| 30         | 2025-06-25T10:34:08+03:00 |
| 35         | 2025-06-26T10:34:08+03:00 |
| 40         | 2025-06-27T10:34:08+03:00 |
| 45         | 2025-06-28T10:34:08+03:00 |
| 50         | 2025-06-29T10:34:08+03:00 |
| 55         | 2025-06-30T10:34:08+03:00 |
| 60         | 2025-07-01T10:34:08+03:00 |
| 65         | 2025-07-02T10:34:08+03:00 |
| 70         | 2025-07-03T10:34:08+03:00 |

Количество записей

От:  До:

Имя переменной

Рисунок 3.8 – Форма «Панель управления – Просмотр исторических данных»

Для отображения исторических данных используется таблица – графический элемент DataGridView. Для фильтрации по количеству записей используются поля «От» и «До», представленные графическим элементом

NumericUpDown. Для выбора переменной используется ComboBox. Для применения настроек нужно нажать на кнопку «Применить».

3.3.2 Разработка программных средств клиентского приложения

Для клиентского приложения была разработана диаграмма компонентов, которая демонстрирует структуру клиентского приложения и взаимодействие ключевых модулей. Диаграмма компонентов представлена на рисунке 3.9.

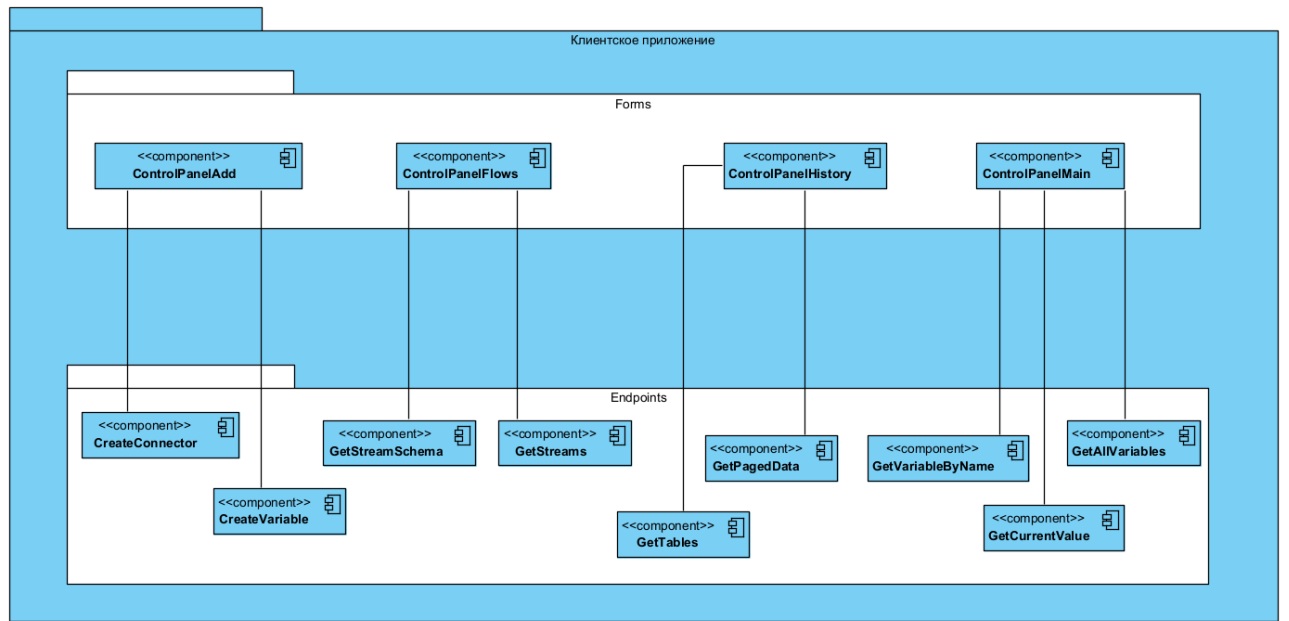


Рисунок 3.9 – Диаграмма компонентов для клиентского приложения

На диаграмме изображены два основных модуля в виде пакетов. Модуль Forms содержит компоненты форм приложения, обеспечивая их создание, обновление и управление. Модуль Endpoint включает в себя точки вызова API, которые обеспечивают возможность взаимодействия с сервисами и реализацию бизнес-логики приложения.

Подробное описание компонентов представлено в таблице 3.7.

Таблица 3.7 – Спецификации диаграммы компонентов клиентского приложения

| Модуль | Компонент           | Описание                                             |
|--------|---------------------|------------------------------------------------------|
| Forms  | ControlPanelAdd     | Форма приложения для добавления переменных           |
| Forms  | ControlPanelFlows   | Форма приложения для просмотра информации о потоках  |
| Forms  | ControlPanelHistory | Форма приложения для просмотра исторических значений |

## Продолжение таблицы 3.7

|          |                   |                                                          |
|----------|-------------------|----------------------------------------------------------|
| Forms    | ControlPanelMain  | Главная форма для просмотра созданных переменных         |
| Endpoint | CreateConnector   | API-запрос для создания коннектора                       |
| Endpoint | CreateVariable    | API-запрос для создания переменной                       |
| Endpoint | GetStreamSchema   | API-запрос для получения схемы переменной                |
| Endpoint | GetStreams        | API-запрос для получения списка потоков                  |
| Endpoint | GetTables         | API-запрос для получения таблиц                          |
| Endpoint | GetPagedData      | API-запрос для получения исторических данных             |
| Endpoint | GetVariableByName | API-запрос для получения переменной по имени             |
| Endpoint | GetCurrentValue   | API-запрос для получения актуального значения переменной |
| Endpoint | GetAllVariables   | API-запрос для получения всех значений переменной        |

Благодаря такой структуре компонентов, приложение обеспечивает полный цикл управления журналами событий и хранилищами исторических данных.

Код программы представлен в приложение Б.

## **4 Контроль качества программных средств взаимодействия с объектами мониторинга и управления**

### **4.1 Планирование процесса контроля качества приложения**

#### **4.1.1 Мероприятия по контролю качества**

Под общим понятием качества понимается количественная или качественная степень соответствия присущих объекту характеристик потребностям или ожиданиям пользователей, которые могут быть установлены как рекомендательные или обязательные [19].

К характеристикам качества программного обеспечения относятся:

- 1) функциональность – набор свойств, обуславливающий способность выполнять функции в соответствии с назначением;
- 2) надёжность – набор свойств, обуславливающий способность сохранять работоспособность за определённый период времени;
- 3) удобство – набор свойств, обуславливающий способность простого освоения программы, подготовки данных и внесение изменений в программу;
- 4) эффективность – набор свойств, обуславливающий способность программного продукта обеспечивать требуемый уровень производительности в соответствии с выделенными ресурсами, временем и другими условиями;
- 5) сопровождаемость – набор свойств, обуславливающий способность адаптации к диагностике отказов и последствий внесения изменений;
- 6) переносимость – набор свойств, обуславливающий способность адаптации к новой среде функционирования [20].

Для проверки данных характеристик и проводятся мероприятия по обеспечению качества программных средств.

В рамках данной выпускной квалифицированной работы в качестве мероприятий по контролю качества были выбраны тестирование и подсчет метрик программного кода.

#### **4.1.2 Обоснование выбора модели тестирования**

Тестирование программного обеспечения – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта [21].

