

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет компьютерных наук

Кафедра программирования и информационных технологий

РАЗРАБОТКА ЯДРА IOT-ПЛАТФОРМЫ ДЛЯ СОЗДАНИЯ
ПОТРЕБИТЕЛЬСКИХ И ПРОМЫШЛЕННЫХ ПРИЛОЖЕНИЙ ИНТЕРНЕТА ВЕЩЕЙ

ВКР Магистерская диссертация
09.03.02 Разработка мобильных приложений и компьютерных игр
Информационные системы и сетевые технологии

Допущено к защите в ГЭК

Зав. кафедрой _____ С.Д. Махортов, д.ф.-м.н, доцент _____.20__

Обучающийся _____ А.О. Бородин, 2 курс (маг), д/о

Руководитель _____ В.С. Тарасов, ст. преподаватель.

Воронеж 2023

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Факультет компьютерных наук
Кафедра программирования и информационных технологий

УТВЕРЖДАЮ
заведующий кафедрой
_____ Махортов С.Д.
_____.____.2023

ЗАДАНИЕ

**НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
ОБУЧАЮЩЕГОСЯ БОРОДИНА АНДРЕЯ ОЛЕГОВИЧА**

1. Тема работы РАЗРАБОТКА ЯДРА IOT-ПЛАТФОРМЫ ДЛЯ СОЗДАНИЯ ПОТРЕБИТЕЛЬСКИХ И ПРОМЫШЛЕННЫХ ПРИЛОЖЕНИЙ ИНТЕРНЕТА ВЕЩЕЙ, утверждена решением учебного совета _____ факультета от _____.____.2023
2. Направление подготовки 09.03.04 Программная инженерия
3. Срок сдачи студентом законченной работы _____.____.2023
4. Календарный план:

№	Структура ВКР	Сроки выполнения	Примечание
	Введение		
	1 Постановка задачи	18.11.2022	
	2 Анализ предметной области	25.12.2022	
	3 Обзор известных аналогов	14.01.2023	
	4 Требования к разрабатываемой платформы	20.02.2023	
	5 Архитектура разрабатываемой платформы	20.02.2023	
	6 Используемые технологии		
	7 Разработанная IoT-платформа	15.03.2023	
	8 Применение разработанной платформы в различных областях и отраслях	01.03.2023	
	9 Дальнейшие планы развития платформы	12.03.2023	
	Заключение	13.03.2023	

Обучающийся _____

Бородин А.О.

Руководитель _____

Тарасов В.С.

Реферат

Магистерская диссертация 90 с., 24 рис., 27 источников.

Ключевые слова: IOT, ИНТЕРНЕТ-ВЕЩЕЙ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, KAFKA, KUBERNETES, CLICKHOUSE, QUARKUS, GOLANG, GRAFANA.

Данная работа посвящена исследованию и разработке ядра IoT-платформы для создания потребительских и промышленных приложений Интернета вещей. В работе были проведен анализ существующих IoT-платформ, выбраны подходящие технологии для реализации системы, выделены требования к системе и разработан прототип ядра IoT-платформы. Были изучены основные компоненты IoT-платформ, такие как устройства IoT, протоколы связи, хранилища данных, системы управления устройствами, а также сервисы обработки и анализа данных.

Для реализации ядра IoT-платформы были выбраны современные технологии и инструменты, такие как Kafka, ClickHouse, Postgres, Grafana, Kubernetes и другие. Ключевой ролью в разработке ядра платформы стало использование языка программирования Golang, который обеспечил высокую производительность и эффективность работы системы, а также простоту разработки и поддержки кода.

Полученные результаты исследования позволили разработать функциональное и масштабируемое ядро IoT-платформы, которое успешно прошло проверку на примере создания потребительских и промышленных приложений Интернета вещей.

Разработанное ядро IoT-платформы может быть использовано для создания приложений Интернета вещей в различных отраслях, которые позволят собирать, обрабатывать и анализировать данные от устройств IoT.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1 ПОСТАНОВКА ЗАДАЧИ.....	8
2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	10
2.1 Общая информация IoT	10
2.2 История развития IoT	11
2.3 Архитектура IoT.....	14
2.4 Технологии IoT	16
2.5 Сферы применения IoT	21
2.6 Преимущества и недостатки IoT.....	22
3 ОБЗОР ИЗВЕСТНЫХ АНАЛОГОВ.....	25
3.1 Rightech IoT Cloud	25
3.2 Yandex IoT Cloud	26
3.3 EMG IoT Platform.....	27
4 ТРЕБОВАНИЯ К РАЗРАБАТЫВАЕМОЙ ПЛАТФОРМЕ	29
5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПЛАТФОРМЫ	30
6 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ	31
6.1 Протоколы обмена данными	31
6.2 Golang.....	36
6.3 Quarkus – Java-фреймворк	37
6.4 Базы данных	38
6.5 Apache Kafka	40
6.6 Grafana.....	41
6.7 Python	42
6.8 Keycloak	43
6.9 Kubernetes	43
7 РАЗРАБОТАННАЯ ИОТ-ПЛАТФОРМА.....	46
7.1 api-gateway	46

7.2	configuration-storage	47
7.3	metrics-engine	49
7.4	alarms-engine	54
7.5	policy-engine	58
7.6	external-notification-engine	60
7.7	Пользовательский интерфейс	62
8	ПРИМЕНЕНИЕ РАЗРАБОТАННОЙ ПЛАТФОРМЫ В	
	РАЗЛИЧНЫХ ОБЛАСТЯХ И ОТРАСЛЯХ	71
9	ДАЛЬНЕЙШИЕ ПЛАНЫ РАЗВИТИЯ ПЛАТФОРМЫ	86
	ЗАКЛЮЧЕНИЕ	87
	СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	88

Введение

В настоящий момент мы находимся на поворотном этапе в нашем обществе, когда мир вокруг нас глубоко погружен в интеллектуальные устройства, которые соединяются друг с другом различными способами: через проводные и беспроводные технологии. Сеть таких физических объектов, или «вещей», в которые встроены различные электронные компоненты, датчики, сенсоры для сбора информации об окружающей среде, и которые имеют возможность осуществлять выход в Интернет для передачи собранных данных, составляет основу философии Интернета вещей (Internet Of Things - IoT).

IoT-системы и предоставляемые ими решения в последнее время приобрели невероятную популярность и распространение. К 2025 году прогнозируется, что IoT-системы увеличат свое экономическое влияние на 421 млрд долларов, при этом среднегодовой темп роста составит 33% в течение 2021-2025 годов [4]. Этот значительный экономический рост является прямым результатом подключения к Интернету к 2025 году более 75 миллионов устройств, большая часть которых – повседневные предметы, используемые человеком в повседневной жизни [5].

Потенциал создания подключенных к Интернету сетей IoT-устройств огромен. Устройства Интернета вещей предлагают различные возможности, которые позволяют облегчить взаимодействие человека с машинами. Примеры решаемых IoT-системами задач относятся к различным областям жизнедеятельности человека. Например, IoT-системы помогают упростить жизнь в здравоохранении путем постоянного мониторинга жизненно важных функций человека с помощью носимых устройств. Кроме того, IoT-системы часто используют в домашней автоматизации, домашней безопасности, в сферах интеллектуальной логистики и производственного хранения. Решения в подобных сферах обладают огромным потенциалом, при этом не меньшую значимость имеет другой аспект IoT, который включает в себя подключение

машин в промышленности к Интернету, друг с другом и с рабочей силой на заводе. Эта философия лежит в основе промышленного Интернета вещей (IIoT).

Введение концепции Интернета вещей и её активное использование способствует формированию новых механизмов экономики коллективного использования, что позволяет изменить бизнес-модели на разных уровнях, вплоть до государственного. Со стороны промышленности Интернет вещей выступает в качестве посредника между сбором, анализом и хранением больших объемов информации, общением между машинами и автоматизацией производства.

1 Постановка задачи

Существующие IoT-платформы для создания приложений Интернета вещей часто ограничены в своей функциональности, не обеспечивают необходимую масштабируемость и безопасность, а также не предоставляют достаточных аналитических возможностей для эффективного управления устройствами. Кроме того, многие из этих платформ предназначены только для конкретных отраслей или типов устройств, что затрудняет их использование в различных сферах промышленности и потребительских приложениях.

Целью данной работы является разработка универсального ядра IoT-платформы с широкими функциональными возможностями, масштабируемостью, безопасностью и возможностями для мониторинга. Разработанная система может быть использована для создания и управления различными типами приложений Интернета вещей в различных отраслях и областях.

Для достижения поставленной цели и необходимо решить ряд следующих **задач**:

- 1) Провести анализ рынка IoT-платформ. Необходимо провести исследование рынка IoT-платформ, чтобы определить сильные и слабые стороны существующих решений и выявить возможности для улучшения функциональности и эффективности разрабатываемой платформы.
- 2) Определить требования к разрабатываемому ядру IoT-платформы. На основе результатов анализа рынка и потребностей конечных пользователей необходимо определить основные требования к ядру IoT-платформы, такие как масштабируемость, безопасность, управляемость и возможности мониторинга.
- 3) Разработать прототип ядра IoT-платформы. На основе выделенных требований к системе необходимо определить архитектуру

разрабатываемой системы. С учетом выбранной архитектуры, нужно разработать прототип ядра IoT-платформы.

- 4) Провести тестирование и оптимизацию ядра IoT-платформы. Создать тестовые сценарии и симуляции, которые будут имитировать реальные условия работы системы, что позволит проверить её эффективность и надежность. Кроме того, предполагается проведение пользовательских тестирований, в ходе которых реальные пользователи будут использовать систему и оценивать ее удобство, функциональность и надежность
- 5) Провести сравнительный анализ разработанного ядра IoT-платформы с аналогичными решениями на рынке с целью выявления её преимуществ и недостатков.

2 Анализ предметной области

2.1 Общая информация IoT

Определение Интернета вещей до конца не формализовано – различные мировые компании имеют взгляды на трактовку понятия IoT с различных аспектов. Например, компания Cisco расширяет концепт IoT до понятия Internet of Everything (IoE), в котором рассматриваются не только взаимодействия вида «машина-машина» (M2M), но взаимодействия вида «машина-человек» (M2P) и «человек-человек» (P2P) в рамках решения на базе Интернета вещей [9]. IBM называет IoT промышленной революцией и выделяет Промышленный IoT (Industrial IoT – IIoT), который отчасти является составной частью IoT и даёт возможность объединения производственных объектов в единую сеть [10]. Microsoft представляет IoT в виде внедрения недорогого и повсеместно используемого аппаратного оборудования, способного собирать информацию об окружающей среде и передавать её через интернет [11].

Рассмотренные понятия выделяют различные аспекты парадигмы Интернета вещей, однако все они следуют одной и той же идее, которая состоит в объединении в единую систему посредством сети Интернет большого количества устройств, предоставляющих информацию об окружающей среде в виде численных данных, с целью обеспечения взаимодействия их с внешней средой или друг с другом.

Первоначальное объяснение концепции Интернета вещей появилось как обозначение будущего использования технологий радиочастотной идентификации для взаимодействия вещей как между собой, так и с внешней средой. Дальнейшее развитие понятия Интернета вещей связано с внедрением различных технологических процессов в области информационных технологий, которые позволили упростить реализацию вышеупомянутой концепции. Причиной сегодняшнего бума Интернета вещей прежде всего предопределено прогрессов облачных технологий, использованием протокола

IPv6 для идентификации оборудования и определением легковесных и простых протоколов межмашинного общения.

2.2 История развития IoT

Концепция подключения различных устройств в единую сеть для их общения начала зарождаться еще в 19 веке, когда в 1832 году российским ученым Павлом Шиллингом был создан первый электромагнитный телеграф. Телеграф позволял обеспечить прямое общение между двумя машинами посредством передачи электрических сигналов. Тем не менее, настоящая история зарождения Интернета вещей напрямую связана с изобретением Интернета - одного из главных компонентов концепции - развитие которого началось в 1950-х и стремительно продолжалось в последующие десятилетия.

Предпосылки появления Интернета вещей были предугаданы ещё в начале прошлого века Николай Тесла. В 1926 году в интервью журналу “Кольерс” физик выразил мысль о том, что при идеальном использовании радиоволн, Земля превратится в единый “большой мозг”, который будет управлять различными физическими предметами, при этом инструменты, через которые будет достигнут данный результат, будут просты и компактны настолько, что поместятся в небольшой карман [12].

Неформальным началом истории развития концепции объединения физических объектов в единую сеть является появление в 1982 году модифицированного торгового автомата Coca-Cola, расположенного в университете Карнеги-Меллон. Данный аппарат стал первым устройством, подключенным к сети ARPANET и предоставляющий возможность удаленно наблюдать за его текущим состоянием и определять, охладился или нет недавно загруженные банки.

Официальная история развития IoT начинается в 1990 году, когда Джон Ромки, один из создателей протокола TCP/IP, создал первую в мире интернет-вещь, подключив к сети свой домашний тостер [14]. При этом он смог

приготовить тост без прямого контакта с прибором, управляя им через удаленное подключение.

Через год, двое ученых Кембриджского университета Квентин Стэффорд-Фрейзер и Пол Жардецкий пришли к идее использовать свой первый прототип веб-камеры для отслеживания количества кофе в локальной компьютерной лаборатории. Они запрограммировал камеру так, чтобы она делала снимки кофейника примерно три раза в минуту, при этом отправляла их на все локальные компьютеры, что позволило всем сотрудникам определять наличие кофе.

Такие попытки неформальной реализации концепции Интернета вещей продолжались вплоть до 1999 года. В 1999 году на презентации для руководства компании Procter&Gamble (P&G) основатель исследовательской группы при Массачусетском Технологическом институте Кевин Эштон ввёл понятие концепции Интернета вещей (Internet Of Things). Он описал данную концепцию как технологию, соединяющую несколько устройств при помощи радиочастотной идентификации (RFID-метки) для управления цепочками поставок продукции P&G. Кевин Эштон предложил использовать специальные RFID-метки в логистической сфере для идентификации товаров на складах и отслеживания их передвижения в процессе перевозки к торговым точкам. Эти RFID-метки должны были отправлять данные о своем местоположении, что позволило бы поставщикам и ритейлерам при изменении спроса и предложения отправлять нужные товары туда, где они были необходимы. Кевин намеренно использовал в своей речи слово «Интернет», чтобы привлечь внимание аудитории, так как в то время Интернет только становился большим событием. В том же году был создан Центр автоматической идентификации, рассматривающий вопросы сенсорных технологий и радиочастотной идентификации, благодаря которому концепция получила широкое развитие и распространение. Несмотря на то, что идея подключения устройств на основе радиочастотной идентификации отличается

от текущей реализации IoT на основе IP, прорыв Эштона сыграл важную роль в истории развития Интернета вещей и технологической революции в целом.

В начале 20-го века термин «Интернет вещей» получил широкое признание в средствах массовой информации, о нем упоминали такие гигантские издания, как The Guardian, Forbes и Boston Globe.

Аналитики компании Cisco Systems определяют «настоящим рождением Интернета вещей» период 2008-2009 годов, определяя Интернет вещей как «просто момент времени, когда к Интернету подключено больше физических объектов (вещей), чем людей». Таким образом произошел переход от «Интернета людей» к «Интернету вещей».

Интерес неумолимо возрастал, что привело к организации при поддержке Европейской комиссии в 2008 году в Брюсселе 1-й ежегодной Международной конференции «Internet of Things». Вот уже какой год, на данной конференции доклады зачитываются доклады чиновников стран Евросоюза, еврокомиссаров и ведущих ученых исследовательских институтов.

Бум Интернета вещей был поддержан добавлением его в цикл хайпа (Gartner Hype) для новых технологий в 2011 году. Через год, в 2012 году состоялся запуск интернет-протокола IPv6, активно используемого в IoT для идентификации устройств.

На протяжении следующих лет соединение в единую сеть различных физических объектов для обеспечения их взаимодействия между собой стало обычным явлением в повседневной жизни. На текущий момент, глобальные технологические титаны, такие как Apple, Samsung, Google, Cisco реализуют свои интересы в сфере Интернета вещей. Всего менее чем за 20 лет Интернет вещей стал мировым трендом в области информационных технологий, пройдя путь от радиометок, наносимых на складские ящики, до глобальной интернетизации и оцифровки всевозможных датчиков и устройств или так называемых “вещей”.

2.3 Архитектура IoT

Базовой архитектурой IoT, которая довольно точно передаёт концепцию Интернета вещей, является трехслойная архитектура. Она была предложена на ранних этапах исследований в сфере Интернета вещей [7]. На рисунке 1 **Ошибка! Ключ шаблона должен быть первым ключом форматирования.** представлена трехуровневая, которая предполагает наличие в системе трех уровней: слой восприятия (perception layer), сетевой уровень (network layer) и прикладной уровень (application layer).

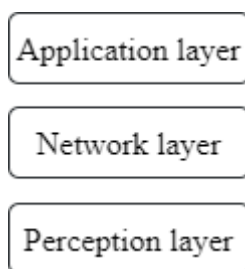


Рисунок 1 – Трехуровневая архитектура Интернета вещей

Слой восприятия представляет собой физический уровень IoT, реализующий “кожу и 5 органов чувств” системы. На данном уровне расположены средства, осуществляющие идентификацию вещей, а также датчики и сенсоры, которые выполняют функцию сбора данных об окружающей среде.

Сетевой уровень является “нейронной сетью и мозгом” IoT. Данный уровень отвечает за соединение одних вещей друг с другом, а также за передачу данных между ними и обработку этих данных.

Прикладной уровень ответственен за доставку решений Интернета вещей конечному пользователю. Иными словами, данный уровень определяет ряд приложений, которые, основываясь на данных, полученных с сетевого уровня, реализуют интеллектуализацию различных процессов сфер жизнедеятельности.

Несмотря на то, что трехслойная архитектура довольно точно определяет основную идею Интернета вещей, её недостаточно для углубленного

исследования, так как исследования часто акцентируются на более тонких аспектах. Решить данную проблему попытались решить китайские исследователи Ву и др. - проанализировав слои архитектуры протокола ТСР/ІР, они расширили архитектуру ІоТ до пяти уровней, что позволило более углубленно изучать принципы ІоТ при исследовании и разработки системы Інтернета вещей [7].

Предложенная архитектура показана на рисунке 2 и состоит из пяти уровней: слой восприятия (perception layer), транспортный слой (transport layer), слой обработки (processing layer), прикладной уровень (application layer) и бизнес слой (business layer).