Были рассмотрены следующие виды тестирования:

1) функциональное тестирование или тестирование методом черного ящика: тесты разрабатываются на основе функций и возможностей системы (описанных в документах или понятных тестировщикам) и их взаимодействия со специфичными системами и могут быть выполнены на всех уровнях тестирования [22];

2) структурное тестирование или тестирование методом белого ящика: применяется к относительно небольшим программным элементам, например, к подпрограммам или методам, у которых анализируется программный код и для получения тестовых данных используются знания о структуре компонента [23];

3) нагрузочное тестирование – это вид тестирования программного обеспечения, которое проводится для понимания поведения приложения при определенной ожидаемой нагрузке как в нормальных, так и в пиковых условиях;

4) интеграционное тестирование (Integration testing) направлено на проверку взаимодействия между частями (модулями) приложения, в результате которой выполняется поиск ошибок, связанных с трактовкой данных, реализацией интерфейса взаимодействия и совместимостью компонент приложения [24];

5) модульное тестирование – предназначено для тестирования отдельных программных элементов или модулей;

6) регрессионное тестирование – это повторное тестирование уже протестированных программ после внесения в них изменений, чтобы обнаружить дефекты, внесенные или пропущенные в результате этих действий [25].

При рассмотрении видов тестирования было выбрано функциональное тестирование, так как программные средства управления журналами событий и

хранилищами исторических данных наиболее зависимы от работоспособности RESTful API сервера. Тестирование черным ящиком позволит проверить функциональность методов сервера, в том числе способность взаимодействия API с другими системами.

#### **4.1.3 Обоснование выбора метрик программного кода**

Метрики кода – это количественные показатели, которые оценивают характеристики исходного кода и помогают определить его сложность, читаемость и качество.

Метрики кода делятся на:

- 1) метрики размера кода;
- 2) метрики сложности кода;
- 3) метрики повторяемости кода.

К метрикам размера кода относятся:

1) метрики Холстеда – основаны на анализе лексических элементов программы и позволяют оценить сложность, объем, время разработки и другие характеристики программного кода;

2) количество строк кода (SLOC, Source Lines of Code) – базовая метрика, указывающая на объем программы;

3) количество функций и классов – помогает оценить архитектурную сложность проекта;

4) средняя длина функций – показывает, насколько функции компактны и удобны для чтения.

К метрикам сложности кода относятся:

1) цикломатическая сложность (McCabe Complexity) – измеряет количество независимых путей выполнения программы;

2) коэффициент вложенности (Depth of Nesting) – показывает уровень вложенности условий, циклов и других конструкций. Сильно вложенный код сложнее понимать и изменять;

3) плотность операторов управления (Control Flow Density) – оценивает количество ветвлений и условий в коде.

К метрикам повторяемости кода относятся:

- 1) коэффициент дублирования кода (Code Duplication Ratio) – оценивает процент повторяющихся фрагментов;
- 2) коэффициент переиспользования (Reusability Index) – отражает, насколько код удобен для повторного использования в других проектах [26].

Из приведенных метрик программного кода была выбрана метрика Холстеда.

## 4.2 Тестирование приложения

### 4.2.1 Функциональное тестирование

Функциональное тестирование RESTFul API – сервера будет проводиться с помощью программы Swagger. Через Swagger UI можно выбрать конкретной адрес (эндпоинт) с заданными параметрами, чтобы совершить тестовый запрос к серверу программного продукта, который отвечает за обработку API. В этом случае Swagger UI будет визуально выводить HTTP-ответы, явно показывая коды ошибок, заголовки и тела сообщений. Это избавляет от написания специальных сниппетов кода, совершающих тестовые запросы и обрабатывающих их ответы [27].

Для проверки работоспособности эндпоинтов были составлены следующие тесты, представленные на таблице 4.1.

Таблица 4.1 – Функциональные тесты

| № | Название теста                                                    | Ожидаемый результат                                                           | Фактический результат                                                     |
|---|-------------------------------------------------------------------|-------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| 1 | Создание переменной с корректным <code>ksqldb</code> запросом     | Создаться переменная                                                          | Произошло создание переменной. Тест пройден (рисунок В.1)                 |
| 2 | Создание переменной с некорректным <code>ksqldb</code> – запросом | Будет выведено сообщение об ошибке связанное с <code>ksqldb</code> – запросом | Вывелось сообщение с ошибкой создания таблицы. Тест пройден (рисунок В.2) |

Продолжение таблицы 4.1

|    |                                                                               |                                                                     |                                                                                            |
|----|-------------------------------------------------------------------------------|---------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| 3  | Получение актуального значения по существующему имени переменной              | Будет получено актуальное значение переменной                       | Вывелось актуальное значение переменной. Тест пройден (рисунок В.3)                        |
| 4  | Получение актуального значения переменной по несуществующему имени переменной | Будет получено сообщение о том, что данной переменной не существует | Вывелось сообщение от том, что данной переменной не существует. Тест пройден (рисунок В.4) |
| 5  | Получение актуального значения по имени потока                                | Будет получено актуальное значение потока                           | Вывелось актуальное значение потока. Тест пройден (рисунок В.5)                            |
| 6  | Получение актуального значения по неправильному имени потока                  | Будет получено сообщение о том, что данного потока не существует    | Вывелось сообщение, что данного потока не существует. Тест пройден. (рисунок В.6)          |
| 7  | Получение списка потоков                                                      | Будет получен список потоков                                        | Вывелся список потоков. Тест пройден (рисунок В.7).                                        |
| 8  | Получения списка таблиц                                                       | Будет получен список таблиц                                         | Вывелся список таблиц. Тест пройден (рисунок В.8).                                         |
| 9  | Получение актуальной схемы данных потока по его имени                         | Будет получена актуальная схема потока по его имени                 | Вывелась схема потока. Тест пройден (рисунок В.9)                                          |
| 10 | Получение актуальной схемы данных потока по неправильному имени               | Будет получено сообщении о том, что данного потока не существует.   | Вывелось сообщении о том, что данного потока не существует. Тест пройден (рисунок В.10)    |
| 11 | Получение исторических данных по имени переменной.                            | Будут получены исторические данные по имени переменной.             | Вывелся список исторических данных. Тест пройден (рисунок В.11).                           |

## Окончание таблицы 4.1

|    |                                                                  |                                                                     |                                                                                            |
|----|------------------------------------------------------------------|---------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| 12 | Получение исторических данных по неправильному имени переменной. | Будет получено сообщения о том, что данной переменной не существует | Вывелось сообщения о том, что данной переменной не существует. Тест пройден (рисунок В.12) |
| 13 | Получение метаданных переменной по имени                         | Будут получены метаданные о переменной                              | Вывелись метаданные переменной. Тест пройден (рисунок В.13).                               |
| 14 | Получение метаданных по неправильному имени переменной           | Будет получено сообщения о том, что данной переменной не существует | Вывелось сообщение о том, что данной переменной не существует. Тест пройден (рисунок В.14) |
| 15 | Создание коннектора для переменной                               | Будет получено сообщение о создании коннектора                      | Вывелось сообщение о том, что коннектор успешно создан (рисунок В.15).                     |
| 16 | Создание коннектора для несуществующей переменной                | Будет получено сообщение о том, что такой переменной не существует. | Вывелось сообщение о том, что такой переменной не существует. (рисунок В.16).              |
| 17 | Создание коннектора для переменной, у которой уже есть коннектор | Будет получено сообщение о том, что такой коннектор уже есть.       | Вывелось сообщение о том, что такой коннектор уже есть. (рисунок В.17).                    |

Все результаты тестирования отображены в приложение В.

### 4.3 Оценка качества программного кода

При оценке качества программного кода использовались метрики Холстеда. Данные метрики основаны на следующих показателях:

- 1)  $n_1$  – число уникальных операторов программы, включая символы, разделители, имена процедур и знаки операций (словарь операторов);
- 2)  $n_2$  – число уникальных операндов программы (словарь операндов);
- 3)  $N_1$  – общее число операторов в программе;
- 4)  $N_2$  – общее число операндов в программе.

На основании этих показателей рассчитываются оценки:

- 1) словарь программы (Halstead Program Vocabulary, HPVoc);

2) длина программы (Halstead Program Length, HPLen);

3) объем программы (Halstead Program Volume, HPVol) [26].

Расчет метрик выполнялся на основе метода `GetCurrentValue`, приведенного на листинге 4.1.

#### Листинг 4.1 – Код метода `GetCurrentValue`

```
public async Task<IActionResult> GetCurrentValue(string name)
{
    try
    {
        var cacheKey = $"elem: {name}";

        var redisValue = await _cache.GetAsync(cacheKey);

        if (redisValue != null)
            return
                Ok(JsonSerializer.Deserialize<object>(redisValue));

        var result = await _context.ExecutePullQuery<object>(
            $"select * from `{name}`;");

        if (result == null)
        {
            return NoContent();
        }

        var serializedData = JsonSerializer.Serialize(result);
        var bytes = Encoding.UTF8.GetBytes(serializedData);

        DistributedCacheEntryOptions options = new
            DistributedCacheEntryOptions
            {
                AbsoluteExpirationRelativeToNow =
                    TimeSpan.FromSeconds(5)
            };
        await _cache.SetAsync(cacheKey, bytes, options);

        return Ok(result);
    }
    catch (Exception ex)
    {
        return
            StatusCode(StatusCode.Status500InternalServerError,
                new { Error = $"Failed to get value: {ex.Message}" });
    }
}
```

Для подсчета метрик Холстеда необходимо подсчитать количество общих и уникальных операторов и операндов. Подсчет количества операторов метода `GetCurrentValue` представлен в таблице 4.2.

Таблица 4.2 – Подсчет количества операторов метода `GetCurrentValue`

| Операторы                  | Число операторов |
|----------------------------|------------------|
| <code>public</code>        | 1                |
| <code>async</code>         | 1                |
| <code>Task</code>          | 1                |
| <code>IActionResult</code> | 1                |
| <code>try</code>           | 1                |
| <code>catch</code>         | 1                |
| <code>await</code>         | 3                |
| <code>return</code>        | 4                |
| <code>if</code>            | 2                |
| <code>var</code>           | 5                |
| <code>=</code>             | 9                |
| <code>==</code>            | 1                |
| <code>.</code>             | 8                |
| <code>()</code>            | 20               |
| <code>{}</code>            | 15               |
| <code>"</code>             | 6                |
| <code>;</code>             | 11               |
| <code>\$</code>            | 3                |
| <code>!=</code>            | 1                |

Подсчет количества операндов метода `GetCurrentValue` представлено на таблице 4.3.

Таблица 4.3 – Подсчет количества операндов метода `GetCurrentValue`

| Операнд                                               | Число вхождений |
|-------------------------------------------------------|-----------------|
| <code>GetCurrentValue</code>                          | 1               |
| <code>name</code>                                     | 3               |
| <code>cacheKey</code>                                 | 3               |
| <code>"elem: {name}"</code>                           | 1               |
| <code>redisValue</code>                               | 3               |
| <code>_cache.GetAsync</code>                          | 1               |
| <code>JsonSerializer.Deserialize&lt;object&gt;</code> | 1               |
| <code>JsonSerializer.Serialize&lt;object&gt;</code>   | 1               |
| <code>result</code>                                   | 3               |
| <code>_context.ExecutePullQuery&lt;object&gt;</code>  | 1               |
| <code>"select * from `{name}`;"</code>                | 1               |
| <code>serializedData</code>                           | 1               |
| <code>bytes</code>                                    | 2               |
| <code>Encoding.UTF8.GetBytes</code>                   | 1               |
| <code>NoContent</code>                                | 1               |
| <code>DistributedCacheEntryOptions</code>             | 2               |

## Продолжение таблицы 4.3

|                                 |   |
|---------------------------------|---|
| options                         | 2 |
| AbsoluteExpirationRelativeToNow | 1 |
| TimeSpan.FromSeconds            | 1 |
| _cache.SetAsync                 | 1 |
| Ok                              | 1 |

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 84 + 32 = 116,$$

где  $N1$  – общее число операторов;

$N2$  – общее число операндов.

Словарь программы вычисляется по формуле:

$$n = n1 + n2 = 19 + 21 = 40,$$

где  $n1$  – число уникальных операторов;

$n2$  – число уникальных операндов.

Объем программы рассчитывается по формуле:

$$HV = N * \log_2(n) = 116 * \log_2(40) = 1348,$$

где  $N$  – длина программы;

$n$  – словарь программы.

Метрики Холстеда показали, что в программе содержится 19 уникальных операторов и 21 уникальный операнд, общая длина программы составляет 116, а объем программы равен 1348 бит.

Показания метрик указывают на то, что процедура имеет средний уровень сложности.

## Заключение

В рамках выпускной квалификационной работы были успешно реализованы программные средства управления журналами событий и хранилищами исторических данных.

Разработка осуществлялась на основе современных принципов проектирования клиент-серверных приложений событийно-ориентированной архитектуры и модульного подхода к логике и интерфейсу.

На начальном этапе был проведён анализ предметной области, охватывающий ключевые сущности, такие как устройства или события. На основании этого анализа были выделены основные бизнес-процессы, сформированы сценарии использования и определены требования к системе.

В процессе планирования разработки был составлен перечень задач, установлены зависимости и сроки выполнения, подготовлена диаграмма Ганта, таблица ресурсов и рассчитаны бюджетные ограничения. Это позволило систематизировать этапы проектирования и обеспечило управляемость процесса.

Далее была выполнена формализация требований и постановка задачи. Определены функциональные и нефункциональные требования, разработана архитектура приложения, модели представления данных и RESTFul API.

Таким образом, в результате выполнения работы:

- 1) разработано полнофункциональное серверное приложение;
- 2) реализована масштабируемая архитектура, пригодная для дальнейшего развития и интеграции новых компонентов;
- 3) подтверждено соответствие программного продукта заявленным требованиям посредством функционального тестирования.

Результаты работы могут быть использованы для внедрения в реальную производственную среду, а также служить основой для расширения функциональности в будущих проектах, например, добавление аналитики и машинного обучения.

### Список использованных источников

1. Шибанов, С. В. Интегрированная модель активных правил / С. В. Шибанов, А. А. Скоробогатько, Э.В. Лысенко // Математическое и программное обеспечение систем в промышленной и социальной сферах: междунар. сб. науч. трудов - Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2011 - Ч1. - с.41 - 46.

2. Шибанов, С. В. Сервис потоковой обработки событий и исполнения активных правил / С. В. Шибанов, Я. С. Шлепнев // Математическое моделирование и суперкомпьютерные технологии : Труды XXI Международной конференции, Нижний Новгород, 22–26 ноября 2021 года. – Нижний Новгород: Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского, 2021. – С. 403 - 407.

3. Шибанов, С. В. Мета модель сервиса потоковой обработки событий / С. В. Шибанов, Я. С. Шлепнев, А. С. Гусаров // Методы, средства и технологии получения и обработки измерительной информации («Шляндинские чтения - 2022»): материалы XIV Международной научно-технической конференции с элементами научной школы и конкурсом научно-исследовательских работ для обучающихся и молодых ученых, Пенза, 24–26 октября 2022 года. – Пенза: Пензенский государственный университет, 2022. – С. 240 - 243.

4. Шибанов, С. В. Портал управления метаданными сервиса потоковой обработки событий и исполнения активных правил / С. В. Шибанов, А. С. Гусаров, Я. С. Шлепнев // Информационные технологии в моделировании и управлении: подходы, методы, решения: Материалы VI Всероссийской научной конференции с международным участием, Тольятти, 18–20 апреля 2023 года. – Тольятти: Тольяттинский государственный университет, 2023. – С. 481 - 489.

5. Фаулер, М. UML. Основы, 3е издание. – Пер. с англ. / М. Фаулер – СПб: СимволПлюс, 2004. – 192 с.