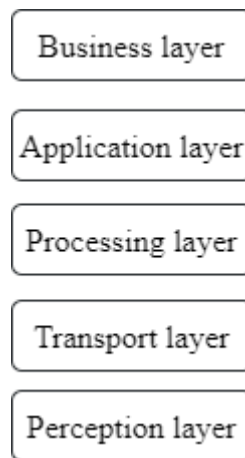


Рисунок 2 – Пятиуровневая архитектура Інтернета вещей

Роль уровня восприятия немного сократилась - теперь он ответственен только за сбор данных о физической среде и преобразовании их в цифровые сигналы, удобные для сетевой передачи.

Роль прикладного уровня осталась такая же, как и в трехуровневой архитектуре.

Транспортный уровень предназначен для передачи данных, которые были собранными датчиками и сенсорами, расположенными на уровне восприятия, в слой обработки и обратно путем использования различных проводных и беспроводных сетевых технологий, таких как WiFi, LAN, 4G, Bluetooth, NFC и другие.

Слой обработки также известен как слой промежуточного программного обеспечения. Главная роль данного уровня состоит в хранении, анализе и обработке огромных объемов данных, поступающий с транспортного уровня.

Бизнес слой контролирует IoT систему в целом, предоставляя как интерфейсы для управления отдельными приложениями, так и различные аналитические данные, позволяющие пользователю определять эффективность системы и на какие проблемы стоит обратить внимание в первую очередь.

2.4 Технологии IoT

В данном разделе рассмотрим технологии, которые активно используются в Интернете вещей для обеспечения функционирования технологии, а именно средства идентификации и средства передачи данных.

2.4.1 Идентификация устройств

В любой системе взаимодействующих элементов идентификация компонентов этой системы необходима для обеспечения правильной структуры и работы системы. Это относится ко всем этапам жизненного цикла системы от разработки до сборки, ввода в эксплуатацию, эксплуатации, обслуживания и даже окончания срока службы.

В общем смысле, идентификатор – это шаблон для однозначного определения отдельной сущности (идентификатор экземпляра) или класса сущностей (идентификатор типа) в определенном контексте.

Для устройств, которые не имеют подключения к сетям передачи данных, должны использоваться специальные средства для осуществления идентификации. В качестве подобных средства могут использоваться различные оптически распознаваемые матричные идентификаторы (QR-коды, штрих-коды) и алгоритмы определения местонахождения в режиме реального времени.

Для объектов, непосредственно подключенных к сети, основным идентификатором является MAC-адрес сетевого устройства, позволяющий

определить устройство на канальном уровне. Причем уникальность каждого адреса достигается тем, что диапазон в стандарте MAC-48 практически неисчерпаем (248 адресов) и, согласно подсчетам IEEE, их хватит до 2100 года. Более широкие возможности для идентификации устройств предлагает пришедший на смену протокола IPv4 интернет-протокол IPv6, позволяющий обеспечить уникальными адресами сетевого уровня не менее 300 млн устройств на одного жителя Земли.

2.4.2 Средства передачи данных

Передача данных с устройств, отслеживающих состояние окружающей среды, также является основным вопросом, решаемым в задаче Интернета вещей. Рассмотрим сетевые протоколы передачи данных, широко используемые в концепции Интернет вещей.

Для беспроводной передачи данных особо важную роль в построении «интернета вещей» играют такие качества, как эффективность в условиях низких скоростей, отказоустойчивость, адаптивность, возможность самоорганизации.

Для сетей ближнего действия наиболее часто используют следующие спецификации:

- NFC – набор протоколы связи между двумя устройствами при их близком контакте (на расстоянии 4 см или менее)
- RFID – технология радиочастотной идентификации, использующая электромагнитные поля для чтения информации, хранящейся в тегах, которые встроены в объекты
- Wi-Fi – технология локальной сети, основанная на стандарте IEEE 802.11, в которой устройства обмениваются данными через общую точку доступа;
- ZigBee – набор коммутационных протоколов для персональных сетей, основанный на стандарте IEEE 802.15.4, обеспечивающий низкое

потребление, низкую скорость передачи данных, низкую стоимость и высокую пропускную способность.

Для беспроводных сетей среднего радиуса действия в данный момент часто используют следующие технологии:

- LTE-Advanced – спецификация высокоскоростной связи для мобильных сетей. Стандарт обеспечивает усовершенствование стандарта LTE с расширенным покрытием, более высокой пропускной способностью и меньшей задержкой;
- 5G – беспроводные сети стандарта 5G могут использоваться для удовлетворения высоких требований к передаче данных IoT и подключения большого количества вещей, даже если они находятся в движении.

Для беспроводных сетей большого радиуса действия наиболее распространены следующие технологии:

- LPWAN (LoRaWan, NB-IoT) – глобальные сети с низким потреблением. Обеспечивают передачу данных на большие расстояния с низкой скоростью передачи данных, что снижает мощность и стоимость передачи;
- VSAT – технология спутниковой передачи данных с использованием небольших параболических антенн для узкополосных и широкополосных данных.

Среди проводных технологий передачи данных, которые играют наиболее важную роль в решениях IoT, являются технологии PLC – коммутационная технология, использующая электрическую проводку и линии электропередач для передачи данных, так как во многих приложениях имеется доступ к электросетям (например, банкоматы, умные счетчики, контроллеры освещения).

2.4.3 Протоколы передачи данных

Прикладные протоколы передачи данных для IoT были разработаны в качестве решения проблемы ненадежных сетей связи. Данная необходимость возникла в мире Интернета вещей из-за увеличения количества небольших, дешевый, маломощных и низкоэлектропотребляемых устройств, появившихся в сети за последние несколько лет.

В качестве прикладных телеметрических протоколов передачи данных в концепции Интернета вещей, используемых на программном уровне, используются следующие протоколы: MQTT, CoAP, AMQP, DDS, WebSockets и HTTP. По порядку рассмотрим краткую характеристику перечисленных протоколов.

MQTT (Message Queuing Telemetry Transport) - легковесный стандарт передачи данных, разработанный специально для IoT систем. Он поддерживает модель обмена сообщениями между устройствами - издателем и подписчиком - и обеспечивает простой поток данных между различными устройствами.

Основным преимуществом MQTT является его архитектура. Он работает поверх протокола TCP/IP, а его архитектура проста и легка, что дает возможность обеспечивать низкое потребление устройств, что очень важно обеспечить в сфере Интернета вещей.

Протокол CoAP (Constrained Application Protocol) разработан для удовлетворения потребностей Интернета вещей на основе протокола HTTP. Несмотря на то, что существующая структура Интернета находится в свободном доступе и может использоваться различными IoT устройствами, она слишком тяжелая и энергоемкая, что не позволяет повсеместно её использоваться. Поэтому многие решения отвергли HTTP протокол как протокол, не подходящий для Интернета вещей.

Протокол CoAP устранил данную проблему, переведя модель HTTP на использование в ограниченных устройствах и сетевых средах. Данный протокол имеет невероятно низкие накладные расходы, прост в использовании и может использоваться для многоадресной рассылки.

AMQP (Advanced Message Queuing Protocol) представляет собой открытый протокол прикладного уровня, используемый для транзакционных сообщений между серверами. Данный протокол структурно похож на протокол MQTT и в концепции Интернета вещей выполняет следующие функции:

- Получение и размещение сообщений в очередях
- Хранение сообщений
- Установление связей отношений между компонентами

Обладая уровнем безопасности и надежности, протокол AMQP чаще всего используется в условиях, когда требуется серверная аналитическая среда, например в банковской сфере. Однако в других местах он широко не используется. Из-за своего веса он не подходит для сенсорных устройств Интернета вещей с ограниченной памятью. В результате его использование в мире Интернета вещей все еще весьма ограничено.

DDS (Data Distribution Service) обеспечивает высококачественную и масштабируемую связь в IoT. DDS, как MQTT и AMQP, работает по модели издатель-подписчик.

Протокол DDS можно развернуть в самых разнообразных средах, от облака до очень маленьких устройств. Это делает его идеальным для систем реального времени и встроенных систем. Более того, в отличие от MQTT, протокол DDS позволяет осуществлять совместный обмен данными независимо от аппаратной и программной платформы. Фактически, он считается первым открытым международным стандартом IoT промежуточного программного обеспечения.

Websockets изначально был разработан еще в 2011 году в рамках инициативы протокола HTML5. Через одно TCP-соединение данных протокол позволяет осуществлять отправку сообщений между клиентом и сервером. Как и CoAp, стандартный протокол подключения WebSocket помогает

упростить многие сложности и трудности, связанные с управлением подключениями и двунаправленной связью в Интернете.

Протокол Websockets успешно применяется в сети IoT, где данные непрерывно передаются между несколькими устройствами, которые действуют как клиент и сервер.

2.5 Сферы применения IoT

Для применения Интернета вещей больше всего подходят сферы и организации, которым было бы полезно использовать датчики и сенсоры в своих бизнес-процессах.

Любое производство может получить конкурентное преимущество, используя датчики и сенсоры для мониторинга производственной линии для предупреждающего обслуживания оборудования, когда измерительные приборы обнаруживают надвигающийся отказ. С помощью таких предупреждений предприятия могут постоянно проверять состояние оборудования и заранее снимать его с производства для ремонта, таким образом избежав неожиданной остановки работы и потерь финансов.

В автомобильной промышленности помимо преимуществ постоянного мониторинга на производственных линиях, датчики и сенсоры, заранее установленные в автомобилях, могут обнаруживать приближающийся отказ оборудования в транспортных средствах, уже находящихся на дороге, при этом уведомляя пользователя подробностями неисправности и рекомендациями по устранению. А благодаря анализу собираемой с этих датчиков информации, производители могут улучшить качество производство и работу автомобилей.

В транспортной сфере и логистике IoT также позволяет получить выгоду от использования. Грузовые транспорты, которые перевозят товары, могут быть перенаправлены в другое место в зависимости от погодных условий, дорожной обстановки, наличия свободных транспортных средств или доступности водителей, информация о которых собирается с помощью

датчиков и сенсоров Интернета вещей. Кроме того, сами транспортные средства могут быть оборудованы датчиками для мониторинга состояния перевозимого груза и избежания ситуации, когда этот груз будет испорченным. Это особенно важно в пищевой и фармацевтической промышленности, где часто перевозятся грузы, чувствительные к температуре, влажности.

В розничной торговле приложения IoT позволяют розничным компаниям качественно управлять запасами, улучшать качество обслуживания клиентов, оптимизировать цепочку поставок и сокращать операционные расходы. Например, датчики веса, расположенные на полках, могут собирать информацию и отправлять данные для автоматического отслеживания запасов продукции и отправки уведомлений, если эта продукция заканчивается.

Государственный сектор может использовать решения IoT для уведомления своих пользователей о массовых отключениях воды или даже о небольших перебоях в подаче воды, электроэнергии или канализации. Со стороны пользователей могут устанавливаться умные счетчики, позволяющие автоматически отсылать данные управляющей компании без участия человека.

Активно используется IoT в отрасли здравоохранения. Врачам, медсестрам необходимо постоянно знать местоположение своих пациентов, средств для оказания помощи. Кроме того, использование датчиков и сенсоров для постоянного мониторинга позволяет наладить эффективное и быстрое реагирование врачей при экстренных ситуациях.

2.6 Преимущества и недостатки IoT

2.6.1 Преимущества

С развитием Интернета вещей все больше и больше устройств и предметов будут подключаться к глобальной сети, тем самым создавая новые возможности в сфере безопасности, аналитики и управления, открывая новые

и более широкие перспективы и способствуя повышению качества жизни населения. Рассмотрим положительные стороны Интернета вещей.

Повышение производительности. Производительность является ключом к успеху в бизнесе и во многом зависит от производительности сотрудников. Интернет вещей обеспечивает обучение сотрудников организации в режиме реального времени и помогает им эффективно общаться во время работы. Интернет вещей для бизнеса также дает возможность автоматизации с помощью различных подключенных устройств, которые отслеживают, управляют и контролируют различные операции в более короткие сроки.

Сбор данных. Интернет вещей дал производствам и организациям возможность собирать важные данные, которые могут улучшить их продукты или услуги. Сбор данных, вероятно, является самым большим преимуществом Интернета вещей после автоматизации и может дать важную информацию, например, о поведении потребителей.

Аналитика в реальном времени. Интернет вещей может помочь предприятиям заранее обнаруживать проблемы и соответствующим образом на них реагировать. Готовые к работе приложения Интернета вещей в режиме реального времени позволяют повысить эффективность за счет экономии времени, затрачиваемого на анализ проблем и принятие корректирующих мер.

Оптимизация затрат. Управление расходами без ущерба для производительности - цель каждой организации. Для этого вам необходимо рабочее оборудование, продуктивные сотрудники и удовлетворенность клиентов. Благодаря Интернету вещей для бизнеса и его влиянию на повышение производительности и использования активов вы можете добиться успеха с минимальными затратами.

2.6.2 Недостатки

Как и каждая новая и довольно сложная технология, Интернет вещей имеет свои недостатки и уязвимости. Среди них можно указать следующие:

Отсутствие единых стандартов. Миллионы устройств связаны друг с другом в экосистеме Интернета вещей, однако все эти устройства созданы разными производителями, что поднимает вопрос совместимости при соединении этих устройств в единую. Эта проблема совместимости может вынудить покупателей или пользователей покупать устройства у определенного производителя, что может создать монополию на рынке.

Конфиденциальность и безопасность. Поскольку системы IoT взаимосвязаны и обмениваются данными по сетям, где необходимо уделять особое внимание безопасности. Нарушение данных любого уровня может серьезно нарушить личную жизнь человека. Чтобы предотвратить утечку данных, устройства Интернета вещей должны обмениваться информацией с первоклассным шифрованием. Это определенно займет некоторое время.

Сложность. Интернет вещей — это сложная и разнообразная сеть, что означает, что в приложениях IoT могут возникнуть различные сбои или отказы. Отказ в работоспособности одного модуля может привести к серьезным потерям, так как он может привести к отказу всей системы.

Безработица. С автоматизацией каждой задачи потребность в человеческом труде резко снизится. Это окажет прямое влияние на возможность трудоустройства. По мере того, как прогресс развития Интернета вещей движется в будущее, процесс найма профессионалов будет заметно сокращаться.

Зависимость от технологий. Интернет вещей оказывает влияние на жизнь почти каждого человека всеми возможными способами. С помощью Интернета вещей эта зависимость станет еще больше в повседневной жизни.

3 Обзор известных аналогов

3.1 Rightech IoT Cloud

Rightech IoT Cloud — это IoT-платформа, которая предназначена для разработки и развертывания IoT-приложений в промышленных и коммерческих секторах. Ниже приведены некоторые преимущества и недостатки этой платформы.

3.1.1 Преимущества

Простота использования: Rightech IoT Cloud имеет простой и интуитивно понятный пользовательский интерфейс, который позволяет пользователям быстро создавать, развертывать и управлять своими IoT-приложениями без необходимости знать сложные программные языки.

Расширяемость: Rightech IoT Cloud предоставляет множество готовых функций и интеграций, которые позволяют пользователям легко интегрировать различные устройства IoT и использовать различные протоколы связи.

Быстрота разработки: Rightech IoT Cloud позволяет пользователям быстро создавать и развертывать IoT-приложения, благодаря чему они могут быстро реагировать на изменения в бизнес-среде и удовлетворять потребности своих клиентов.

Масштабируемость: Rightech IoT Cloud позволяет пользователям легко масштабировать свои IoT-приложения в зависимости от их роста и потребностей, без необходимости инвестировать в дополнительную аппаратную инфраструктуру.

3.1.2 Недостатки

Ограниченный функционал: Rightech IoT Cloud не обладает всеми функциями и интеграциями, которые могут понадобиться для разработки сложных IoT-приложений. Например, не поддерживается работа с определенными протоколами связи или устройствами IoT.

Высокая стоимость: Rightech IoT Cloud имеет относительно высокую цену по сравнению с другими IoT-платформами, что может быть неприемлемо для многих пользователей.

Неудобство использования собственных API: Использование собственных API для интеграции с Rightech IoT Cloud может быть неудобным, поскольку документация не всегда ясна и подробна.

3.2 Yandex IoT Cloud

Yandex IoT Core — это IoT-платформа, разработанная компанией Яндекс для создания и управления приложениями Интернета вещей. Ниже приведены некоторые преимущества и недостатки этой платформы.

3.2.1 Преимущества

Гибкость: Yandex IoT Core позволяет настраивать и адаптировать свои приложения под конкретные потребности, предоставляя широкий набор готовых инструментов и возможностей настройки.

Безопасность: Yandex IoT Core предоставляет надежную защиту от угроз безопасности, таких как взломы и кибератаки, благодаря шифрованию трафика и механизмам аутентификации и авторизации.

Простота использования: Yandex IoT Core имеет интуитивно понятный пользовательский интерфейс, что позволяет пользователям быстро создавать и управлять своими IoT-приложениями.

Высокая масштабируемость: Yandex IoT Core легко масштабируется при необходимости, благодаря чему пользователи могут быстро расширять свои приложения и увеличивать количество устройств, подключенных к платформе.

3.2.2 Недостатки

Ограниченный выбор интеграций: Yandex IoT Core может быть ограничен в интеграции с другими системами и сервисами, что может быть

проблемой для пользователей, которые хотят интегрировать свои приложения с другими существующими системами.

Ограничения на объемы передачи данных: Yandex IoT Core может предоставлять ограниченные объемы передачи данных, что может быть неприемлемо для пользователей, которые обрабатывают большие объемы данных или имеют приложения, которые требуют высокой пропускной способности.

Высокая стоимость: Yandex IoT Core может быть дорогой для пользователей, особенно для тех, кто создает небольшие IoT-приложения.

3.3 EMG IoT Platform

EMG (Energomonitoring Group) IoT Platform — это платформа Интернета вещей, разработанная компанией Energomonitoring Group, которая предоставляет решения для мониторинга и управления потреблением энергоресурсов.

3.3.1 Преимущества

Гибкость: система может быть настроена на работу с различными устройствами и протоколами связи.

Масштабируемость: система может масштабироваться от небольших предприятий до крупных промышленных комплексов.

Высокая точность измерений: система предоставляет точную информацию о потреблении энергоресурсов.