6. OrangeScada. Введение / OrangeScada [Электронный ресурс] URL: <https://www.orangescada.ru/docs/introduction.html> (Дата обращения 15.05.2025);

7. MasterSCADA 4D / MasterSCADA [Электронный ресурс] URL: <https://masterscada.ru/masterscada4d> (Дата обращения 15.05.2025);

8. Обзор платформы [Электронный ресурс] URL: [https://aggregate.digital/docs/aggregate/ru\\_6.4/basics.htm](https://aggregate.digital/docs/aggregate/ru_6.4/basics.htm) (Дата обращения 15.05.2025).

9. Функциональные и нефункциональные требования: различия и особенности // Практикум | Блог [Электронный ресурс] URL: <https://practicum.yandex.ru/blog/funkcionalnye-i-nefunkcionalnye-trebovaniya/> (дата обращения: 25.05.2025).

10. Моисеев, А. Н. Основы языка UML: учеб. пособие. / А. Н. Моисеев, М.И. Литовченко – Томск: Издательство Томского государственного университета, 2023 – 96 с.

11. Вигерс, К. Разработка требований к программному обеспечению. 3-е изд., дополненное / К. Вигерс, Б. Джой - Пер. с англ. – М.: Издательство «Русская редакция»; СПб.: БХВ-Петербург, 2014. - 736 стр.

12. Обзор Windows Forms // Microsoft Learn [Электронный ресурс] URL: <https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/> (дата обращения: 26.05.2025).

13. Беджек, Б Kafka Streams в действии. Приложения и микросервисы для работы в реальном времени. / Б. Беджек – СПб.: Питер, 2019. – 304 с.

14. Сеймур, М. Kafka Streams и ksqlDB: данные в реальном времени. / М. Сеймур - СПб.: Питер, 2023. – 432 с.

15. Kafka Connect // Confluent | Documentation [Электронный ресурс] URL: <https://docs.confluent.io/platform/current/connect/index.html> (дата обращения: 27.05.2025).

16. What is Deployment Diagram? // Visual Paradigm [Электронный ресурс] URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/> (дата обращения: 28.05.2025).

17. Персиваль Г. Грегори Б. Паттерны разработки на Python: TDD, DDD и событийно-ориентированная архитектура. / Г. Персиваль, Б. Грегори. СПб.: Питер, 2022 – 336 с.: ил.

18. Васильев, А. Н. Программирование на C# для начинающих. Особенности языка. / А. Васильев. – Москва: Эксмо, 2019 – 528 с.

19. Белик, А. Г. Качество и надежность программных систем: учеб. пособие / А. Г. Белик, В. Н. Цыганенко ; Минобрнауки России, ОмГТУ. – Омск : Изд – во ОмГТУ, 2018. – 80 с.

20. Семахин, А. М. Методы верификации и оценки качества программного обеспечения: учебное пособие. / А. М. Семахин. – Курган: Изд-во КГУ, 2018 – 150 с.

21. Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. - Минск: Четыре четверти, 2020. – 312 с.

22. Пероцкая, В. Н. Основы тестирования программного обеспечения: учеб. пособие / В. Н. Пероцкая, Д. А. Градусов ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир: Изд-во ВлГУ, 2017 – 100 с.

23. Лихицкий, А. С. Исследование стратегий тестирования программного обеспечения / А. С. Лихицкий. – Текст: непосредственный // Молодой ученый. – 2016. – № 9 (113). – С. 71-74. – URL: <https://moluch.ru/archive/113/28881/> (дата обращения: 29.05.2025).

24. Кулаков, Кирилл Александрович Основы тестирования программного обеспечения [Электронный ресурс]: учебное электронное пособие для обучающихся Института математики и информационных технологий / К. А. Кулаков, В. М. Димитров; М-во образования и науки Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования Петрозавод. гос. ун-т. – Петрозаводск : Издательство ПетрГУ, 2018.

25. Метрики в программировании: виды и применение // Polygant [Электронный ресурс] URL: <https://polygant.net/ru/blog/metriki-v-programmirovanii-vidy-i-primenenie/?ysclid=mc99e25sn5520263774> (дата обращения: 01.06.2025).

26. Программный код и его метрики // Хабр [Электронный ресурс] URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 02.06.2025).

27. Курушин, М. Что такое Swagger и как он облегчает работу с API // Timeweb [Электронный ресурс] URL: <https://timeweb.cloud/blog/что-такое-swagger-i-kak-on-oblegchaet-rabotu-s-api#testirovanie-api> (дата обращения: 02.06.2025).

**Приложение А**  
**(справочное)**  
**Глоссарий**

ИУС – информационно–управляющие системы.

IoT (Internet of Things) – интернет вещей.

MQTT (Message Queuing Telemetry Transport) – протокол обмена данными для интернета вещей.

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста.

HTTPS ((HyperText Transfer Protocol Secure) – протокол защищённой передачи данных.

API (Application Programming Interface, программный интерфейс приложения) – это набор правил и способов, с помощью которых разные сервисы обмениваются друг с другом данными.

Yandex IoT Core – сервис для интернета вещей.

ПС – программные средства.

ПО – программное обеспечение.

Топик – канал отправки и получения сообщений.

Поток – это неизменяемая секционированная последовательность событий или записей, которая представляет данные, хранящиеся в топиках Apache Kafka.

Коннектор – это исполняемые библиотеки, которые отвечают за перемещение данных между брокером Kafka и внешними системами.

UML (Unified Modeling Language) – язык моделирования.

REST (Representational State Transfer) – это архитектурный стиль для разработки веб-сервисов, который использует стандартные HTTP-методы для взаимодействия между клиентом и сервером.

Endpoint – конечная точка веб-сервиса, к которой клиентское приложение обращается для выполнения определённых операций или получения данных.

**Приложение Б**  
**(обязательное)**  
**Листинг RESTFul API**

## Листинг Б.1 – Код контроллера VariableController

```

using ControlPanelApi.Models;
using ControlPanelApi.Repositories;
using ksqlDB.RestApi.Client.KSql.RestApi;
using ksqlDB.RestApi.Client.KSql.RestApi.Statements;
using Microsoft.AspNetCore.Mvc;

namespace ControlPanelApi.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class VariableController : ControllerBase
    {
        private readonly IKSqlDbRestApiClient _client;
        private readonly VariableRepository _variableRepository;

        public VariableController(IKSqlDbRestApiClient client,
            IConfiguration configuration,
            VariableRepository variableRepository)
        {
            _variableRepository = variableRepository;
            _client = client;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Variable>>>
            GetAllVariables()
        {
            try
            {
                var variables = await
                    _variableRepository.GetAsync();
                if (variables == null)
                {
                    return NotFound();
                }
                return Ok(variables);
            }
            catch (Exception ex)
            {
                return
                    StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
            }
        }

        [HttpGet("name/{name}")]
        public async Task<ActionResult<Variable>>
            GetVariableByName(string name)
        {
            try
            {

```

```

        var variable = await
        _variableRepository.GetByNameAsync(name);
        if (variable == null)
        {
            return NotFound();
        }
        return Ok(variable);
    }
    catch (Exception ex)
    {
        return
        StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
    }
}

[HttpGet("{id}")]
public async Task<ActionResult<Variable>>
GetVariableById(string id)
{
    try
    {
        var variable = await
        _variableRepository.GetByIdAsync(id);
        if (variable == null)
        {
            return NotFound();
        }
        return Ok(variable);
    }
    catch (Exception ex)
    {
        return
        StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
    }
}

[HttpPost]
public async Task<ActionResult<Variable>>
CreateVariable([FromBody] Variable variable)
{
    try
    {
        var result = await
        _client.ExecuteStatementAsync(new
        KSqlDbStatement(variable.QueryKsql));

        if (!result.IsSuccessStatusCode)
        {
            var errorContent = await
            result.Content.ReadAsStringAsync();
            return BadRequest($"KSQL execution failed:
            {errorContent}");
        }
    }
}

```

```

        await _variableRepository.CreateAsync(variable);

        return CreatedAtAction(nameof(GetVariableById),
new { id = variable.Id }, variable);
    }
    catch (Exception ex)
    {
        return
        StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
    }
}
}
}

```

## Листинг Б.2 – Код контроллера VariableDataController

```

using ControlPanelApi.Repositories;
using ksqlDB.RestApi.Client.KSql.Query.Context;
using ksqlDB.RestApi.Client.KSql.RestApi;
using ksqlDB.RestApi.Client.KSql.RestApi.Statements;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Caching.Distributed;
using System.Text;
using System.Text.Json;
using System.Text.Json.Nodes;

namespace KsqlDbApi.Controllers;

[ApiController]
[Route("api/[controller]")]
public class VariableDataController : ControllerBase
{
    private readonly IKSqlDbRestApiClient _client;
    private readonly IDistributedCache _cache;
    private readonly KSqlDBContext _context;
    private readonly IConfiguration _configuration;
    private readonly VariableDataRepository
_variableDataRepository;

    public VariableDataController(IKSqlDbRestApiClient client,
IConfiguration configuration,
IDistributedCache cache, KSqlDBContext kSqlDBContext,
VariableDataRepository variableDataRepository)
    {
        _client = client;
        _cache = cache;
        _context = kSqlDBContext;
        _configuration = configuration;
        _variableDataRepository = variableDataRepository;
    }
}

```

```

[HttpPost("createConnector/{tableName}")]
public async Task<IActionResult> CreateConnector(string
tableName)
{
    try
    {
        var tables = await _client.GetTablesAsync();

        if (!tables[0].Tables.Any(t =>
t.Name.Equals(tableName, StringComparison.OrdinalIgnoreCase)))
        {
            return BadRequest($"Table '{tableName}' does not
exist in ksqlDB");
        }

        var connectorName =
$"connector_{tableName.ToLower()}";

        var kafkaConnectUrl =
_configuration["KafkaConnect:Url"];

        var mongoUrl =
_configuration["KafkaConnect:MongoUrl"];

        if (string.IsNullOrEmpty(kafkaConnectUrl) &&
string.IsNullOrEmpty(mongoUrl))
        {
            return StatusCode(500, "Kafka Connect URLs is not
configured");
        }

        using var httpClient = new HttpClient
        {
            BaseAddress = new Uri(kafkaConnectUrl)
        };

        var connectorConfig = new
        {
            name = connectorName,
            config = new Dictionary<string, string>
            {
                ["connector.class"] =
"com.mongodb.kafka.connect.MongoSinkConnector",
                ["topics"] = tableName,
                ["connection.uri"] = mongoUrl,
                ["database"] = "iot-devices",
                ["collection"] = tableName,
                ["writes.mode"] = "upsert_document",
                ["change.data.capture.handler"] = "",
                ["key.converter"] =
"org.apache.kafka.connect.storage.StringConverter",
                ["value.converter"] =
"org.apache.kafka.connect.json.JsonConverter",
            }
        }
    }
}

```

```

        ["value.converter.schemas.enable"] = "false"
    }
};

var response = await
httpClient.PostAsJsonAsync("/connectors", connectorConfig);

if (!response.IsSuccessStatusCode)
{
    var error = await
response.Content.ReadAsStringAsync();
    return BadRequest(new
    {
        Success = false,
        Error = error,
        ConnectorName = connectorName
    });
}

var result = await
response.Content.ReadAsStringAsync();
return Ok(new
{
    Success = true,
    ConnectorName = connectorName,
    Result =
JsonSerializer.Deserialize<object>(result)
});
}
catch (Exception ex)
{
    return StatusCode(500, new
    {
        Success = false,
        Error = ex.Message,
        ex.StackTrace
    });
}

}

[HttpGet("historyData/{collectionName}")]
public async Task<IActionResult> GetPagedData(
    string collectionName,
    [FromQuery] int skip = 0,
    [FromQuery] int limit = 10)
{
    try
    {
        var cacheKey =
        $"{collectionName}_skip_{skip}_limit_{limit}";

        var redisValue = await _cache.GetAsync(cacheKey);
        if (redisValue != null)

```

```

        return
        Ok(JsonSerializer.Deserialize<object>(redisValue));

        var data = await
        _variableDataRepository.GetPagedAsync<object>(collectionName,
        skip, limit);
        if (data == null)
        {
            return NotFound();
        }

        var serializedData = JsonSerializer.Serialize(data);
        var bytes = Encoding.UTF8.GetBytes(serializedData);

        DistributedCacheEntryOptions options = new
        DistributedCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow =
        TimeSpan.FromSeconds(5)
        };
        await _cache.SetAsync(cacheKey, bytes, options);

        return Ok(data);
    }
    catch (Exception)
    {
        return StatusCode(500, "Internal server error");
    }
}

[HttpGet("streamsList")]
public async Task<IActionResult> GetStreams()
{
    try
    {
        var infoForStreams = await _client.GetStreamsAsync();

        var streamsList = infoForStreams[0].Streams?
            .Where(stream => stream.Name?.EndsWith("stream",
                StringComparison.OrdinalIgnoreCase) == true)
            .Select(stream => stream.Name)
            .ToList();

        if (streamsList == null || streamsList.Count == 0)
        {
            return NotFound();
        }
        return Ok(streamsList);
    }
    catch (Exception ex)
    {
        return
        StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
    }
}

```

```

    }
}

[HttpGet("streamSchema/{name}")]
public async Task<IActionResult> GetStreamSchema(string name)
{
    try
    {
        var query = new KSqlDbStatement($"DESCRIBE
`{name}`;");
        var response = await
_client.ExecuteStatementAsync(query);

        if (!response.IsSuccessStatusCode)
        {
            return StatusCode((int)response.StatusCode, await
response.Content.ReadAsStringAsync());
        }

        var jsonResponse = await
response.Content.ReadAsStringAsync();
        var jsonNode = JsonNode.Parse(jsonResponse);

        var createStatement =
jsonNode?[0]?["sourceDescription"]?["statement"]?.GetValue<string>
();

        if (string.IsNullOrEmpty(createStatement))
        {
            return NotFound($"Could not extract schema for
{name}");
        }

        return Ok(createStatement);
    }
    catch (Exception ex)
    {
        return
StatusCode(StatusCode.Status500InternalServerError, ex.Message);
    }
}

[HttpGet("tablesList")]
public async Task<IActionResult> GetTables()
{
    try
    {
        var infoForTables = await _client.GetTablesAsync();

        var tablesList = infoForTables[0].Tables?
.Select(table => table.Name)
.ToList();
    }
}

```

```

        if (tablesList == null || tablesList.Count == 0)
        {
            return NotFound();
        }
        return Ok(tablesList);
    }
    catch (Exception ex)
    {
        return
        StatusCode(StatusCodes.Status500InternalServerError, ex.Message);
    }
}

[HttpGet("currentValue/{name}")]
public async Task<IActionResult> GetCurrentValue(string name)
{
    try
    {
        var cacheKey = $"elem: {name}";

        var redisValue = await _cache.GetAsync(cacheKey);

        if (redisValue != null)
            return
            Ok(JsonSerializer.Deserialize<object>(redisValue));

        var result = await _context.ExecutePullQuery<object>(
            $"select * from `{name}`");

        if (result == null)
        {
            return NoContent();
        }

        var serializedData = JsonSerializer.Serialize(result);
        var bytes = Encoding.UTF8.GetBytes(serializedData);

        DistributedCacheEntryOptions options = new
        DistributedCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow =
            TimeSpan.FromSeconds(5)
        };
        await _cache.SetAsync(cacheKey, bytes, options);

        return Ok(result);
    }
    catch (Exception ex)
    {
        return
        StatusCode(StatusCodes.Status500InternalServerError,
            new { Error = $"Failed to get value: {ex.Message}"
        });
    }
}