Наличие функции автоматического контроля и управления: система позволяет автоматически контролировать и управлять потреблением энергоресурсов на основе заданных параметров.

Интеграция с другими системами: система может интегрироваться с другими системами управления, такими как системы управления зданиями, системы безопасности и т.д.

3.3.2 Недостатки

Высокая стоимость: система имеет высокую стоимость, что может быть препятствием для небольших предприятий.

Ограниченность функциональности: система имеет ограниченный набор функций и возможностей по сравнению с другими IoT-платформами.

Необходимость квалифицированных специалистов: система требует наличия квалифицированных специалистов для установки, настройки и обслуживания.

Недостаток гибкости при изменении условий эксплуатации: система может оказаться недостаточно гибкой для адаптации к изменяющимся условиям эксплуатации.

4 Требования к разрабатываемой платформе

Проанализировав имеющиеся аналоги на рынке, было принято решение создать ядро IoT-платформы, которое можно будет использоваться для создания приложения потребительского и промышленного отраслей Интернета Вещей.

Разрабатываемая система должно иметь возможность осуществлять сбор данных по различными телеметрическим протоколам, обрабатывать эти данные и выполнять различные действия на основе результата обработки данных.

В качестве функциональных требований были выделены следующие пункты:

- 1) Возможность подключения к широкому спектру устройств и сенсоров, использующих различные протоколы связи и форматы данных.
- 2) Возможность обработки полученных данных, включая агрегирование, фильтрацию, преобразование и другие операции.
- 3) Возможность отправки данных в сторонние сервисы и системы, включая базы данных, облачные сервисы, мессенджеры и другие каналы связи.
- 4) Возможность работы с аналитическими инструментами для обработки и анализа полученных данных.
- 5) Поддержка интеграции со сторонними системами и сервисами через API и другие протоколы.

5 Архитектура разрабатываемой платформы

Архитектура разрабатываемой платформы отображена на рисунке 3.

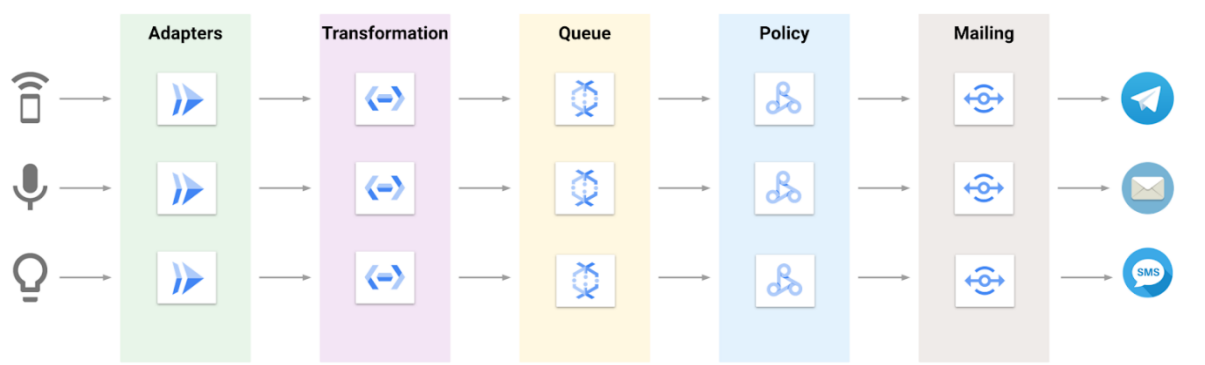


Рисунок 3 – Архитектура разрабатываемой платформы

Данные с устройств и сенсоров считываются по различным телеметрическим протоколам связи и передаются в систему через адаптеры.

После получения данных их нужно привести к единому виду в системе для удобства работы с ними, т.е. произвести их трансформацию.

Затем эти данные будут отправляться в очереди сообщений для обмена с другими компонентами системы.

Далее сообщения с очередей считываются и обрабатываются различными политиками.

И последним этапом работы системы является рассылка в сторонние сервисы.

6 Используемые технологии

6.1 Протоколы обмена данными

В качестве протоколов обмена данными в рассматриваемой IoT-системе была реализована поддержка протоколов MQTT и STOMP.

6.1.1 MQTT

MQTT — это протокол передачи данных, основанный на принципе издатель-подписчик (publish/subscribe) и реализованный поверх TCP/IP. Протокол был создан для передачи информации, где не требуется передавать большие сообщения (например, данные, собираемые сенсором) и есть ограничения по пропускной способности канала.

В процессе передачи данных участвуют издатель (publisher), брокер (broker) и подписчик (subscriber).

- Издателями являются устройства, которые осуществляют отправку сообщений.
- Брокер — центральный узел в протоколе MQTT, отвечающий за взаимодействие между издателем и подписчиком. MQTT-брокер занимается принятием сообщений от издателей, обработкой полученных данных, передачей сообщений подписчикам и контролем их доставки.
- Подписчиками являются конечные получатели данных от издателей.

Схема передачи сообщений в протоколе MQTT показана на рисунке 4.

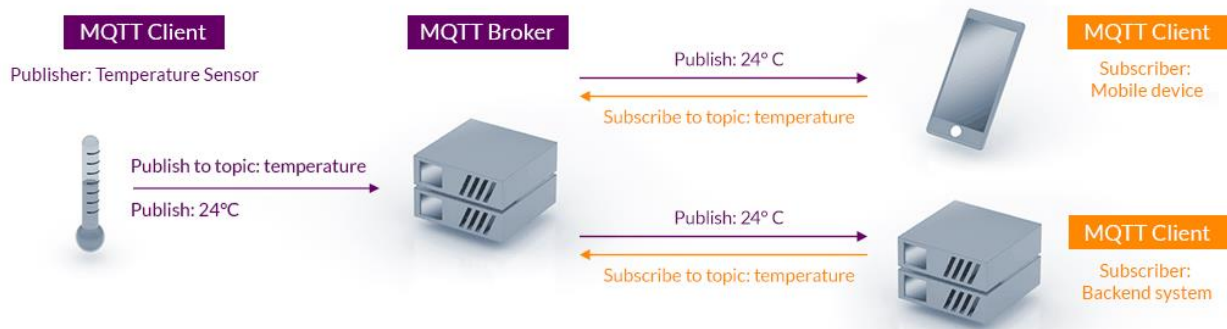


Рисунок 4 – Передача сообщений в протоколе MQTT

Издатель отправляет данные на MQTT брокер, указывая в сообщении определенную тему, именуемую в протоколе MQTT топиком (Topic). Подписчики могут получать разные данные от множества издателей в зависимости от подписки на соответствующие топики.

Топики представляют собой строку в кодировке UTF-8. При этом каждый топик имеет иерархическую структуру дерева, что позволяет упростить организацию и доступ к данным. Уровни дерева разделены между собой косой чертой (например, /hotel/floor5/room2/temperature).

Подписчики могут принимать данные как с конкретного топика, так и сразу с нескольких топиков. Для этого используется механизм wildcard, который бывает двух типов: одноуровневый и многоуровневый. Одноуровневый wildcard обозначается символом “+” и используется для получения данных с конкретных топиков на определенном уровне (например, /hotel/floor5+/temperature - получение данных со всех датчиков температуры на пятом этаже). Многоуровневый wildcard обозначается символом “#” и позволяет получить данные со всех топиков на конкретном уровне (например, /hotel/floor3/room2/# - получение данных со всех датчиков во второй комнате на третьем этаже).

На рисунке 5 представлена диаграмма последовательности общей работы протокола MQTT.

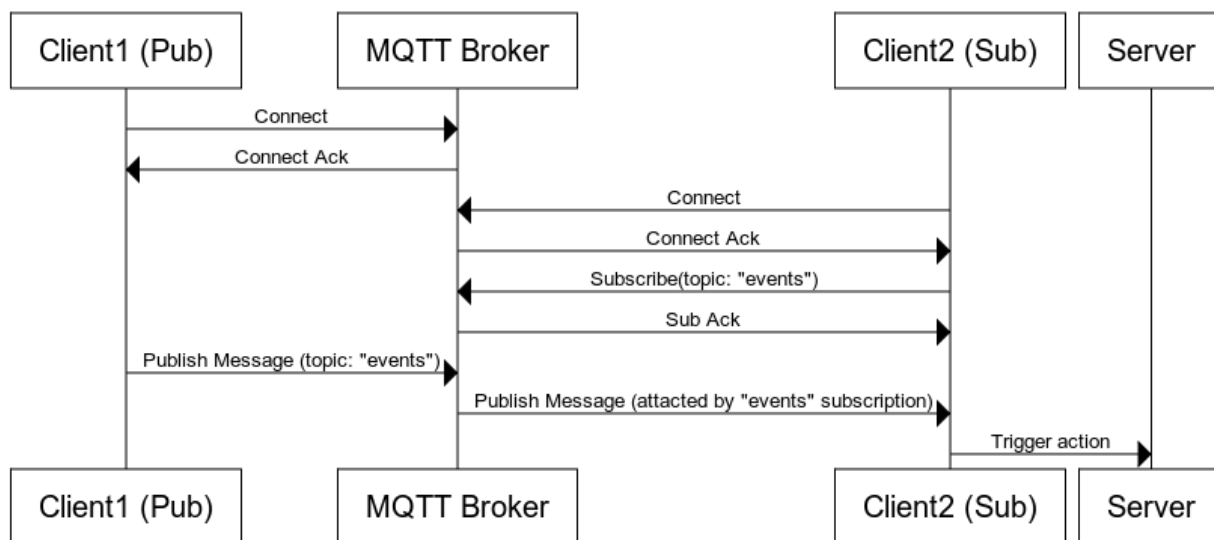


Рисунок 5 – Диаграмма последовательности работы протокола MQTT

Сообщение, отправляемое в протоколе MQTT, состоит из нескольких частей: фиксированный заголовок (обязательно присутствует во всех сообщениях), переменный заголовок и сама полезная нагрузка (могут присутствовать в некоторых сообщениях). Структура MQTT сообщения приведена на рисунке 6.

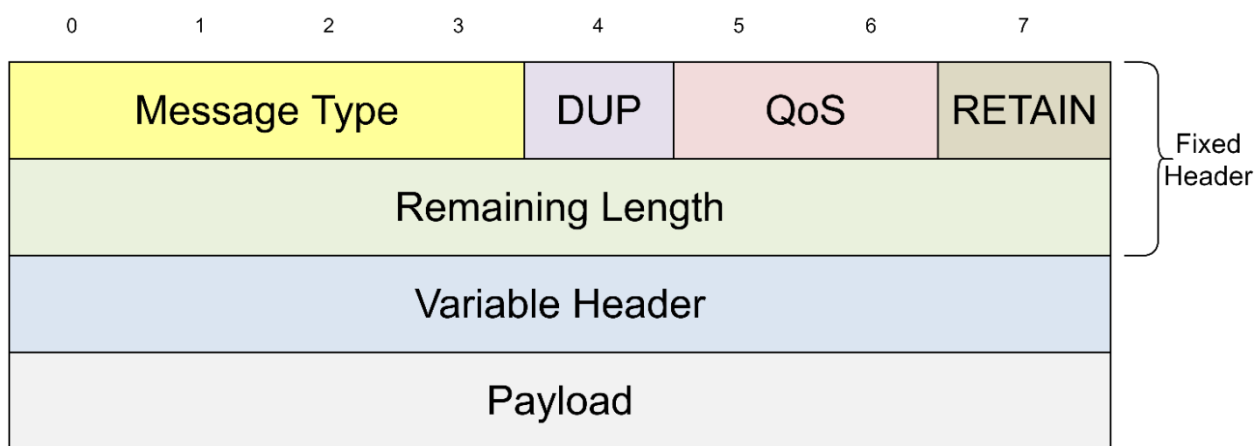


Рисунок 6 – Структура MQTT сообщения

Протокол MQTT поддерживает три уровня качества обслуживания при отправке сообщений:

- QoS 0 (At most once). На данном уровне издатель отправляет одно сообщение, не заботясь о доставке сообщения подписчикам.

- QoS 1 (At least once). Уровень гарантирует доставку сообщения брокеру, но есть вероятность дублирования информации от издателя.
- QoS 2 (Exactly once). Гарантированная отправка только одного сообщения, исключая дублирования информации.

Для защиты передаваемой информации используется несколько методов:

- Использование в заголовке полей Username и Password при подключении к MQTT-брокеру.
- Контроль брокером доступа к подключению через Client ID
- Подключение к брокеру по протоколам TLS/SSL

В качестве MQTT-брокера для поддержки протокола MQTT в разрабатываемой IoT-системе был выбран EMQX-брокер — масштабируемый и надежный MQTT-брокер с открытым исходным кодом, позволяющий гибко, быстро и просто настраивать пользователей, топики и ACL-правила путём HTTP запросов.

6.1.2 STOMP

В качестве второго поддерживаемого в системе протокола передачи данных был выбран STOMP (Simple Text Orientated Messaging Protocol). STOMP — это простой текстовый протокол взаимодействия, работающий поверх Websockets и разработанный для асинхронной передачи сообщений между клиентами через посреднические серверы, именуемые брокерами. На рисунке 7 представлена диаграмма последовательности подписки клиента на STOMP-брокер.

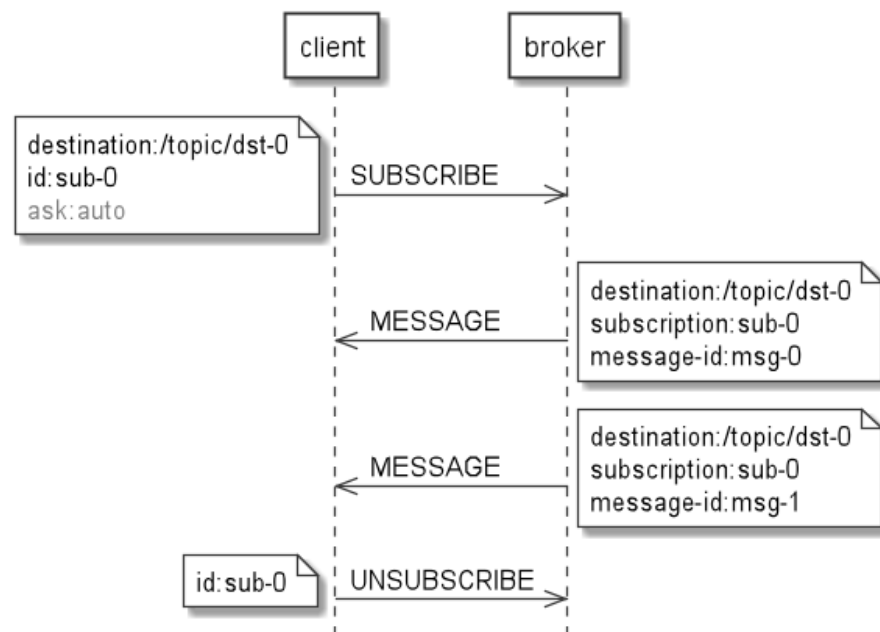


Рисунок 7 – Диаграмма последовательности работы протокола STOMP

Данный протокол осуществляет передачу данных через фреймы (Frame), которые сформированы по образцу HTTP. Диаграмма последовательности отправки STOMP-фрейма на STOMP-брокер представлена на рисунке 8. Каждый фрейм состоит из команды, необязательных заголовков и необязательного тела для полезной нагрузки:

```

COMMAND
header1:value1
header2:value2
Body^@
  
```

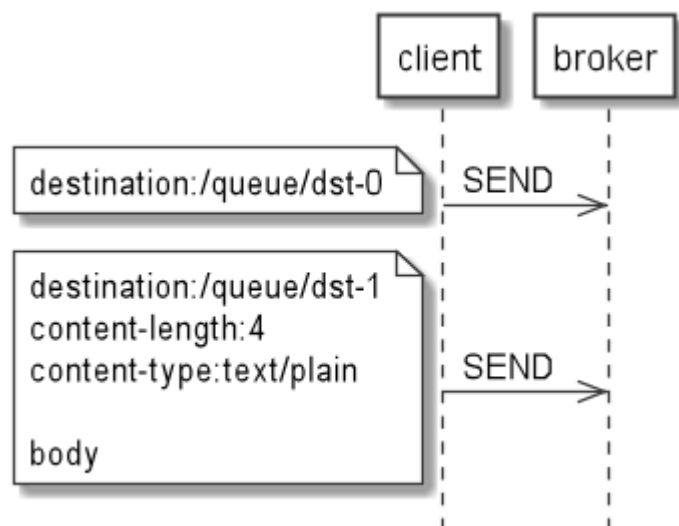


Рисунок 8 – Отправка STOMP-фрейма

STOMP-брокер для разрабатываемой системы реализован на языке Java с использованием асинхронного веб-фреймворка Vert.x, работающего на событийно-ориентированной архитектуре и запускаемого поверх виртуальной машины Java.

6.2 Golang

Почти все микросервисы в реализованной IoT-системе написаны на языке Go.

Go (или Golang) — язык программирования, созданный компанией Google в 2007 году. Цель создания языка Go заключалась в упрощении разработки масштабируемых и надежных программных систем.

Основные особенности Golang:

- 1) Компилируемый язык. Код на Go компилируется в машинный код, что обеспечивает высокую скорость работы программ.
- 2) Кроссплатформенность. Приложения, написанные на Go, могут работать на разных операционных системах и архитектурах процессоров.

- 3) Простота и эффективность. Go был разработан таким образом, чтобы программисты могли быстро и эффективно создавать приложения, даже если они не имеют опыта работы с этим языком.
- 4) Богатая стандартная библиотека. Go поставляется со множеством стандартных библиотек, включая библиотеки для работы с сетью, шифрованием, JSON и XML, обработки изображений, тестирования и т.д.
- 5) Многопоточность. Go имеет встроенную поддержку многопоточности, что облегчает разработку приложений, которые могут эффективно использовать несколько ядер процессора.
- 6) Сборщик мусора. Go имеет встроенный сборщик мусора, что упрощает жизнь разработчикам, позволяя избавиться от необходимости управлять памятью вручную.

Использование Golang для реализации прототипа ядра IoT-платформы имеет ряд преимуществ:

- 1) Высокая скорость выполнения кода, что особенно важно для IoT-систем, работающих с большим объемом данных.
- 2) Простота и эффективность разработки, что позволяет быстро создавать и тестировать прототипы.
- 3) Наличие библиотек для работы с сетью, базами данных и другими технологиями, необходимыми для разработки IoT-приложений.
- 4) Поддержка многопоточности, что позволяет эффективно использовать несколько ядер процессора для обработки данных.