```

```

    }
}
}

```

### Листинг Б.3 – Код модели IotDevicesDatabaseSettings

```

namespace ControlPanelApi.Models
{
    public class IotDevicesDatabaseSettings
    {
        public string ConnectionString { get; set; } = null!;

        public string DatabaseName { get; set; } = null!;

        public string VariablesCollectionName { get; set; } =
null!;
    }
}

```

### Листинг Б.4 – Код модели Variable

```

using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
using System.Text.Json.Serialization;

namespace ControlPanelApi.Models;

[JsonConverter(typeof(JsonStringEnumConverter))]
public enum VariableTypes
{
    Atom,
    Struct,
    Series
}

public partial class Variable
{
    [JsonIgnore]
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string? Id { get; set; }

    public VariableTypes TypeVariableId { get; set; }

    public string Name { get; set; } = null!;

    public string Description { get; set; } = null!;

    public string QueryKsql { get; set; } = null!;
}

```

### Листинг Б.5 – Код репозитория IDataRepository

```
namespace ControlPanelApi.Repositories
{
    public interface IDataRepository
    {
        public Task<List<T>> GetPagedAsync<T>(string
collectionName, int skip, int limit);
    }
}
```

### Листинг Б.6 – Код репозитория IRepository

```
namespace ControlPanelApi.Repositories
{
    public interface IRepository<T>
    {
        Task<List<T>> GetAsync();
        Task<T?> GetByNameAsync(string name);
        Task<T?> GetByIdAsync(string id);
        Task CreateAsync(T entity);
        Task UpdateAsync(string id, T updatedEntity);
        Task RemoveAsync(string id);
    }
}
```

### Листинг Б.7 – Код репозитория VariableDataRepository

```
using ControlPanelApi.Models;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

namespace ControlPanelApi.Repositories
{
    public class VariableDataRepository: IDataRepository
    {
        private readonly IMongoDatabase _database;

        public VariableDataRepository(
            IOptions<IotDevicesDatabaseSettings>
IotDevicesDatabaseSettings)
        {
            var mongoClient = new MongoClient(
IotDevicesDatabaseSettings.Value.ConnectionString);

            _database = mongoClient.GetDatabase(
                IotDevicesDatabaseSettings.Value.DatabaseName);
        }

        public async Task<List<T>> GetPagedAsync<T>(string
collectionName, int skip, int limit)
```

```

        {
            return await
            _database.GetCollection<T>(collectionName)
                .Find(_ => true)
                .Skip(skip)
                .Limit(limit)
                .ToListAsync();
        }
    }
}

```

### Листинг Б.8 – Код репозитория VariableRepository

```

using ControlPanelApi.Models;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

namespace ControlPanelApi.Repositories
{
    public class VariableRepository: IRepository<Variable>
    {
        private readonly IMongoCollection<Variable>
        _variablesCollection;

        public VariableRepository(
            IOptions<IotDevicesDatabaseSettings>
            IotDevicesDatabaseSettings)
        {
            var mongoClient = new MongoClient(

            IotDevicesDatabaseSettings.Value.ConnectionString);

            var mongoDatabase = mongoClient.GetDatabase(
                IotDevicesDatabaseSettings.Value.DatabaseName);

            _variablesCollection =
            mongoDatabase.GetCollection<Variable>(

            IotDevicesDatabaseSettings.Value.VariablesCollectionName);
        }

        public async Task<List<Variable>> GetAsync() =>
            await _variablesCollection.Find(_ =>
            true).ToListAsync();

        public async Task<Variable?> GetByNameAsync(string name)
        =>
            await _variablesCollection.Find(x => x.Name ==
            name).FirstOrDefaultAsync();

        public async Task<Variable?> GetByIdAsync(string id) =>

```

```

        await _variablesCollection.Find(x => x.Id ==
id).FirstOrDefaultAsync();

        public async Task CreateAsync(Variable entity) =>
            await _variablesCollection.InsertOneAsync(entity);

        public async Task UpdateAsync(string id, Variable
updatedTypesVariable) =>
            await _variablesCollection.ReplaceOneAsync(x => x.Id
== id, updatedTypesVariable);

        public async Task RemoveAsync(string id) =>
            await _variablesCollection.DeleteOneAsync(x => x.Id ==
id);
    }
}

```

### Листинг Б.9 – Код файла Program.cs

```

using ControlPanelApi.Repositories;
using ControlPanelApi.Models;
using ksqlDB.RestApi.Client.KSql.Query.Context;
using ksqlDB.RestApi.Client.KSql.Query.Context.Options;
using ksqlDB.RestApi.Client.KSql.RestApi;
using ksqlDB.RestApi.Client.KSql.RestApi.Http;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddHttpClient();

builder.Services.Configure<IoTDevicesDatabaseSettings>(
    builder.Configuration.GetSection("IoTDevicesDatabase"));

builder.Services.AddScoped<VariableRepository>();

builder.Services.AddScoped<VariableDataRepository>();

builder.Services.AddScoped<IKSqlDbRestApiClient>(_ =>
{
    var httpClient = new HttpClient()
    {
        BaseAddress = new Uri(builder.Configuration["KsqlDb:Url"]
?? "http://localhost:8088")
    };
    var httpClientFactory = new HttpClientFactory(httpClient);
    return new KSqlDbRestApiClient(httpClientFactory);
});

builder.Services.AddScoped<KSqlDBContext>(provider =>
{
    var options = new KSqlDbContextOptionsBuilder()
        .UseKSqlDb(builder.Configuration["KsqlDb:Url"]

```

```

        ?? "http://localhost:8088")
        .Options;

    return new KSqlDbContext(options);
});

builder.Services.AddStackExchangeRedisCache(options =>
{
    options.Configuration =
builder.Configuration["RedisCacheOptions:Configuration"];
    options.InstanceName =
builder.Configuration["RedisCacheOptions:InstanceName"];
});

builder.Services.AddControllers();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

if (app.Environment.IsProduction())
{
    app.UseHttpsRedirection();
}

app.MapControllers();

app.Run();

```

**Приложение В**  
**(рекомендуемое)**  
**Тестирование**

201

Undocumented

Response body

```
{
  "typeVariableId": "Atom",
  "name": "humidityCurrentValue",
  "description": "Последнее значение датчика движения humidity-sensor-1",
  "queryKsql": "CREATE TABLE `motion_current` AS SELECT `deviceId` AS `device_id`, LATEST_BY_OFFSET(`value`) AS `last_status`, LATEST_BY_OFFSET(`createdAt`) AS `last_update` FROM `motion_data_stream` WHERE `deviceId` = 'humidity-sensor-1' GROUP BY `deviceId` EMIT CHANGES;"
}
```



Download

Рисунок В.1 – Функциональный тест №1

400

Undocumented

Error: response status is 400

Response body

```
KSQL execution failed: {"@type":"statement_error","error_code":40001,"message":"line 1:89: Syntax Error\nSyntax error at or near ':= ' at line 1:89","statementText":"CREATE TABLE `temperature_stats` AS SELECT `deviceId` AS `device_id`, STRUCT(`deviceId` := `deviceId`, `avg_value` := ROUND(AVG(`value`), 2), `max_value` := MAX(`value`), `min_value` := MIN(`value`), `readings_count` := COUNT(*), `time_start` := TIMESTAMPTOSTRING(WINDOWSTART, 'yyyy-MM-dd HH:mm:ss', 'Europe/Moscow'), `time_end` := TIMESTAMPTOSTRING(WINDOWEND, 'yyyy-MM-dd HH:mm:ss', 'Europe/Moscow')) AS `time_stats` FROM `temperature_data_stream` WINDOW TUMBLING (SIZE 2 MINUTES, GRACE PERIOD 1 MINUTE) GROUP BY `deviceId` EMIT FINAL;","entities":[]}
```



Download

Response headers

Рисунок В.2 – Функциональный тест №2

200

Response body

```
{
  "device_id": "temp-sensor-1",
  "last_value": 20,
  "last_update": "2025-06-21T23:13:18+03:00"
}
```

Рисунок В.3 – Функциональный тест №3

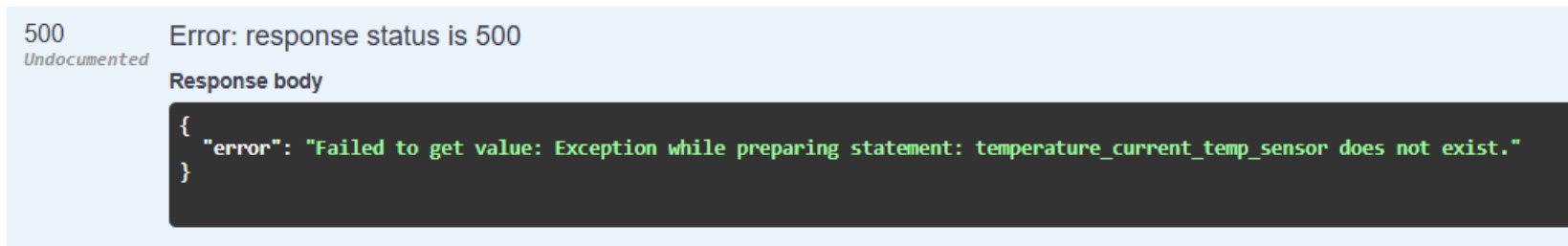


Рисунок В.4 – Функциональный тест №4

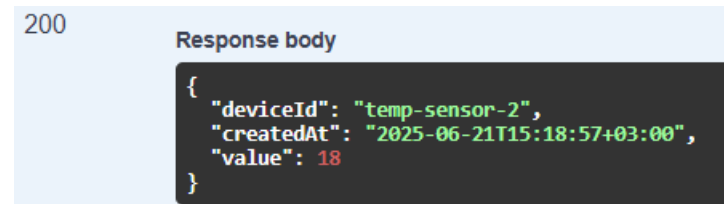


Рисунок В.5 – Функциональный тест №5

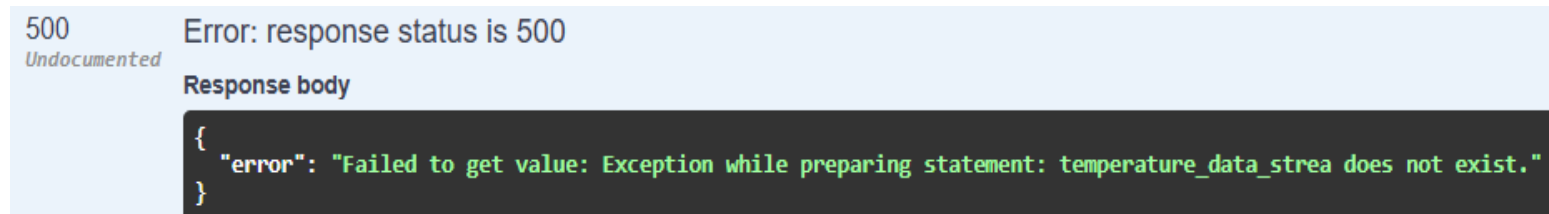


Рисунок В.6 – Функциональный тест №6

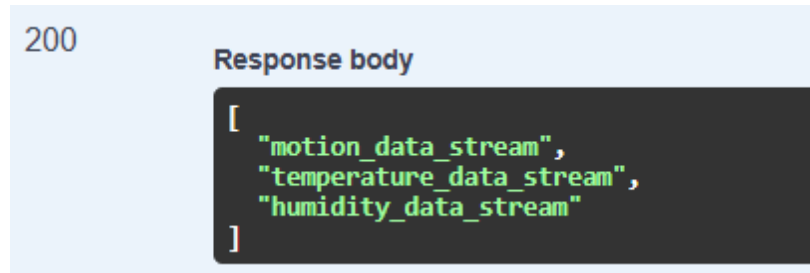


Рисунок В.7 – Функциональный тест №7

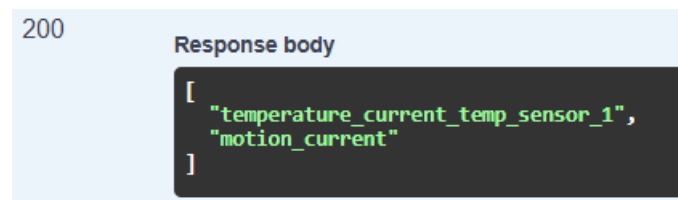


Рисунок В. 8 – Функциональный тест №8

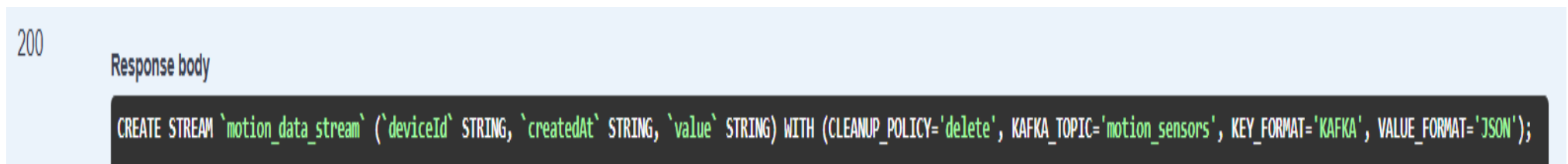


Рисунок В.9 – Функциональный тест №9

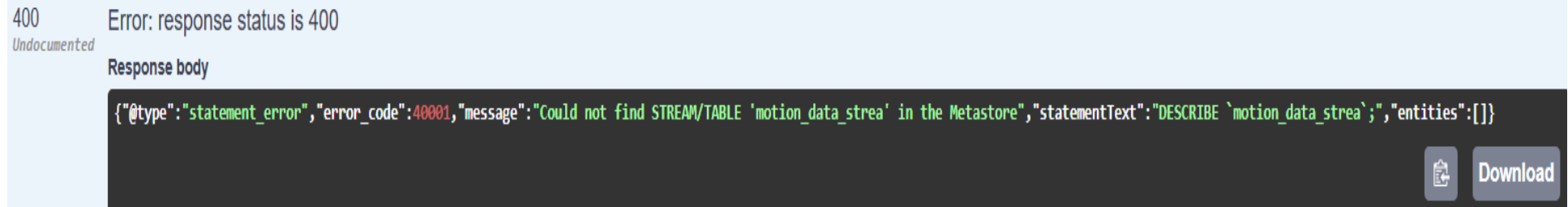


Рисунок В.10 – Функциональный тест №10

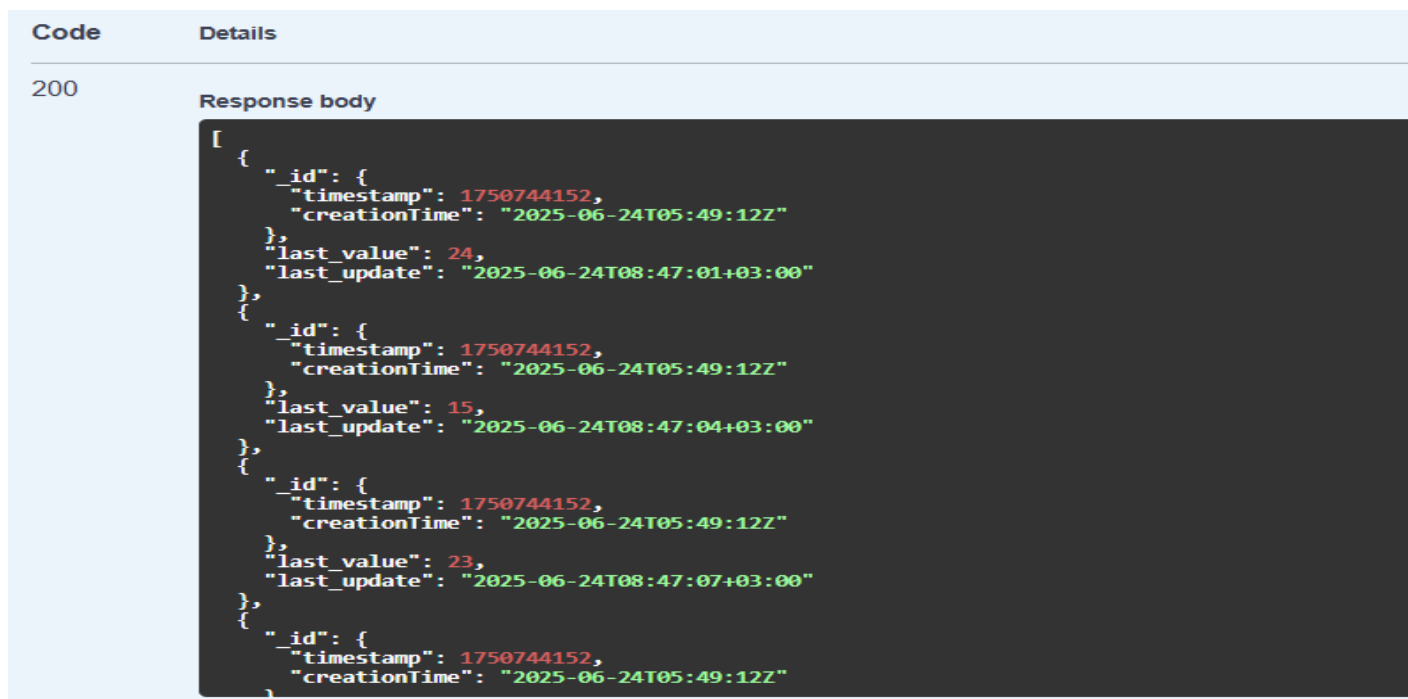


Рисунок В.11 – Функциональный тест №11

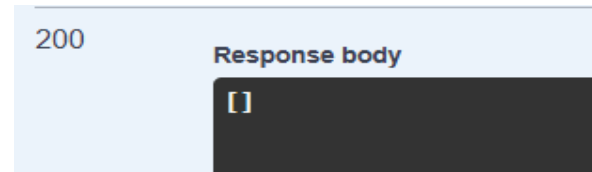


Рисунок В.12 – Функциональный тест №12

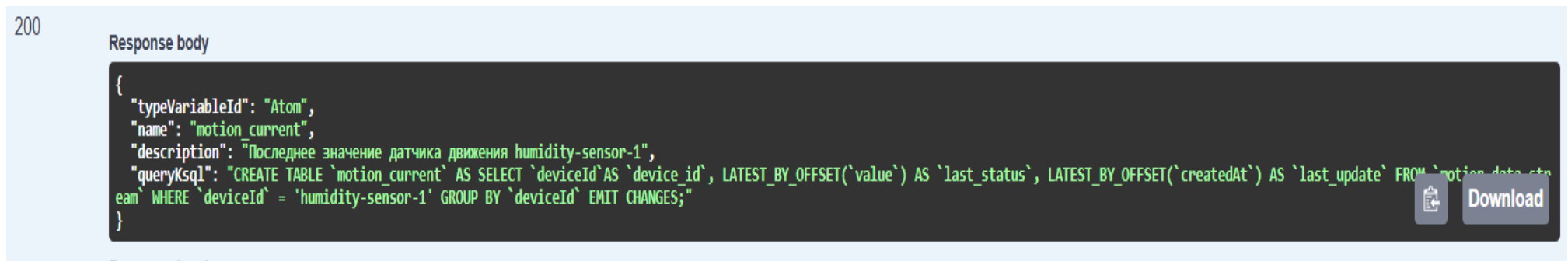


Рисунок В.13 – Функциональный тест №13

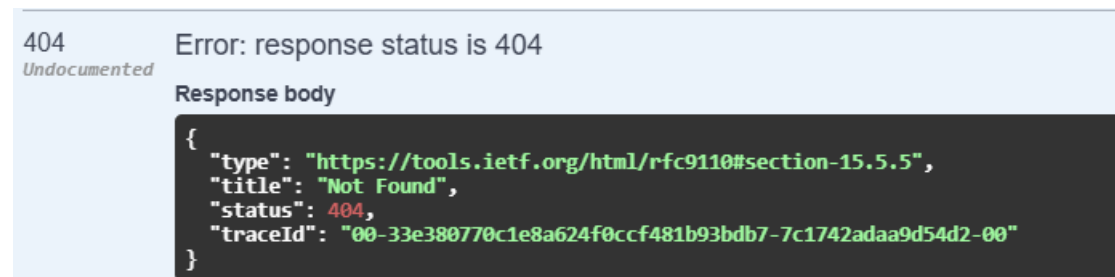


Рисунок В.14 – Функциональный тест №14