6.3 Quarkus – Java-фреймворк

Часть микросервисов в реализованной IoT-системе написаны на языке Java с использованием фреймворка Quarkus.

Quarkus — относительно новый Java-фреймворк, представленный компанией RedHat. Данный фреймворк базируется на виртуальных машинах

GraalVM и OpenJDK HotSpot и предназначен для Kubernetes, имея встроенную поддержку Docker.

Фреймворк обеспечивает очень низкое потребление памяти и малое время загрузки. Это делает Quarkus идеально подходящим для рабочих нагрузок Java, выполняемых в качестве микросервисов в Kubernetes и OpenShift, а также для рабочих нагрузок Java, выполняемых в виде serverless-функций.

Quarkus поддерживает Hot Reload, что позволяет менять приложение, не перезапуская его, тем самым разработка становится быстрее.

В классах-контроллерах, написанных с помощью данного фреймворка, используются стандартные аннотации из других фреймворков — CDI (в качестве внедрения зависимостей в Quarkus используется собственное решение, основанное на спецификации CDI для Java 2.0) и JAX-RX (Java API для веб-сервисов RESTful), что позволяет легко переходить разработчикам, уже имеющим опыт в работе с данными фреймворками. Используемые аннотации позволяют реализовать полноценный класс-контроллер, написав минимум кода.

Для работы с базами данных, Quarkus использует Hibernate и стандартные JPA аннотации для сущностей. Как и в случае с контроллерами, для создания сущностей необходимо написать минимум кода.

6.4 Базы данных

6.4.1 Postgres

В качестве основного хранилища данных в разрабатываемых сервисах использовалась база данных Postgres

Postgres (или PostgreSQL) — это открытая реляционная база данных с акцентом на расширяемость, соответствие стандартам и производительность.

Она имеет следующие характеристики, которые могут быть полезны для разработки ядра IoT-платформы:

- 1) **Расширяемость.** Postgres позволяет разработчикам создавать пользовательские типы данных, функции и операторы, что позволяет расширять возможности базы данных для работы с конкретными задачами.
- 2) **Поддержка JSON.** Postgres имеет встроенную поддержку JSON, что позволяет эффективно работать с данными IoT устройств, которые обычно отправляют данные в формате JSON.
- 3) **Транзакционность.** Postgres обеспечивает ACID-совместимые транзакции, что обеспечивает целостность данных и избегает ошибок в работе с ними.
- 4) **Производительность.** Postgres может работать с большими объемами данных, благодаря эффективной оптимизации запросов и использованию параллельной обработки.
- 5) **Репликация.** Postgres поддерживает различные методы репликации, что позволяет создавать резервные копии и обеспечивать отказоустойчивость системы.
- 6) **Безопасность.** Postgres предоставляет механизмы авторизации и аутентификации, что обеспечивает безопасность хранения и доступа к данным IoT устройств.

6.4.2 ClickHouse

В качестве аналитического хранилища данных в разрабатываемой системе использовался ClickHouse.

ClickHouse — колоночная СУБД с открытым исходным кодом, разработанная для обработки больших объемов данных в реальном времени с высокой скоростью. Она обеспечивает высокую скорость обработки данных, возможность масштабирования и эффективное использование ресурсов. ClickHouse хранит данные в компактном формате и использует сжатие данных для экономии места на диске.

Характеристики ClickHouse, которые могут быть полезны при разработке ядра IoT-платформы:

- 1) Высокая скорость обработки данных. ClickHouse может обрабатывать миллиарды строк данных в секунду, что является критически важным при обработке большого объема данных, собираемых с IoT-устройств.
- 2) Масштабируемость. ClickHouse легко масштабируется горизонтально, что позволяет обеспечить хранение и обработку большого объема данных.
- 3) Низкие требования к аппаратному обеспечению. ClickHouse требует меньше ресурсов, чем другие колоночные СУБД при обработке больших объемов данных, что позволяет снизить затраты на оборудование.
- 4) Поддержка SQL-запросов. ClickHouse поддерживает язык SQL, что упрощает работу с данными и позволяет создавать более сложные запросы для анализа данных.
- 5) Гибкость. ClickHouse поддерживает различные форматы данных, что позволяет обрабатывать данные, собранные с различных IoT-устройств и в разных форматах.
- 6) Открытый исходный код. ClickHouse имеет открытый исходный код, что позволяет настраивать и дорабатывать платформу в соответствии с потребностями и требованиями проекта.

6.5 Apache Kafka

Apache Kafka — это распределенная платформа, разработанная для обработки потоков данных и реализации шаблона "Producer-Consumer".

Kafka предоставляет возможность обрабатывать и хранить потоковые данные в реальном времени, обеспечивая высокую производительность, масштабируемость и надежность.

Характеристики Kafka, которые могут быть полезны при разработке ядра IoT-платформы:

- 1) Высокая производительность и масштабируемость. Kafka способен обрабатывать миллионы сообщений в секунду и масштабироваться до нескольких терабайт данных в день.
- 2) Распределенная архитектура. Kafka имеет распределенную архитектуру, что позволяет легко масштабировать и обрабатывать большие объемы данных.
- 3) Устойчивость к отказам. Kafka обеспечивает высокую устойчивость к отказам благодаря репликации и резервному копированию данных.
- 4) Гарантированная доставка сообщений. Kafka гарантирует доставку сообщений с использованием механизма подтверждений.
- 5) Низкая задержка. Kafka обеспечивает низкую задержку в обработке сообщений, что делает его подходящим для обработки данных в реальном времени.
- 6) Гибкость. Kafka поддерживает различные протоколы и форматы данных, что позволяет интегрировать его с различными приложениями и системами.
- 7) Богатый функционал. Kafka имеет широкий спектр функций, включая потоковую обработку данных, управление потоками, управление потребителями и многое другое.

6.6 Grafana

Grafana — это открытый инструмент для визуализации и анализа данных. Он позволяет создавать гибкие и интерактивные панели для мониторинга и анализа различных метрик и логов из разных источников данных, таких как базы данных, системы мониторинга, приложения и другие.

Grafana поддерживает множество источников данных, включая Graphite, Elasticsearch, InfluxDB, Prometheus, MySQL, PostgreSQL и многие другие. Он предоставляет богатый набор возможностей для визуализации данных, таких как графики, таблицы, диаграммы, гистограммы и т.д. Пользователи могут

настраивать панели в соответствии со своими потребностями, используя различные плагины и настраиваемые панели.

Grafana также обладает мощным инструментом алертинга, который позволяет настраивать оповещения на основе определенных условий и метрик, что позволяет быстро реагировать на проблемы и уведомлять об этом соответствующих сотрудников.

Grafana может быть использован для мониторинга и анализа различных систем, включая Интернет вещей, и может быть интегрирован с другими системами мониторинга и анализа данных, что делает его мощным инструментом для создания дашбордов и отчетов для управления IoT-системами.

6.7 Python

Python в разрабатываемой системе используется как средство пользовательского расширения функционала системы.

Python — высокоуровневый интерпретируемый язык программирования с динамической типизацией и автоматическим управлением памятью. Он был создан в конце 1980-х годов Гвидо ван Россумом и получил своё название в честь комедийного телешоу Monty Python's Flying Circus.

Python имеет простой и понятный синтаксис, что делает его очень легко читаемым и понятным для новичков. Он также обладает мощным набором стандартных библиотек и средств разработки, которые упрощают разработку программ на Python. Python поддерживает многопоточность и асинхронность, что позволяет создавать масштабируемые приложения.

Python используется для различных задач, включая научные исследования, анализ данных, создание веб-серверов и API, автоматизацию задач, разработку игр, создание мобильных и настольных приложений и многое другое. Он также широко используется в машинном обучении, искусственном интеллекте и анализе данных благодаря мощным библиотекам, таким как NumPy, Pandas, Scikit-Learn и TensorFlow.

Python имеет активное сообщество разработчиков и множество сторонних библиотек и фреймворков, что делает его одним из наиболее популярных языков программирования в мире.

6.8 Keycloak

Одной из главных проблем в концепции Интернета вещей является проблема безопасности.

Keycloak — это открытое программное обеспечение для управления аутентификацией и авторизацией в приложениях. Он предоставляет удобный интерфейс для управления пользователями, группами, ролями, правами доступа и другими функциями безопасности. Keycloak поддерживает множество протоколов аутентификации и авторизации, включая OpenID Connect, OAuth 2.0 и SAML.

Для использования в разработке ядра IoT-платформы, Keycloak может использоваться для управления доступом к различным ресурсам в системе, например, к API, консоли управления и другим сервисам. Он позволяет легко создавать и управлять пользователями, ролями и группами, устанавливать права доступа на основе ролей и предоставлять аутентификацию через различные социальные сети и другие системы.

Keycloak также обеспечивает поддержку многих функций безопасности, включая двухфакторную аутентификацию и управление сессиями пользователей.

6.9 Kubernetes

Kubernetes — это современная платформа для управления контейнерами, которая может быть использована в качестве инструмента для развертывания и управления приложениями Интернета вещей. Он обеспечивает автоматизацию процессов развертывания, масштабирования и управления контейнерами, что позволяет значительно упростить процесс разработки и развертывания приложений IoT.

Одной из ключевых особенностей Kubernetes является его способность к масштабированию приложений, что может быть особенно полезно в случае с IoT-приложениями, где количество устройств, с которыми нужно взаимодействовать, может значительно возрасть. Kubernetes также обеспечивает высокую доступность приложений благодаря механизмам отказоустойчивости и восстановления после сбоев.

Характеристики Kubernetes, которые могут быть полезны при разработке ядра IoT-платформы:

- 1) Масштабируемость. Kubernetes предоставляет мощные средства для управления масштабированием приложений, что является важным аспектом разработки ядра IoT-платформы. Кластер Kubernetes позволяет горизонтально масштабировать приложения, что особенно важно в случае Интернета вещей, где множество устройств может генерировать большое количество данных и требовать высокой производительности и отказоустойчивости.
- 2) Гибкость. Kubernetes предоставляет гибкую архитектуру, позволяющую разработчикам настраивать и настраивать приложения в соответствии с требованиями IoT-платформы. Это включает в себя возможность использования контейнеров, которые являются легковесными и переносимыми единицами развертывания, что позволяет упростить процесс разработки и развертывания IoT-приложений.
- 3) Отказоустойчивость. Kubernetes предоставляет механизмы автоматического восстановления и самоисцеления приложений, что способствует повышению отказоустойчивости разрабатываемого ядра IoT-платформы. Кластер Kubernetes автоматически перезапускает неисправные контейнеры, переносит их на другие узлы и осуществляет мониторинг состояния приложений.
- 4) Управляемость. Kubernetes предоставляет богатый набор инструментов и функций для управления приложениями, включая

мониторинг, логирование, масштабирование, обновление и откат, что облегчает процесс разработки и управления ядром IoT-платформы.

- 5) Экосистема. Kubernetes имеет широкую и активную экосистему, включающую множество плагинов, расширений и интеграций, что позволяет расширять функциональность разрабатываемой IoT-платформы и интегрироваться с другими технологиями и сервисами.
- 6) Открытость и сообщество. Kubernetes является проектом с открытым исходным кодом, поддерживаемым международным сообществом разработчиков.

7 Разработанная IoT-платформа

7.1 api-gateway

api-gateway — это сервис, который принимает запросы от клиентов и проводит их маршрутизацию к соответствующим микросервисам. Он является единой точкой входа для всех запросов, входящих в систему, и выполняет ряд функций, таких как аутентификация, авторизация, маршрутизация и трансформация запросов и ответов.

Использование единой точки входа в систему позволяет обеспечить легкую масштабируемость и гибкость при разработке и поддержке системы, поскольку он позволяет скрыть сложность микросервисной архитектуры от конечных пользователей и разработчиков. Кроме того, api-gateway обеспечивает более эффективное использование сетевых ресурсов, позволяя объединять запросы и ограничивать доступ к микросервисам.

В целом, использование api-gateway помогает улучшить производительность, надежность и безопасность системы, а также облегчает ее разработку и поддержку.

Компонентная структура расположения микросервисов и взаимодействия между ними показана на рисунке 9.

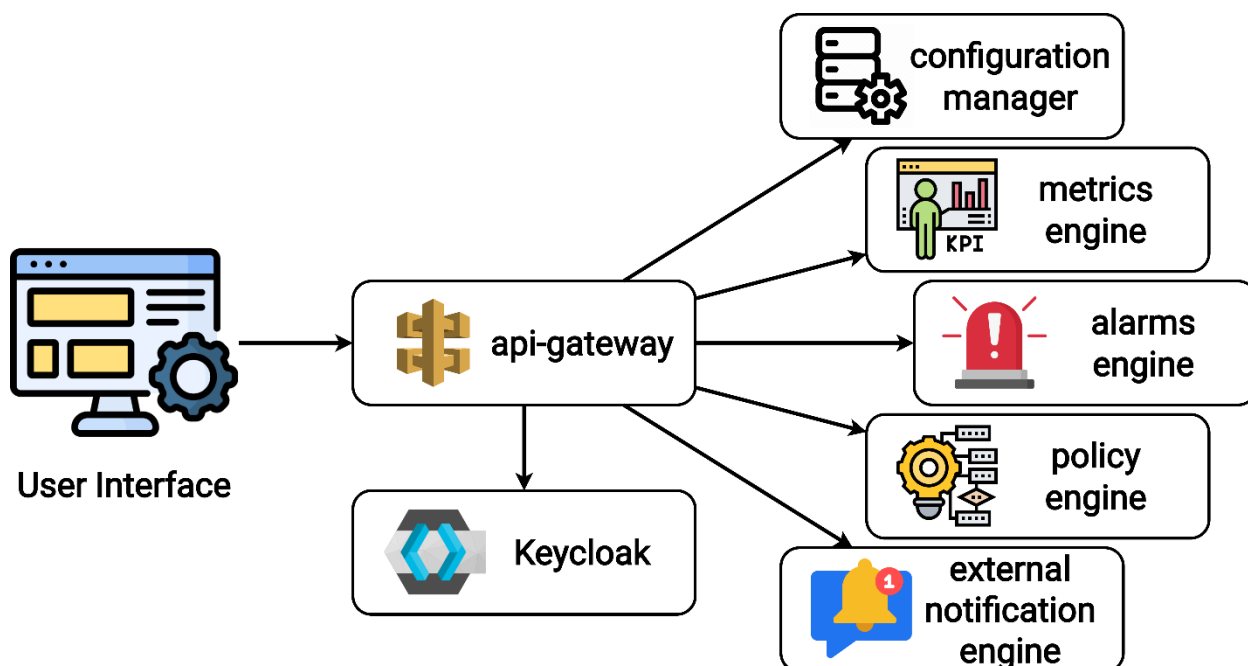


Рисунок 9 – Компонентная архитектура сервиса Api Gateway

Данный компонент написан на Java с использованием Spring Boot и библиотеки Zuul, который непосредственно осуществляет маршрутизацию и авторизацию пользователя.

В качестве сервера авторизации был выбран Keycloak — open-source сервер аутентификации и управления учетными записями от JBoss, построенный на базе спецификаций OAuth 2.0, Open ID Connect, JSON Web Token (JWT).

7.2 configuration-storage

Для начала стоит ознакомиться с сущностями, которыми владеет разрабатываемая система. Сущности и связи между этими сущностями представлены на рисунке 10.

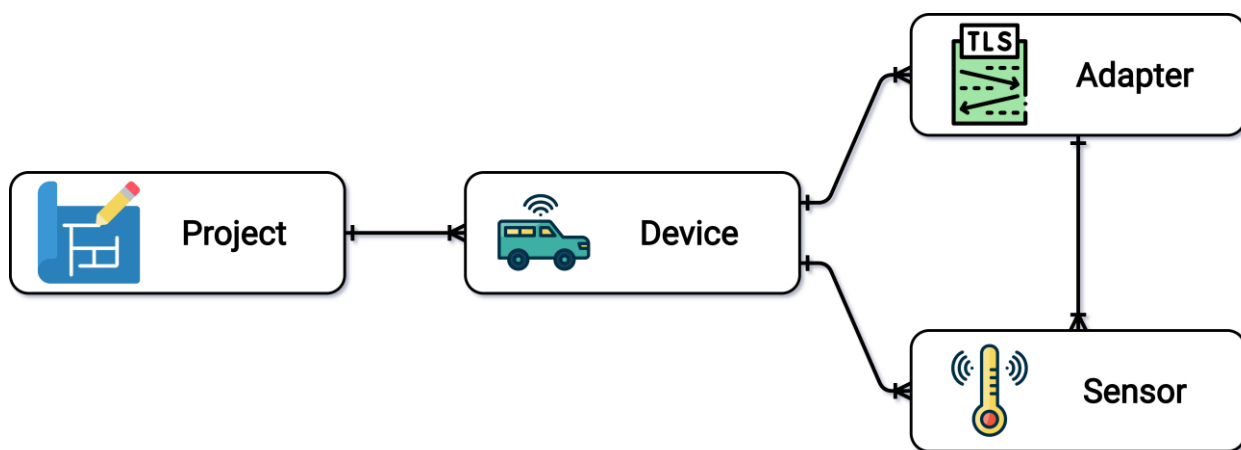


Рисунок 10 – Ключевые сущности, оперируемые системой

Система оперирует различными сущностями, основной из которых является проект. Проект (Project) — это набор объектов, объединенными связями, который привязан к определенному пользователю системы (организации, подразделение организации). Проект является логическим контейнером всех сущностей, которые должны функционировать как единое целое в составе данного проекта. Несколько проектов в конкретной организации может соединяться другими проектами-каталогами.

Для системы Интернета вещей интересно понятие объекта управления, то есть девайс (Device) — ключ компонент системы, с которого пользователь хочет собирать данные.

На девайсе также может быть несколько устройств, которые осуществляют сбор и отправку данных. Эта информация поступает с сенсоров (Sensor) через каналы связи, которые с точки зрения системы называются адаптерами (Adapter).

Например, есть система каршеринга, которая владеет автопарком. Компания установила на машины телеметрическое оборудование при помощи какой-то сторонней организации. Также написала программное обеспечение для конечного клиента, например, мобильное приложение, но им не хватает связующего элемента, который свяжет систему работы с клиентом и с системой автопарка (например, чтобы транспорт не угнали или не повредили).

С точки зрения сбора данных телеметрии оборудования, компания могла бы купить услуги разрабатываемой системы для конкретного проекта в определенном городе. И, если она владеет автопарком в нескольких городах, то может уже создать проекты-каталоги для каждого города отдельно, либо разделить по следующему принципу, например, по качеству автомобилей (бизнес-класс и эконом).

Общая структура сервиса Configuration Storage представлена на рисунке 11.

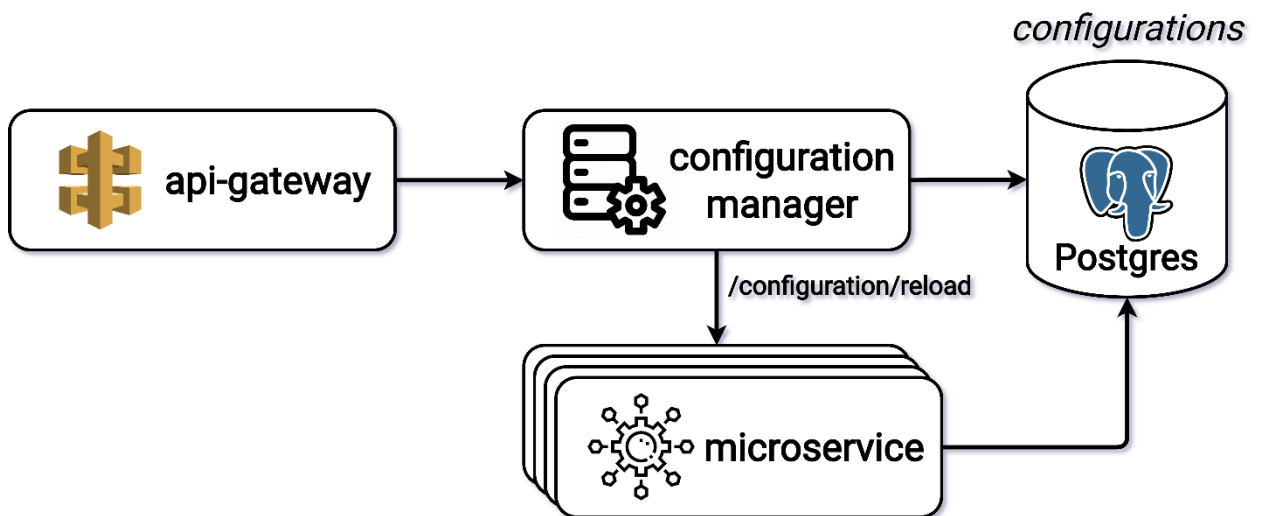


Рисунок 11 – Компонентная архитектура сервиса Configuration Storage

Данный компонент написан на языке Java с использованием фреймворка Quarkus и привлечением базы данных PostgreSQL.

Основная цель данного сервиса предоставить интерфейс для сохранения конфигураций проекта, а также для поддержки их редактирования и удаления. Кроме того, сервис осуществляет передачу этих конфигураций дальше по архитектуре системы.

7.3 metrics-engine

7.3.1 Описание сервиса

Сервис написан на языке Go с привлечением баз данных Postgres, ClickHouse и очереди сообщений Kafka.

Основная цель данного сервиса заключается в следующем:

- обработка сообщений (метрик), приходящих с адаптеров и со сторонних систем (через HTTP);
- вычисление агрегационных метрик (KPI) на основе заданных пользователем конфигураций;
- сохранение метрик в аналитической базе данных ClickHouse;
- отправка метрик в очередь сообщений в специальный Kafka топик для их последующей обработки.

7.3.2 Метрики (Metric)

Любая приходящая метрика должна иметь набор определённых полей в соответствии с контрактом IoT-платформы.

```
{
  "__name__": "temperature",
  "__value__": 70,
  "__timestamp__": "2013-04-14T20:00:00Z",
  "__sensor_id__": "d2c87c41-9bf7-4ab0-8399-088d56d1e73f",
  "__sensor_name__": "Temperature Sensor",
  "property_1": "value_1",
  "property_2": "value_2"
}
```

Служебные поля:

- **__name__** – название метрики
- **__value__** – значение метрики
- **__timestamp__** – время, когда эта метрика была собрана. В случае, если в пришедшей метрики отсутствует данное поле, метрики присвоится временная отметка момента, когда она пришла в сервис metrics-engine
- **__sensor_id__** – UUID устройства (сенсора), с которого эта метрика была собрана
- **__sensor_name__** – название устройства, с которого эта метрика была собрана

Пользователь имеет возможность указывать другие поля, которые он пожелает. Эти поля будут сохранены как отдельные свойства (properties) для дальнейшей фильтрации.

7.3.3 Timeseries и samples

Пришедшая метрика разбивается на две отдельные сущности: sample и timeseries, которые связаны между собой специальным идентификатором fingerprint. Подобное разбиение позволяет хранить описание метрики и её значения в различных таблицах, что уменьшает размер хранимой информации и увеличивает скорость получения результатов из аналитической базы данных.

Timeseries (таймсерия) – сущность, которая описывает уникальную метрику. Уникальность метрики определяется её служебными полями и дополнительными свойствами. Идентификатор Fingerprint рассчитывается как хэш от набора служебных полей и дополнительных свойств.

SQL запрос для создания таблицы timeseries выглядит следующим образом:

```
CREATE TABLE IF NOT EXISTS timeseries ON CLUSTER '{cluster}'
(
    `metric_key`    FixedString(1) DEFAULT substringUTF8(metric,
1, 1),
    `metric`        String,
    `fingerprint`   FixedString(32),
    `project_id`    UUID,
    `object_id`     UUID,
    `sensor_id`     UUID,
    `properties`    JSON
)
ENGINE =
ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/time_se
ries_{uuid}', '{replica}')
PARTITION BY (metric_key)
ORDER BY (metric, sensor_id, fingerprint);
```

Sample (сэмпл) – сущность, которая содержит в себе значение в определённый момент времени для конкретной таймсерии.

SQL запрос для создания таблицы `samples` выглядит следующим образом:

```
CREATE TABLE IF NOT EXISTS samples ON CLUSTER `{cluster}`
(
    `date`                Date,
    `timestamp`           DateTime64(3),
    `fingerprint`         FixedString(32),
    `value`               Float64,
    `insert_timestamp`    DateTime64(3) DEFAULT now64(3)
)
ENGINE =
ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/samples
_{uuid}', '{replica}', insert_timestamp)
PARTITION BY toYYYYMM(date)
ORDER BY (date, fingerprint, timestamp);
```

7.3.4 Конфигурация сервиса

Пример конфигурации для создания агрегаций на основе приходящих метрик приведена ниже.

```
[
  {
    "aggregation_name": "avg_temperature_living_room_10m",
    "formula": "avg(temperature_living_room, 10m)",
    "selector": {}
  },
  {
    "aggregation_name": "min_temperature_living_room_10m",
    "formula": "min(temperature_living_room, 10m)",
    "selector": {}
  },
  {
    "aggregation_name": "max_temperature_living_room_10m",
    "formula": "max(temperature_living_room, 10m)",
    "selector": {}
  }
]
```

Пользователь при создании конфигурации для агрегаций, должен указать следующие поля:

- *aggregation_name* – название агрегации
- *formula* – формула, по которой рассчитывается агрегация

- *selector* – набор полей из метрики для выбора соответствующей метрики, в случае, если в системе наблюдаются метрики с одинаковым названием

7.3.5 Компонентная структура сервиса

Компонентная структура данного сервиса представлена на рисунке 12.

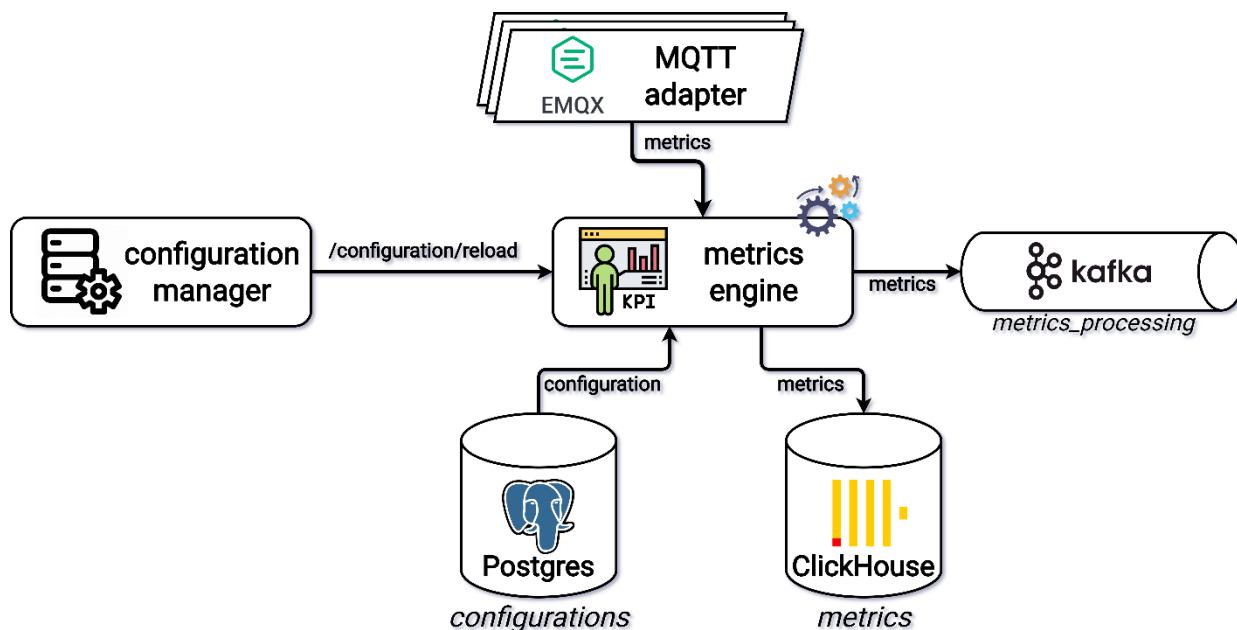


Рисунок 12 – Компонентная архитектура сервиса Metrics Engine

Конфигурация сервиса хранится в соответствующей таблице конфигурационной базы данных. При запуске, сервис загружает конфигурацию себе в память. В случае, если пользователь добавил, изменил или удалил конфигурацию, *configuration-manager* посылает запрос *metrics-engine* для перезагрузки конфигурации.

Метрики, отправляемые пользователем на конкретные адаптеры (MQTT или STOMP), через WebHook пересылаются на специальный эндпоинт сервиса *metrics-engine*.

Полученные метрики сразу же отправляются с специальный Kafka-топик (*metrics_preprocessing*) для первоначальной обработки самим же сервисом *metrics-engine*. Это сделано для того, чтобы не тратить время на процессинг

непосредственно во время обработки запроса, а делать это в отдельном потоке в фоне.

Параллельно обработке HTTP запросов, метрики в фоне считываются с Kafka-топика `metrics_preprocessing`.

Считанная метрика сохраняется в соответствующую таблицу в аналитической базе данных в ClickHouse и подвергается дополнительному процессингу.

В случае, если в конфигурации есть формула для агрегации, в которой присутствует пришедшая метрика, создается новая метрика со названием и значением в соответствии с указанной конфигурацией. Эта метрика также отправляется в Kafka-топика `metrics_preprocessing` и процесс её обработки аналогичен описанному выше.

7.4 alarms-engine

7.4.1 Описание сервиса

Сервис написан на языке Go с привлечением баз данных Postgres, ClickHouse и очереди сообщений Kafka.

Основная цель данного сервиса заключается в следующем:

- обработка метрик, находящихся в специальном Kafka топике;
- создание оповещений (Alarm) на основе заданных пользователем конфигураций;
- сохранение оповещений в аналитической базе данных ClickHouse;
- отправка оповещений в очередь сообщений в специальный Kafka топик для их последующей обработки.

7.4.2 Оповещение (Alarm)

В качестве единой сущности, используемой для выполнения бизнес-логики в разработанной IoT-системе, используется понятие оповещения (аларм, аварийный сигнал), которое регулируется в соответствии с рекомендацией X.733 System Management.

Оповещение – ключевая сущность IoT-платформы, которая хранит в себе информацию о конкретной возникшей ситуации. Оповещения возникают в системе, на основе конфигурации, заданной пользователем.

SQL запрос для создания таблицы *alarms* выглядит следующим образом:

```
CREATE TABLE IF NOT EXISTS alarms ON CLUSTER '{cluster}'
(
    `date`                Date,
    `timestamp`           DateTime,
    `alarm_id`            UUID,
    `alarm_name`          String,
    `sensor_id`           UUID,
    `sensor_name`         String,
    `severity`            String,
    `properties`          JSON
)
ENGINE =
ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/alarms_{uuid}', '{replica}')
PARTITION BY toYYYYMM(date)
ORDER BY (date, timestamp);
```

7.4.3 Конфигурация сервиса

Пример конфигурации для создания алармов на основе приходящих метрик приведена ниже.

```
[
  {
    "alarm_name": "temperature_low",
    "condition": "temperature_living_room < 12",
    "severity": "Critical",
    "selector": {}
  },
  {
    "alarm_name": "temperature_low",
    "condition": "temperature_living_room >= 12 and
temperature_living_room < 16",
    "severity": "Major",
    "selector": {}
  },
  {
    "alarm_name": "temperature_low",
    "condition": "temperature_living_room >= 16 and
temperature_living_room < 18",
    "severity": "Minor",
    "selector": {}
  }
]
```

Пользователь при создании конфигурации для возникновения алармов, должен указать следующие поля:

- *alarm_name* – название аларма
- *condition* – формула (или условие), при котором должен возникнуть аларм
- *severity* – серьезность аларма. В соответствии с рекомендацией X.733 System Management [27], данное поле должно иметь одно из следующих значений:
 - **Cleared.** Уровень серьезности Cleared указывает на удаление одного или нескольких ранее зарегистрированных алармов
 - **Indeterminate.** Уровень серьезности Indeterminate указывает на то, что уровень серьезности не может быть определен
 - **Critical.** Уровень серьезности Critical указывает на то, что возникло состояние, влияющее на обслуживание, и требуется немедленное действие по исправлению ситуации
 - **Major.** Уровень серьезности Major указывает на то, что у возникло состояние, влияющее на обслуживание, и требуется срочное корректирующее действие
 - **Minor.** Уровень серьезности Minor указывает на существование состояния неисправности, не влияющего на обслуживание, и что необходимо предпринять корректирующие действия для предотвращения более серьезной (например, влияющей на обслуживание) неисправности
 - **Warning.** Уровень серьезности Warning указывает на обнаружение потенциальной или надвигающейся неисправности, влияющей на обслуживание. Необходимо принять меры для дальнейшей диагностики (если это необходимо) и устранения проблемы, чтобы предотвратить ее перерастание в более серьезную проблему

- *selector* – набор полей из метрики для выбора соответствующей метрики, в случае, если в системе наблюдаются метрики с одинаковым названием

7.4.4 Компонентная архитектура сервиса и его работа

Компонентная структура данного сервиса представлена на рисунке 13.

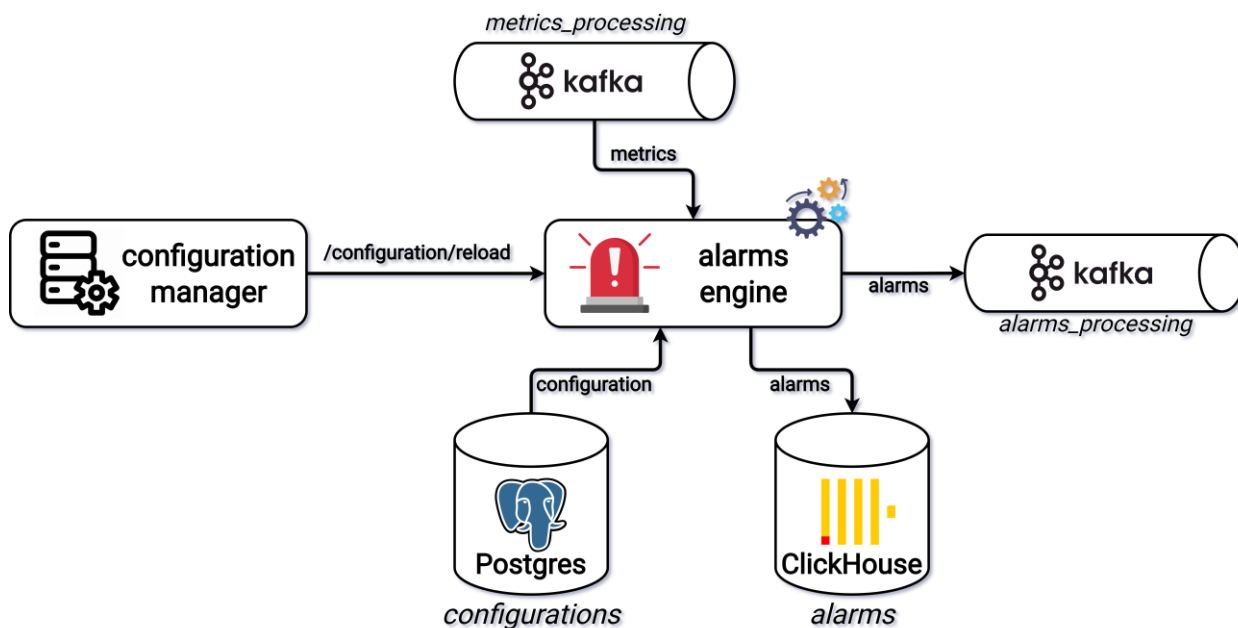


Рисунок 13 – Компонентная архитектура сервиса Alarms Engine

Конфигурация сервиса хранится в соответствующей таблице конфигурационной базы данных. При запуске, сервис загружает конфигурацию себе в память. В случае, если пользователь добавил, изменил или удалил конфигурацию, **configuration-manager** посылает запрос **alarms-engine** для перезагрузки конфигурации.

Сообщения-метрики, записанные в специальный топик сервисом **metrics-engine**, считываются **alarms-engine**.

Полученные метрики проверяются в соответствии с конфигурациями. В случае, если пришедшая метрика удовлетворяет условию возникновения аларма, он сохраняется в соответствующую таблицу в аналитической базе данных в **ClickHouse**, и затем отправляется в специальный **Kafka**-топик для дальнейшей обработки другими сервисами.