```

200
Response body
{
  "success": true,
  "connectorName": "connector_motion_current_motion_sens_1",
  "result": {
    "name": "connector_motion_current_motion_sens_1",
    "config": {
      "connector.class": "com.mongodb.kafka.connect.MongoSinkConnector",
      "topics": "motion_current_motion_sens_1",
      "connection.uri": "mongodb://host.docker.internal:27017",
      "database": "iot-devices",
      "collection": "motion_current_motion_sens_1",
      "writes.mode": "upsert_document",
      "change.data.capture.handler": "",
      "key.converter": "org.apache.kafka.connect.storage.StringConverter",
      "value.converter": "org.apache.kafka.connect.json.JsonConverter",
      "value.converter.schemas.enable": "false",
      "name": "connector_motion_current_motion_sens_1"
    },
    "tasks": [],
    "type": "sink"
  }
}

```

Рисунок В.15 – Функциональный тест №15

```

400
Undocumented
Error: response status is 400
Response body
Table 'motion_current_motion_sens_' does not exist in ksqlDB

```

Рисунок В.16 – Функциональный тест №16

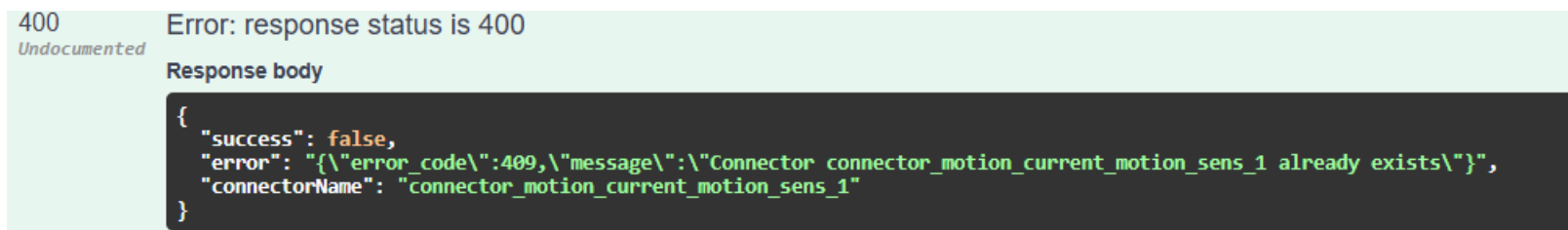


Рисунок В.17 – Функциональный тест №17