7.5 policy-engine

7.5.1 Описание сервиса

Сервис написан на языке Go с привлечением базы данных Postgres и очереди сообщений Kafka

Основная цель данного сервиса заключается в следующем:

- считывание алармов, находящихся в специальном Kafka топике;
- применение различных политик для обработки алармов на основе заданных пользователем конфигураций.

7.5.2 Политика (Policy), условие (Condition) и действие (Action)

В качестве главной сущности, которыми оперирует policy-engine является политика (Policy) или правило.

Политика (Policy) — объект системы, состоящий из условия (Condition) и набора действий (Actions), которые должны быть выполнены при выполнении условия.

7.5.3 Конфигурация сервиса

Пример конфигурации политики приведен ниже.

```
[
  {
    "condition": {
      "condition_name": "alarm_raised",
      "selector": {
        "alarm_name": "temperature_low",
        "severity": "Critical"
      }
    },
    "actions": [
      {
        "action_name": "console_log",
        "properties": {
          "message_template":
            "[{{.Alarm.Timestamp}}] [{{.Alarm.Severity}}] received
            alarm `{{.Alarm.AlarmName}}`, please check {{
            .Alarm.SensorName }}",
```

```

        "log_level": "INFO"
    },
    {
        "action_name": "external_notification",
        "properties": {
            "notification_type": "telegram",
            "message_template":
"[[{{.Alarm.Timestamp}}] [{{.Alarm.Severity}}] received
alarm `{{ .Alarm.AlarmName }}`, please check {{
.Alarm.SensorName }}",
            "recipients": [
                "811071503"
            ],
            "additional_properties": {
                "host":
"https://api.telegram.org",
                "token": "<token>"
            }
        }
    }
]
}
1

```

7.5.4 Компонентная архитектура сервиса и его работа

Компонентная структура данного сервиса представлена на рисунке 14.

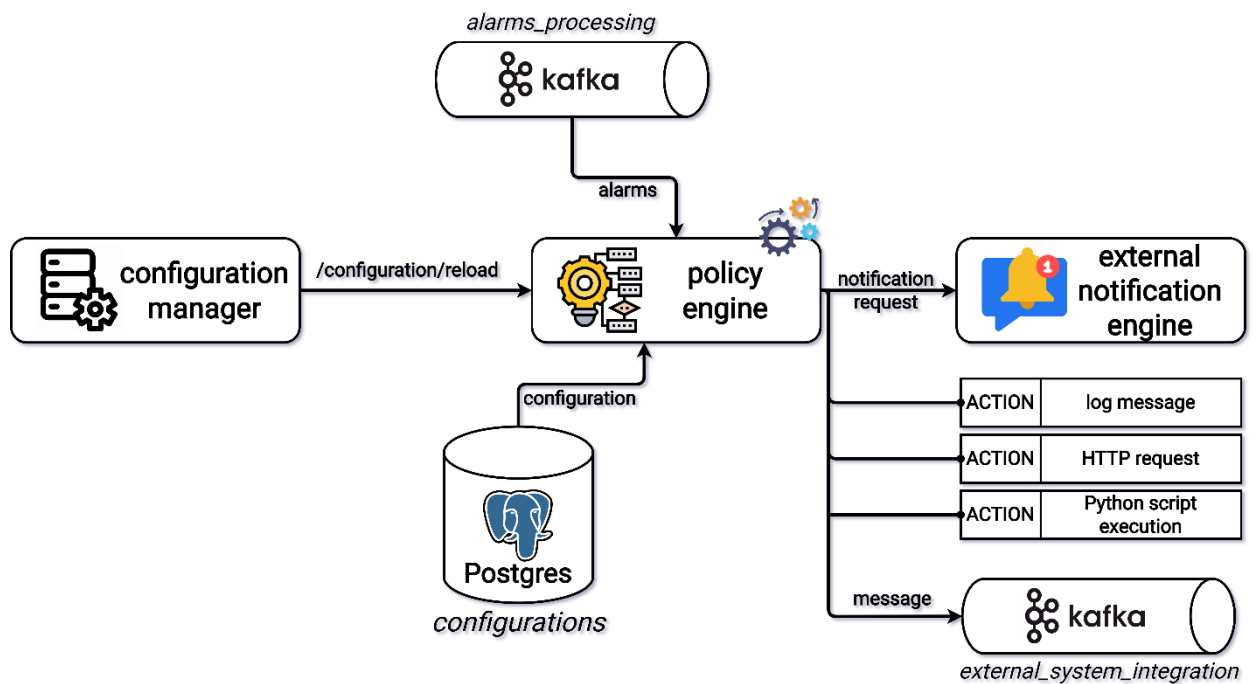


Рисунок 14 – Компонентная архитектура сервиса Policy Engine

7.6 external-notification-engine

7.6.1 Описание сервиса

Сервис написан на языке Go.

Основная цель данного сервиса заключается в следующем:

- отправка оповещений (нотификаций) на устройства пользователей, используя различные каналы связи.

7.6.2 Компонентная архитектура сервиса и его работа

Компонентная архитектура сервиса представлена на рисунке 15.

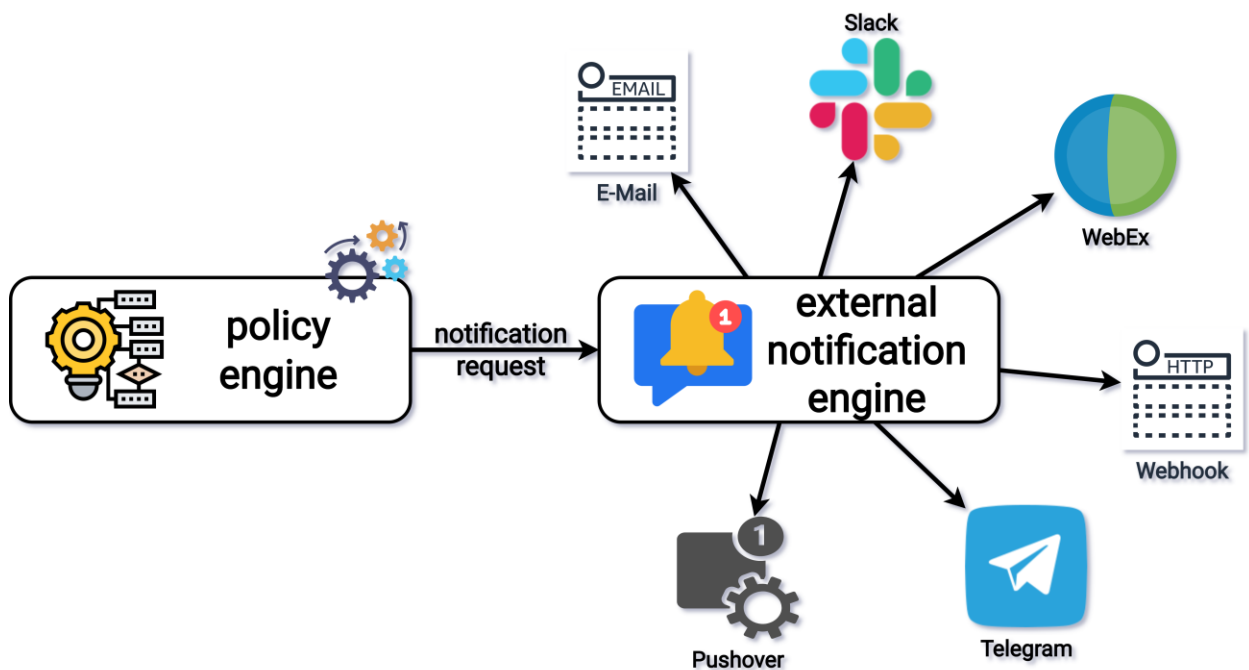


Рисунок 15 – Компонентная архитектура сервиса External Notification Engine

Структура микросервиса предоставляет разработчику набор интерфейсов, позволяющих их реализовывать и без проблем добавлять новые виды протоколов отправки.

Для отправки нотификации, на сервис нужно отправить запрос на эндпоинт следующего вида:

<http://external-notification-engine:8080/api/v1/<service>/send>,

где <service> — тип сервиса, через который нужно отправить нотификацию.

Тело запроса должно иметь следующий вид.

```
{
  "message_text": "[2023-04-19 09:00:07.0124905 +0300 MSK]
[Critical] received alarm `temperature_low`, please check
`Temperature Sensor (Living Room 536)`",
  "recipients": [
    "811071503"
  ],
  "additional_properties": {
    "host": "https://api.telegram.org",
    "token": "<token>"
  }
}
```


На данный момент поддерживаются следующие виды сервисов:

- Email – отправка нотификации через электронную почту
- Slack – отправка нотификации в мессенджер Slack

7.7 Пользовательский интерфейс

Пользовательский интерфейс, предоставляющий пользователю возможность создания сущностей и конфигураций, которые используются в системе, реализован в виде одностраничного приложения с помощью JavaScript-фреймворка ReactJS и компонентной библиотеки Ant Design. Данная библиотека предоставляет большое разнообразие различных компонентов, которые удобно встраивать в страницу и легко дополнять и расширять.

С точки зрения пользовательского интерфейса и требований, которые были для него определены, было решено создать макеты для UI. На рисунке 16 показан макет страницы с личной информацией о пользователе.

Octopus

Личная информация

Основная информация

Фотография

Имя

Имя имя

>

Пароль

>

Имя пользователя

NameName

>

Контактная информация

Почта

name@mail.ru

>

Телефон

+7123123

>

Рисунок 16 – Макет страницы с личной информацией о пользователе

Для отображения пользовательской информации была реализована страница пользователя, которая отображает личную информацию пользователя. Для просмотра проектов, которые есть у пользователя была создана страница для просмотра проектов, созданных пользователем (рисунок 17).

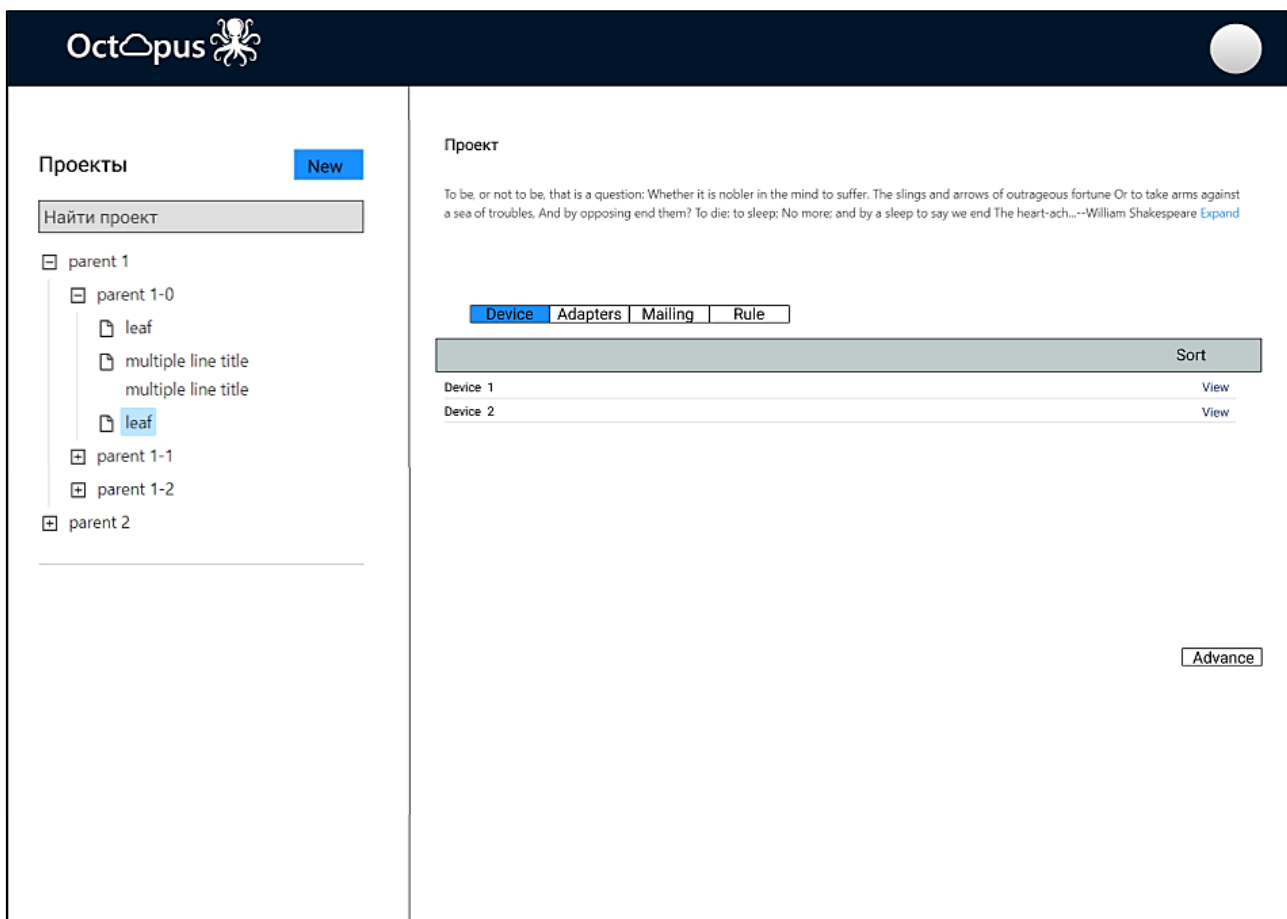


Рисунок 17 – Макет страницы с информацией о созданных проектах пользователя

Создание проекта осуществляется при помощи Мастера создания проекта (рисунки 18 – 24), который поэтапно позволяет пользователю полностью осуществить создание проекта и наполнение его связанными с ним элементами, такими как Девайсы, Адаптеры, Сенсоры, Конфигурации рассылки и Политики.

Рисунок 18 – Мастер создания проекта. Создание проекта

На первом шаге (рисунок 18) пользователь должен заполнить информацию о создаваемом проекте: название проекта, описание проекта и его версию. При нажатии на кнопку Next, производится запрос на сервер для создания объекта проекта с заданными полями.

Рисунок 19 – Мастер создания проекта. Создание девайсов

На втором шаге (рисунок 19) пользователю предлагается заполнить информацию о Девайсах, которые принадлежат проекту: название девайса, описание девайса, состояние девайса (указывается на то, будет ли активным

данный девайс сразу после создания) и дополнительные свойства девайса, которые могут использоваться при обработке политик. Кроме того, пользователь на данном этапе может создать сразу несколько девайсов.

The screenshot displays the 'Adapter 0' configuration interface. At the top, there's a title 'Adapter 0' and a 'Remove adapter 0' button. Below this are several form fields: '* Device:' with a dropdown menu showing 'bmw'; '* Adapter name:' with a text input containing 'mqttbmwadapter'; '* Message Protocol:' with a dropdown menu showing 'MQTT'; '* Username:' with a text input containing 'usertest'; and '* Password:' with a masked input field. Under the 'PROPERTIES' section, there is a table with one row: 'country' (type 'string', value 'Russia'). Below the table is an '+ Add property' button. At the bottom of the form is an '+ Add adapter' button. The footer contains 'Previous' and 'Next' navigation buttons, with 'Next' being highlighted in blue.

Рисунок 20 – Мастер создания проекта. Создание адаптеров

На следующем шаге (рисунок 20), перед созданием сенсоров, пользователю предлагается создать адаптеры — объекты системы, через которые осуществляется передача данных по различным протоколам передачи данных. Для создания адаптера пользователю нужно выбрать один из девайсов, указать название этого адаптера и выбрать протокол передачи данных (на данный момент реализована поддержка двух протоколов — MQTT и STOMP).

The screenshot shows the 'Sensor 0' configuration screen. At the top, there's a 'Remove Sensor 0' button. Below it, the 'Device' is set to 'bmw', the 'Adapter' to 'mqttbmwadapter (mqtt)', and the 'Sensor name' to 'alcosensor'. The 'State' is toggled to 'Enabled'. The 'SENSOR TOPIC' section has a 'Topic' dropdown set to 'Create new topic'. Below this, the 'TOPIC PREVIEW' shows 'group1/metric/alco', with 'group1' and 'Metric' as dropdowns and 'alco' as a text input. The 'SENSOR PROPERTIES' section has two buttons: '+ Add property' and '+ Add Sensor'. At the bottom, there are 'Previous' and 'Next' buttons.

Рисунок 21 – Мастер создания проекта. Создание сенсоров

На следующем шаге (рисунок 21) предлагается создать сенсоры и топики, в которые эти сенсоры должны отправлять информацию.

Каждый сенсор привязан к определённому адаптеру, поэтому для начала нужно выбрать Адаптер, затем указать название сенсора и создать для этого сенсора топик. При создании топика нужно указать группу, тип и значение топика. Группа позволяет агрегировать топики по смысловой нагрузке и в дальнейшем может использоваться для обобщенного применения политик. Тип топика *Metric* — передача метрических данных, *Event* — передача пары ключ-значение для вызова действий. Значение — это собственно конечное название топика.

SENDER CONFIGURATION

Sender Configuration: Create new config

* Protocol: Telegram

* host: https://api.telegram.org/

* username: equilibriumgroup_bot

* token: 1685152028:AAEOGQdXN42_rdpS1hZtuPZlIbz5p8PXAxSo

CONSUMERS

Consumer 0

Remove Consumer 0

PROPERTIES

Name	Type	Value
+ Add property		
+ Add consumer		
+ Add mailing configuration		

Рисунок 22 – Мастер создания проекта. Создание конфигураций рассылки

На этапе создания конфигураций рассылки (рисунок 22) пользователь указывает способ передачи сообщения, параметры для отправки сообщений и конечных адресатов.

При добавлении адресата нужно в зависимости от способа отправки добавить свойства, характеризующие адресата. В данном случае нужно добавить свойство с именем `chat_id`, типом `integer` и в качестве значения указать `id` чата, в который бот должен отправить сообщение

Remove rule 0

* Rule Name: alcorule2

CONDITIONS

Condition 0

Remove condition 0

* Condition Name: alcocondition2

* Device: bmw

* Adapter: mqttbmwadapter (mqtt)

* Group: group1

* Topic: group1/metric/alco

* Template: TemplateCondition

* My Value:

Рисунок 23 – Мастер создания проектов. Создание политик для обработки сообщений. Создание условий

+ Add condition

ACTIONS

Action 0

Remove action 0

* Action Name: aclonotification

* Template: TemplateAction

* Message: Водитель пьян!

* Mail configuration: (f7694a60-c704-4d6a-9491-c6e97c35e36d)

+ Add action

+ Add rule

Previous Success

Рисунок 24 – Мастер создания проектов. Создание политик для обработки сообщений. Создание действий

При создании политик пользователь должен указать действие (Action, рисунок 24), которое должно выполняться при обработке политики и условие (Condition, рисунок 23), при котором выполняется действие.

При создании условия и действия нужно выбрать шаблон условия или действия, по которому строятся нужные поля на UI.

8 Применение разработанной платформы в различных областях и отраслях

Проведем проверку работоспособности разработанного ядра на примере создания приложения Интернета вещей для отельной отрасли. Для этого сначала нужно определить конфигурацию проекта.

Предположим, требуется наблюдать за температурой в номерах, и, в случае высокой, либо низкой температуры, отправлять администратору сообщение в Telegram.

В качестве объекта наблюдения будем рассматривать комнаты с датчиками, которые отслеживают и собирают значения температуры, а также могут отправлять собранные данные в систему по протоколам MQTT и STOMP.

8.1 Создание конфигурации проекта

8.1.1 Описание проекта

Для поддержки заданного сценария достаточно создать единственный проект. В случае, если у заказчика сеть отелей, то он может создать несколько проектов, где каждый проект соответствует определенному отелю

В качестве название введем “Demo Hotel”, а для описания – “Project Wizard Demonstration”.

Заполненные поля на первом шаге создания конфигурации проекта приведены на рисунке 25.

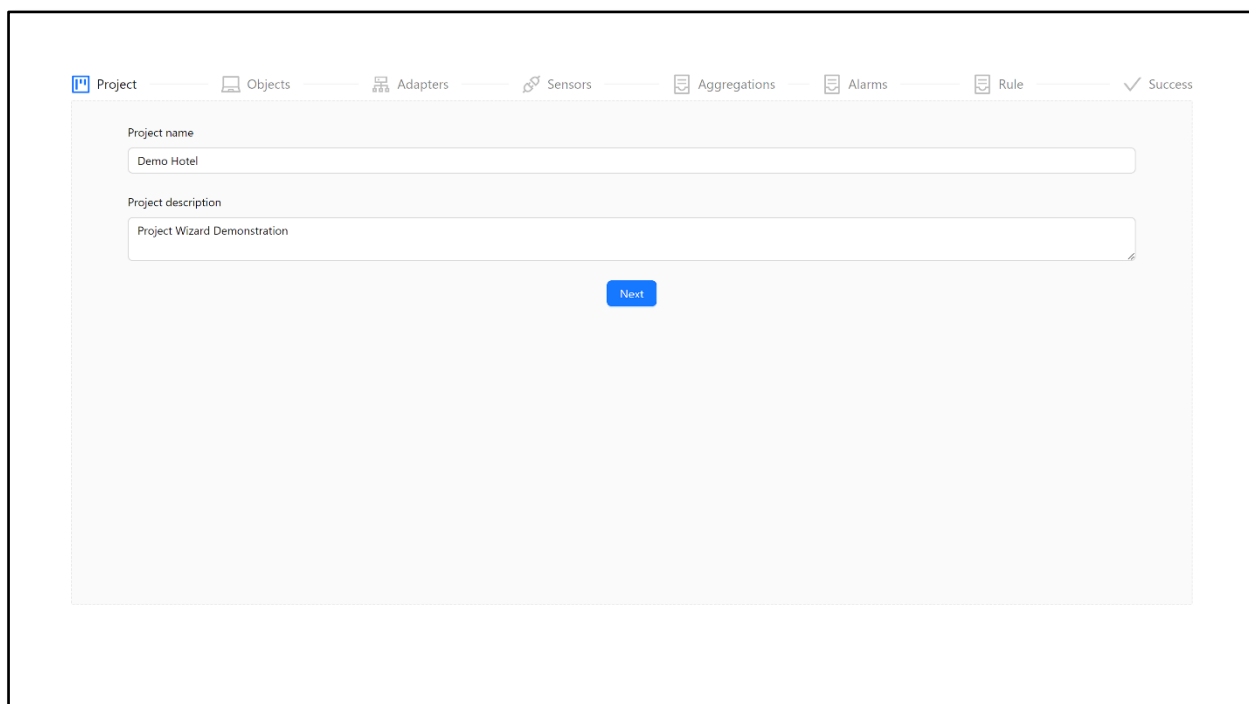


Рисунок 25 – Мастер создания проекта. Конфигурация проекта

8.1.2 Описание объектов наблюдения

В качестве объекта наблюдения в данном сценарии выступает конкретный номер в отеле.

Создадим для примера два разных номера (Room 384 и Room 472), в котором располагаются датчики для измерения температуры.

В качестве описания (description) объекта можно указывать любое значение, которое в дальнейшем поможет пользователю различать объекты на странице проекта. Введем для примера этаж, на котором располагается создаваемые номера.

В качестве дополнительных свойств для разграничения создаваемых объектов можно указать различные атрибуты – пары ключ-значения. В дальнейшем по этим атрибутам будет возможность в том числе фильтровать метрики, которые относятся к данному объекту. Но сейчас этот функционал не используется, а атрибуты только сохраняются в конфигурационную базу данных.

Заполненные поля на втором шаге создания конфигурации проекта приведены на рисунке 26.

The screenshot displays the 'Objects' configuration step of a project wizard. The progress bar at the top indicates the current step. Two objects are listed:

- Object 0:**
 - Object Name: Room 384
 - Object Description: Floor 3
 - OBJECT PROPERTIES:
 - floor (integer): 3
- Object 1:**
 - Object Name: Room 472
 - Object Description: Floor 4
 - OBJECT PROPERTIES:
 - floor (integer): 3

At the bottom, there are 'Previous' and 'Next' buttons.

Рисунок 26 – Мастер создания проекта. Конфигурация объектов наблюдения

8.1.3 Описание адаптеров

The screenshot displays the 'Adapters' configuration page. At the top, a navigation bar includes links for Project, Objects, Adapters (active), Sensors, Aggregations, Alarms, Rule, and Success. The main content area is divided into two sections for 'Adapter: 0' and 'Adapter: 1'. Each section contains a 'Monitoring Object' dropdown, an 'Adapter name' text field, a 'Message Protocol' dropdown, a 'Username' text field, and a 'Password' text field with a toggle for visibility. Below these fields is an 'ADAPTER PROPERTIES' section with a '+ Add property' button. At the bottom of the page, there is a '+ Add Adapter' button and 'Previous' and 'Next' navigation buttons.

Рисунок 27 – Мастер создания проекта. Конфигурация адаптеров

8.1.4 Описание датчиков (сенсоров)

- Выбираем объект, в данном случае Room 384
- Выбираем адаптер, в данном случае MQTT
- Создаем топик (так как ещё нет созданных топиков). Можно указать группу
- Создаем несколько датчиков – датчик для измерения температуры в гостевой, на кухне; датчик для измерения влажности воздуха в гостевой
- Датчики для измерения температуры пишут в один топик, датчик для измерения влажности воздуха – в другой

Project — Objects — Adapters — Sensors — Aggregations — Alarms — Rule — Success

Sensor: 0

Monitoring Object

Room 384

Adapter

Room 384 adapter (MQTT)

Sensor name

temperature_living_room

Topic

New Topic

room384

temperature

TOPIC PREVIEW: room384/temperature

SENSOR PROPERTIES

+ Add property

Sensor: 1

Monitoring Object

Room 384

Adapter

Room 384 adapter (MQTT)

Sensor name

temperature_kitchen

Topic

New Topic

room384

temperature

TOPIC PREVIEW: room384/temperature

SENSOR PROPERTIES

+ Add property

Sensor: 2

Monitoring Object

Room 384

Adapter

Room 384 adapter (MQTT)

Sensor name

air_humidity_living_room

Topic

New Topic

room384

air_humidity

TOPIC PREVIEW: room384/air_humidity

SENSOR PROPERTIES

+ Add property

+ Add Sensor

Previous

Next

Рисунок 28 – Мастер создания проектов. Конфигурация сенсоров

8.1.5 Описание агрегационных правил

Создаем три агрегации

1. Средняя температура в номере (в гостиной и кухне) за последние 5 минут

2. Максимальная температура в гостиной за последние 5 минут
3. Среднее значение влажности воздуха в гостиной за последний час

The screenshot displays the 'Project' configuration page in a web application. The top navigation bar includes tabs for Project, Objects, Adapters, Sensors, Aggregations, Alarms, Rule, and a Success indicator. The main content area is titled 'Aggregation: 0' and contains three aggregation rules, each with an 'Aggregation Name' and an 'Aggregation Formula' field.

- Aggregation: 0**
Aggregation Name: temperature_avg_5m
Aggregation Formula: (avg('temperature_living_room', '5m') + avg('temperature_kitchen', '5m')) * 0.5
- Aggregation: 1**
Aggregation Name: temperature_living_room_max_5m
Aggregation Formula: max('temperature_living_room', '5m')
- Aggregation: 2**
Aggregation Name: air_humidity_living_room_avg_1h
Aggregation Formula: avg('air_humidity_living_room', '1h')

At the bottom of the aggregation list, there is a '+ Add Aggregation' button and a 'Previous' button, with a 'Next' button highlighted in blue.

Рисунок 29 – Мастер создания проекта. Конфигурация агрегационных метрик

8.1.6 Описание оповещений

Создаем четыре оповещения

1. Если среднее значение температуры в гостиной за последнюю минуту больше 27, то создать оповещение уровня Major
2. Если последнее значение температуры в гостиной больше 35, то создать оповещение уровня Critical
3. Если среднее значение температуры в гостиной за последнюю минуту меньше 18, то создать оповещение уровня Major
4. Если последнее значение температуры в гостиной меньше 14, то создать оповещение уровня Critical

The screenshot displays the 'Project' master configuration interface, specifically the 'Alarms' section. The breadcrumb trail at the top indicates the navigation path: Project > Objects > Adapters > Sensors > Aggregations > Alarms > Rule > Success. The interface shows four alarm rules, each with a collapse/expand toggle and a close button.

- Alarm 0:**
 - Alarm Name: `temperature_living_room_1m_high`
 - Alarm Condition: `avg('temperature_living_room', '1m') > 27`
 - Alarm Severity: MAJOR
 - + Add Selector
- Alarm 1:**
 - Alarm Name: `temperature_living_room_latest_high`
 - Alarm Condition: `latest('temperature_living_room') > 35`
 - Alarm Severity: CRITICAL
 - + Add Selector
- Alarm 2:**
 - Alarm Name: `temperature_living_room_1m_low`
 - Alarm Condition: `avg('temperature_living_room', '1m') < 18`
 - Alarm Severity: MAJOR
 - + Add Selector
- Alarm 3:**
 - Alarm Name: `temperature_living_room_latest_low`
 - Alarm Condition: `latest('temperature_living_room') < 14`
 - Alarm Severity: CRITICAL
 - + Add Selector
 - + Add Alarm

At the bottom of the interface, there are 'Previous' and 'Next' buttons.

Рисунок 30 – Мастер создания проекта. Конфигурация Оповещений (Alarm)

8.1.7 Описание правил

Создаем два правила

1. Условие: создание оповещения с названием `temperature_living_room_1h_high`. Действие: отправка внешней нотификации в Telegram, с заданным шаблоном

2. Условие: создание оповещения с названием temperature_living_room_latest_high. Действие: отправка внешней нотификации в Telegram, с заданным шаблоном

The screenshot displays the 'Rule' configuration page in the Project Master interface. It shows two rules, Rule 0 and Rule 1, configured for alarm processing.

Rule 0 Configuration:

- Condition:** On Alarm Raised. Selector: alarm_name, Value: temperature_living_room_tm_high.
- Actions:**
 - Action:** External Notification
 - Notification Service:** Telegram
 - Message Template:** [[[Alarm.Timestamp]]] [[[Alarm.Severity]]] received alarm '{{ Alarm.AlarmName }}'. Temperature is high. Please check {{ Alarm.SensorName }} in {{ Alarm.ObjectName }}
 - Recipients:** 811071503
 - Telegram API Host:** (empty)
 - Telegram API Token:** 6051006612AAE7vVUqh34taKZvap8GW3vokGt5vx3ctw

Rule 1 Configuration:

- Condition:** On Alarm Raised. Selector: alarm_name, Value: temperature_living_room_latest_high.
- Actions:**
 - Action:** External Notification
 - Notification Service:** Telegram
 - Message Template:** [[[Alarm.Timestamp]]] [[[Alarm.Severity]]] received alarm '{{ Alarm.AlarmName }}'. Temperature is VERY high. Please check {{ Alarm.SensorName }} in {{ Alarm.ObjectName }} ASAP
 - Recipients:** 811071503
 - Telegram API Host:** (empty)
 - Telegram API Token:** 6051006612AAE7vVUqh34taKZvap8GW3vokGt5vx3ctw

At the bottom of the interface, there are 'Previous' and 'Success' buttons.

Рисунок 31 – Мастер создания проекта. Конфигурация правил обработки Оповещений

8.1.8 Оповещение об успешном создании проекта

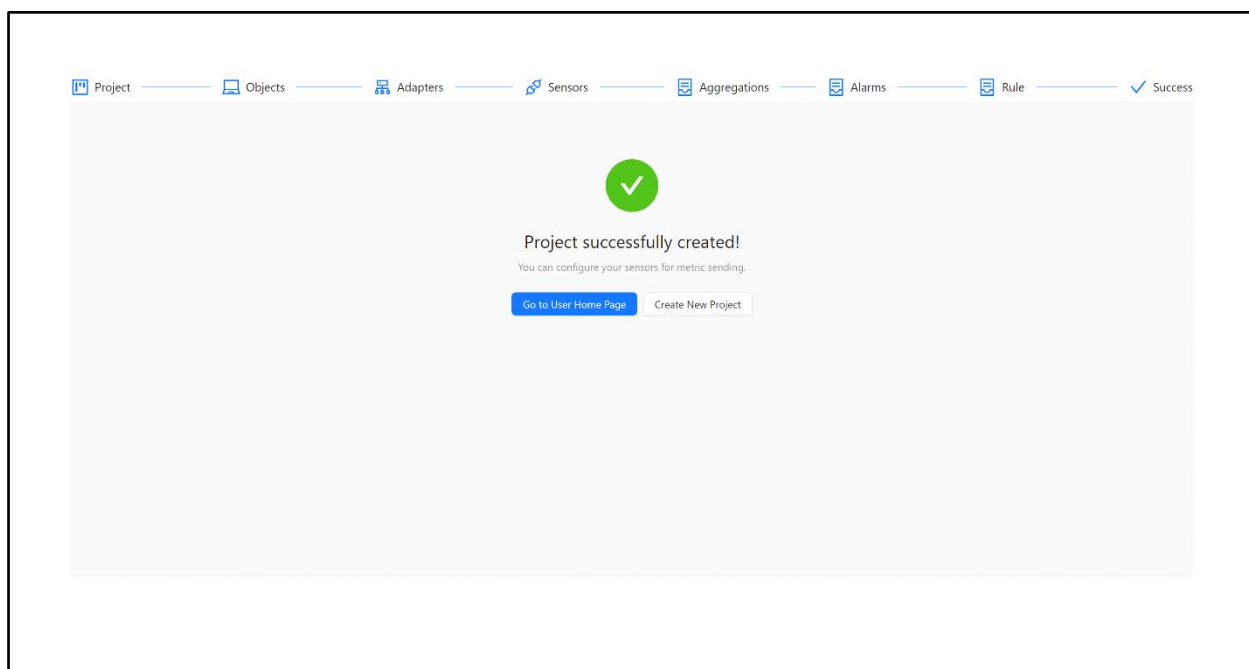


Рисунок 32 – Мастер создания проекта. Успешное создание конфигурации проекта

8.1.9 Реакция сервисов на созданную конфигурацию проекта

На стороне **metrics-engine** создались и применились следующие правила агрегаций и оповещений.

```
[
  {
    "aggregation_name": "temperature_avg_5m",
    "formula": "(avg('temperature_living_room', '5m') +
avg('temperature_kitchen', '5m')) * 0.5"
  },
  {
    "aggregation_name": "temperature_living_room_max_5m",
    "formula": "max('temperature_living_room', '5m')"
  },
  {
    "aggregation_name": "air_humidity_living_room_avg_1h",
    "formula": "avg('air_humidity_living_room', '1h')"
  }
]
```

```
[
  {
    "alarm_name": "temperature_living_room_1m_high",
    "condition": "avg('temperature_living_room', '1m') > 27",
    "severity": "Major"
  },
  {
    "alarm_name": "temperature_living_room_latest_high",
    "condition": "latest('temperature_living_room') > 35",
    "severity": "Critical"
  },
  {
    "alarm_name": "temperature_living_room_1m_low",
    "condition": "avg('temperature_living_room', '1m') < 18",
    "severity": "Major"
  },
  {
    "alarm_name": "temperature_living_room_1m_low",
    "condition": "latest('temperature_living_room') < 14",
    "severity": "Critical"
  }
]
]
```

На стороне **policy-engine** создалась и применилась следующая конфигурация с правилами.

```
[
  {
    "condition": {
      "condition_name": "alarm_raised",
      "selector": {
        "alarm_name": "temperature_living_room_1m_high"
      }
    },
    "actions": [
      {
        "action_name": "external_notification",
        "properties": {
          "notification_type": "telegram",
          "message_template": "[{{.Alarm.Timestamp}}] [{{.Alarm.Severity}}]. received alarm `{{.Alarm.AlarmName}}`. Temperature is high. Please check {{index .Alarm.Properties \"sensor_name\"}} in {{index .Alarm.Properties \"object_name\"}}",
          "recipients": [
            "811071503"
          ],
          "additional_properties": {
            "token":
"6051006612:AAE7vVUqh34taKZrvap8GW3vokGt9wx3ctw"
          }
        }
      }
    ]
  }
]
```



```

    }
  }
}
],
{
  "condition": {
    "condition_name": "alarm_raised",
    "selector": {
      "alarm_name": "temperature_living_room_latest_high"
    }
  },
  "actions": [
    {
      "action_name": "external_notification",
      "properties": {
        "notification_type": "telegram",
        "message_template": "[{{.Alarm.Timestamp}}]
[{{.Alarm.Severity}}]. received alarm `{{ .Alarm.AlarmName }}`.
Temperature is VERY high. Please check {{index .Alarm.Properties
\"sensor_name\"}} in {{index .Alarm.Properties \"object_name\"}}
ASAP",
        "recipients": [
          "811071503"
        ],
        "additional_properties": {
          "token":
"6051006612:AAE7vVUqh34taKZrvap8GW3vokGt9wx3ctw"
        }
      }
    }
  ]
}
]

```

8.2 Отправка сообщения с метрикой и его обработка системой

Для эмуляции отправки сообщения сенсором по MQTT протоколу, отправим следующее сообщение на HTTP эндпоинт сервиса metrics-engine для получения метрик.

```

{
  "username": "room384",
  "topic": "/room384/temperature",
  "payload": {
    "__name__": "temperature_living_room",
    "__value__": 40,
    "__timestamp__": "{{ $isoTimestamp }}",
    "__project_id__": "5433eca5-e099-4ef6-8487-
aa9537a2f476",

```

```

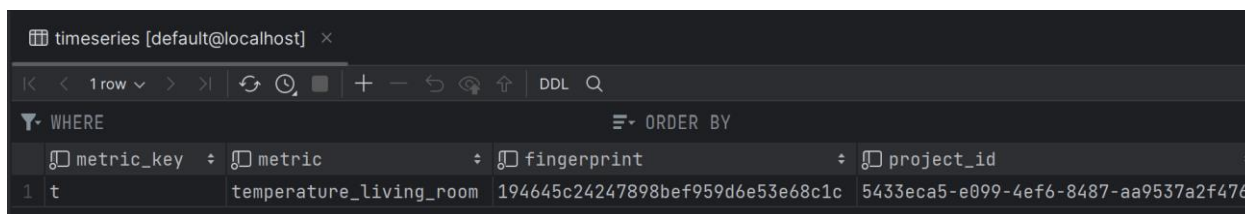
    "_object_id_": "802e3c41-1d56-4208-9eb8-752e0353495b",
    "_sensor_id_": "d2c87c41-9bf7-4ab0-8399-088d56d1e73f",
    "project_name": "Demo Hotel",
    "object_name": "Room 384",
    "sensor_name": "Temperature (Living Room)"
  }
}

```

8.2.1 Обработка сообщения с метрикой сервисом metrics-engine

После отправки сообщения соответствующая метрика должна сохраниться в аналитической базе данных.

Соответствующая запись появилась в таблице **timeseries** аналитической базы данных. На рисунке 33 результат выполнения SELECT запроса в данную таблицу.

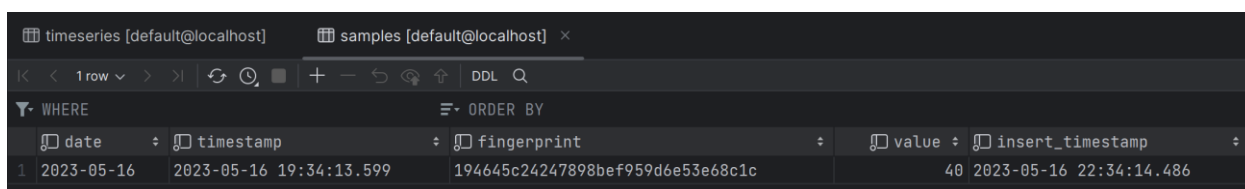


The screenshot shows a database interface with a query result for the **timeseries** table. The query is `SELECT * FROM timeseries`. The result shows one row with the following data:

	metric_key	metric	fingerprint	project_id
1	t	temperature_living_room	194645c24247898bef959d6e53e68c1c	5433eca5-e099-4ef6-8487-aa9537a2f476

Рисунок 33 – Таймсерия сохранена в таблице **timeseries** аналитической базы данных сервиса metrics-engine

Само непосредственное значение метрики сохранено в таблице **samples** аналитической базы данных. На рисунке 34 результат выполнения SELECT запроса в данную таблицу.



The screenshot shows a database interface with a query result for the **samples** table. The query is `SELECT * FROM samples`. The result shows one row with the following data:

	date	timestamp	fingerprint	value	insert_timestamp
1	2023-05-16	2023-05-16 19:34:13.599	194645c24247898bef959d6e53e68c1c	40	2023-05-16 22:34:14.486

Рисунок 34 – Семпл сохранен в таблице **samples** аналитической базы данных сервиса metrics-engine

При обработки данного сообщения должно создаться два оповещения (**temperature_living_room_1m_high**, **temperature_living_room_latest_high**), сконфигурированных ранее, так как присланная метрика подходит под

указанные условия. Созданные оповещения отправляются в Kafka топик *alarms_processing* для последующей обработки сервисом **alarms-engine**.

Отправленные сообщения с оповещениями можно наблюдать в пользовательском интерфейсе АКНQ для Kafka на рисунке 35.

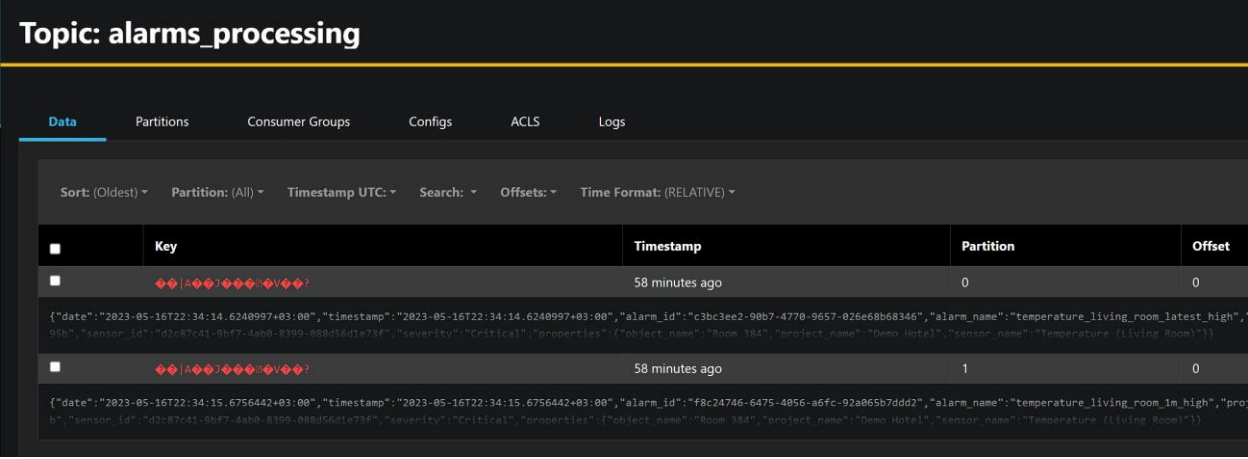


Рисунок 35 – Интерфейс АКНQ. Записанные сообщения с Оповещениями в топик *alarms_processing*

8.2.2 Обработка сообщений с оповещениями сервисом **alarms-engine**

Сервис считывает сообщения из Kafka топика *alarms_processing* и записывает оповещения в таблицу **alarms** аналитической базы данных. На рисунке 34 результат выполнения SELECT запроса в данную таблицу.



Рисунок 36 – Оповещения сохранены в таблице *alarms* аналитической базы данных сервиса **alarms-engine**

После записи оповещений в базу данных сервис отправляет оповещения в Kafka топик *alarms_post_processing* для последующей обработки сервисом **policy-engine**.

Отправленные сообщения с оповещениями можно наблюдать в пользовательском интерфейсе АКНQ для Kafka на рисунке 35.

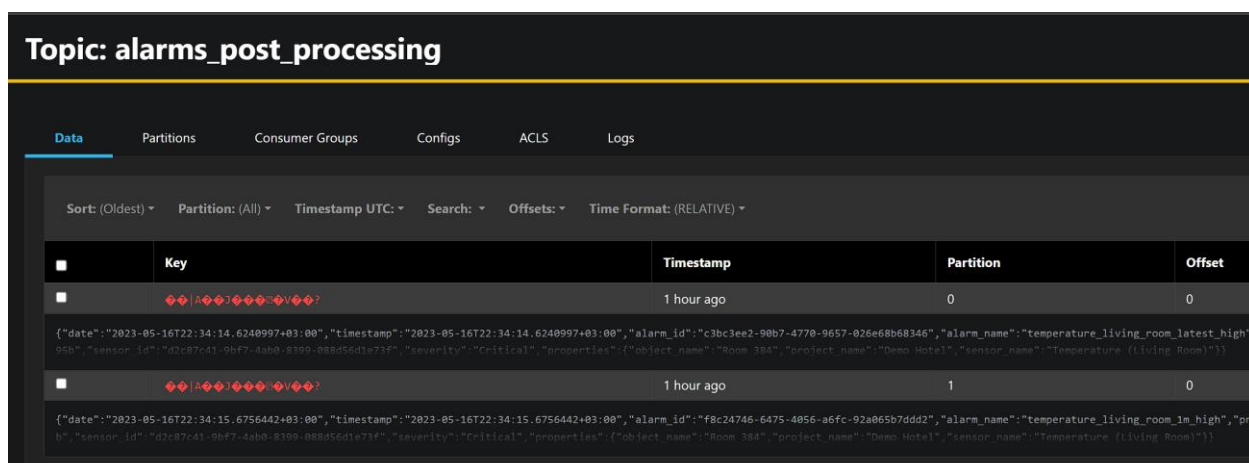


Рисунок 37 – Интерфейс АКНQ. Записанные сообщения с оповещениями в топик `alarms_post_processing`

8.2.3 Обработка сообщений с оповещениями сервисом `policy-engine`

Сервис считывает сообщения из Kafka топика `alarms_post_processing` и проверяет, подходит ли оповещения под созданные правила.

В данном случае для обоих оповещений созданы подходящие правила, поэтому должно выполниться условие – отправка сообщения через Telegram.

Сначала формируется сообщение в соответствии с указанным в конфигурации правила шаблоном (`message_template`). Вместо плейсхолдеров вида `{{ .Alarm... }}` подставляется актуальные сообщения из оповещения.

Затем **policy-engine** отправляет запрос на сервис **external-notification-manager** с сформированным сообщением и указанными параметрами в конфигурации для отправки сообщения в Telegram.

8.2.4 Отправка уведомлений сервисом `external-notification-manager`

Получив запрос на отправку уведомлений в Telegram, сервис выполняет запрос на Telegram Bot API с указанными в теле запроса параметрами.

Отправленные сообщения в Telegram можно наблюдать на рисунке 38.

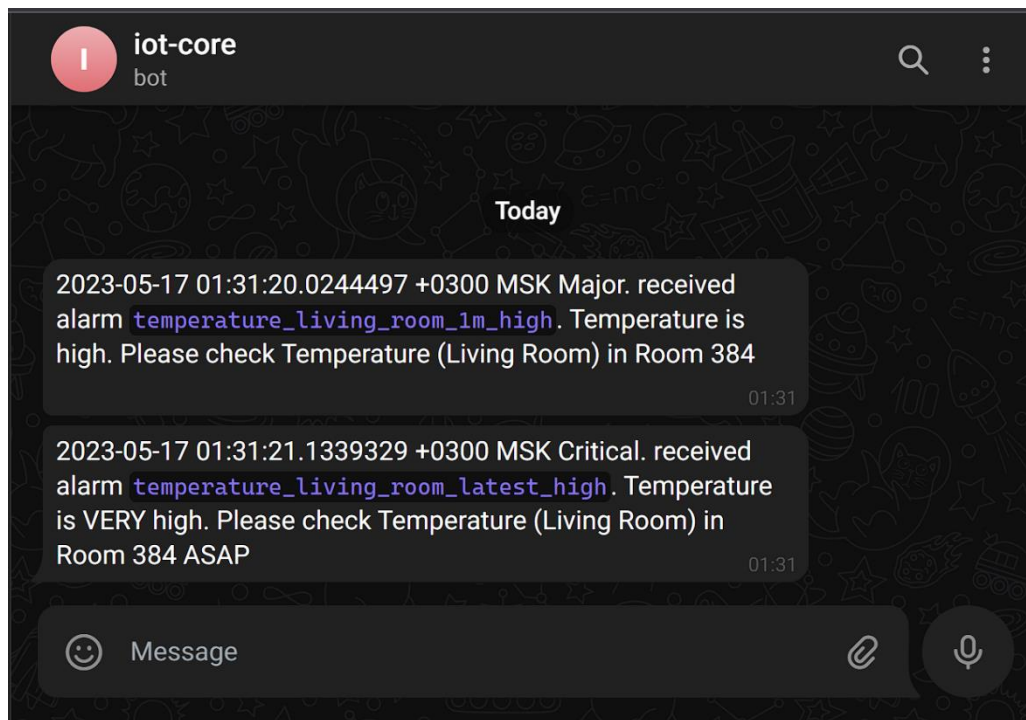


Рисунок 38 – Отправленные сообщения в Telegram

9 Дальнейшие планы развития платформы

Заключение

В рамках данной работы были выполнены разработка и реализация универсального ядра IoT-платформы с широким спектром функциональных возможностей, таких как масштабируемость, безопасность, управляемость и аналитические возможности. В ходе тестирования и оценки эффективности разработанного ядра платформы на реальных примерах была определена его конкурентоспособность и перспективность в развитии Интернета вещей.

Выделенные задачи для успешного подтверждения поставленной гипотезы, были успешно выполнены. Кроме того, были проанализированы различные аспекты разработки и применения IoT-платформы в различных отраслях (промышленность и потребительская сфера), что позволило оценить ее эффективность и применимость.

Таким образом, можно сделать вывод о том, что работа выполнена успешно, все задачи были выполнены, система проверена для различных отраслей, и гипотеза была подтверждена.

Разработанное ядро IoT-платформы может стать полезным инструментом для создания потребительских и промышленных приложений Интернета вещей и имеет перспективы в развитии этой области.

Список используемых источников

- 1 The Internet of Things / N.Gershenfeld, R.Krikorian, D.Cohen. // Scientific American. – 2004. – С. 76-81.
- 2 The Internet of Things. How the Next Evolution of the Internet Is Changing Everything / Dave Evans. // Cisco IBSG – 2011. – С. 1-6.
- 3 That «Internet of Things» Thing. In the real world, things matter more than ideas / Kevin Ashton. // RFID Journal – 2019. – С. 1-3.
- 4 The Internet of Things: A critique of ambient technology and the all-seeing network of RFID / Rob van Kranenburg. // Network Notebooks – 2008. – С. 10-19.
- 5 Global Industrial Internet of Things (IoT) Market 2021-2025: [Электронный ресурс]. URL: https://www.reportlinker.com/p06028692/Global-Industrial-Internet-of-Things-IoT-Market.html?utm_source=GNW (Дата обращения: 20.09.2022)
- 6 State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating: [Электронный ресурс]. URL: <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b> (Дата обращения: 20.09.2022)
- 7 Research on the architecture of Internet of things / M.Wu [и др.] // JOURNAL OF CRITICAL REVIEWS – 2020. – С. 1-3.
- 8 Интернет вещей / Р.Алгулиев, Р.Махмудов // Информационное сообщество – 2013. – С. 42-48.
- 9 Internet of Things (IoT) Products and Solutions: [Электронный ресурс]. URL: <https://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html> (Дата обращения: 13.10.2022)
- 10 Интернет вещей – Российская федерация | IBM: [Электронный ресурс]. URL: <https://www.ibm.com/ru-ru/cloud/internet-of-things> (Дата обращения: 14.10.2022)

- 11 Azure IoT — платформа для Интернета вещей | Microsoft Azure: [Электронный ресурс]. URL: <https://azure.microsoft.com/ru-ru/overview/iot/> (Дата обращения: 15.10.2022)
- 12 When Woman is Boss. An interview with Nikola Tesla by John B. Kennedy: [Электронный ресурс]. URL: <http://www.tfcbooks.com/tesla/1926-01-30.htm> (Дата обращения: 15.11.2022)
- 13 The little-known story of the first IoT device - Industrious: [Электронный ресурс]. URL: <https://www.ibm.com/blogs/industries/little-known-story-first-iot-device> (Дата обращения: 20.11.2022)
- 14 The toaster that changed the world - Article | The New Wave | BBC StoryWorks: [Электронный ресурс]. URL: <http://www.bbc.com/storyworks/future/the-new-wave/innovation> (Дата обращения: 15.11.2022)
- 15 Trojan Room Coffee Pot Biography: [Электронный ресурс]. URL: <https://www.cl.cam.ac.uk/coffee/qsf/coffee.html> (Дата обращения: 13.11.2022)
- 16 Why it is called Internet of Things: Definition, history, disambiguation: [Электронный ресурс]. URL: <https://iot-analytics.com/internet-of-things-definition/> (Дата обращения: 20.10.2022)
- 17 The Complete List Of Wireless IoT Network Protocols: [Электронный ресурс]. URL: <https://www.link-labs.com/blog/complete-list-iot-network-protocols> (Дата обращения: 25.11.2022)
- 18 Rightech - Just Do IoT: [Электронный ресурс]. URL: <https://rightech.io/> (Дата обращения: 05.12.2022)
- 19 Надежное облако Яндекс для вашего бизнеса — Yandex.Cloud: [Электронный ресурс]. URL: <https://cloud.yandex.ru/> (Дата обращения: 05.12.2022)
- 20 Quarkus - Supersonic Subatomic Java: [Электронный ресурс]. URL: <https://quarkus.io/> (Дата обращения: 17.03.2023)

21 PostgreSQL: The world's most advanced open-source database: [Электронный ресурс]. URL: <https://www.postgresql.org/> (Дата обращения: 26.03.2021)

22 Что такое ClickHouse: [Электронный ресурс]. URL: <https://clickhouse.com/docs/ru> (Дата обращения: 26.03.2021)

23 MQTT Version 3.1.1: [Электронный ресурс]. URL: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html> (Дата обращения: 04.03.2023)

24 STOMP Protocol Specification, Version 1.2: [Электронный ресурс]. URL: <https://stomp.github.io/stomp-specification-1.2.html> (Дата обращения: 04.03.2023)

25 Keycloak: [Электронный ресурс]. URL: <https://www.keycloak.org/> (Дата обращения: 10.03.2023)

26 Что такое Kubernetes | Kubernetes: [Электронный ресурс]. URL: <https://kubernetes.io/ru/docs/concepts/overview/what-is-kubernetes/> (Дата обращения: 20.03.2023)

27 Information Technology – Open System Interconnection – System Management: Alarm Reporting Function. Recommendation X.733 / Data Communication Networks. // International Telecommunication Union. – 1992. – С. 8-9.