Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ФИЗИКИ

КАФЕДРА РАДИОЭЛЕКТРОНИКИ

Направление: 03.03.03 – Радиофизика
Профиль: Радиофизические методы по областям применения (Радиофизические измерения)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«РАЗРАБОТКА КОМПЛЕКСА УДАЛЕННОГО МОНИТОРИНГА И УПРАВЛЕНИЯ ТЕХНОЛОГИЧЕСКИМ ОБОРУДОВАНИЕМ НА ПРИМЕРЕ СЕРВИСНОГО ОБСЛУЖИВАНИЯ И ЭКСПЛУАТАЦИИ ХОЛОДИЛЬНЫХ АГРЕГАТОВ»

Студент 4 курса группы 06-521
«19» июня 2019 г. _____ (Егоров Д.В)

Научный руководитель
к.ф.-м.н., доцент
«19» июня 2019 г. _____ (Гаврилов А.Г.)

Консультанты:
«19» июня 2019 г. _____ (Деркач А.И.)

«19» июня 2019 г. _____ (Маценко В.А.)

Заведующий кафедрой радиоэлектроники
д.ф.-м.н., доцент
«19» июня 2019 г. _____ (Овчинников М.Н.)

Казань-2019

Оглавление

Введение 4
Глава 1. Концепция Интернет Вещей (IoT) и промышленного Интернета Вещей (IIoT) 6
1.1 Введение в облачные сервисы и их классификация 7
1.1.1 SaaS - software as a service 7
1.1.2 IaaS - Infrastructure-as-a-Service 9
1.1.2.1 Характеристики и провайдеры IaaS 9
1.1.3 PaaS - Platform as a Service 11
1.1.3.1 Ключевые критерии выбора PaaS 12
Глава 2. Базы данных 15
2.1 Введение в базы данных 15
2.2 База данных SQLite 16
2.3 База данных MongoDB 17
2.3.1 Формат данных в MongoDB 18
2.3.2 Кроссплатформенность 19
2.3.3 Документы вместо строк 19
2.3.4 Коллекции 20
2.3.5 Репликация 20
2.3.6 Простота в использовании 20
2.3.7 GridFS 21
Глава 3. NODE-RED – инструмент визуального программирования для Промышленного Интернета Вещей (IIoT) 22
3.1 NODE.JS 22
3.1.1 Преимущества и важные нюансы 23
3.2 NODE-RED 24
3.2.1 Ключевые идеи 26
3.2.1.1 Потоки 26
3.2.1.2 Сообщения 26
3.2.1.3 Узлы 27
3.2.1.4 «Провода» 29
3.2.1.5 Контекст 29

3.2.1.6 Функциональные узлы 30
Глава 4. Применение концепции Промышленного Интернета вещей (IIoT) для построения системы телемеханики и телеметрии распределенных промышленных объектов. 32
4.1 Структурная схема проекта 32
4.2 Имитационная модель технологического объекта 33
4.3 Серверная часть Web-приложения 35
4.3.1 Начало работы с NODE-RED 35
4.3.2 Дополнительная установка узлов 36
4.3.3 Архивация данных 38
4.3.4 Получение исторических данных 39
4.3.5 Аварийные оповещения 40
4.4 Клиентская часть Web-приложения 42
4.4.1 Управление и мониторинг 42
4.4.2 Вывод исторических данных 47
4.4.3 Вывод аварийных состояний 48
Заключение 50
Список литературы 51
Приложение А ..... **Ошибка! Закладка не определена.**
Приложение Б ..... **Ошибка! Закладка не определена.**
Приложение В ..... **Ошибка! Закладка не определена.**
Приложение Г ..... **Ошибка! Закладка не определена.**

Введение

Большой вклад в науку и жизнь в целом принес Интернет Вещей. Это поспособствовало появлению новых технологий, оборудованию, развитию промышленности и т.д. Если раньше, например, при обслуживании какого-либо оборудования, которое находится на большом расстоянии друг от друга, приходилось тратить не мало времени, то теперь это можно осуществлять при помощи приложения, которое находится на вашем электронном устройстве.

Безусловно, автоматизация началась уже давно, но она была не такой эффективной.

Теперь же, с развитием интернета появилась возможность разработать несложное ПО и оно будет взаимодействовать при помощи облачных сервисов, работоспособность которых почти 100%.

Конечно же, при работе с Интернетом Вещей есть свои минусы:

- У вас может пропасть связь с Интернетом;
- У вас может перестать работать электронное устройство;
- Функционал некоторых сервисов очень ограничен в бесплатном варианте и для его расширения приходится платить.

Но это все эти минусы могут перекрыть несколько больших плюсов:

- Вы можете управлять процессами находясь на другом конце Земного шара;
- Безопасность. Многие профессии очень опасны для жизни и Интернет Вещей может в этом помочь, ведь можно автоматизировать те процессы, которые грозят большой опасностью для людей.

Целью настоящей работы является разработать и отладить комплекс удаленного мониторинга и управления технологическим оборудованием на примере сервисного обслуживания и эксплуатации холодильных агрегатов.

Для реализации поставленной цели необходимо было сделать следующее:

- Освоить инструментальное средство визуального программирования NODE-RED;

- Ознакомиться и освоить протокол mqtt, а также облачный сервис CloudMQTT для передачи данных;
- Освоить языки программирования JavaScript, CSS, а также язык разметки HTML;
- Ознакомится и освоить работу с системой управления базами данных MongoDB.

Глава 1. Концепция Интернет Вещей (IoT) и промышленного Интернета Вещей (IIoT)

IoT или Интернет вещей — это сеть связанных через Интернет объектов, способных собирать данные и обмениваться данными, поступающими со встроенных сервисов. [1]

Концепция Интернета вещей базируется на принципе межмашинного общения: без вмешательства человека электронные устройства «общаются» между собой. Интернет вещей — это автоматизация, но более высокого уровня. В отличие от «умных» домов, узлы системы используют TCP/IP-протоколы для обмена данными через каналы глобальной сети Интернет. [2]

Такой метод коммуникации дает серьезное преимущество — возможность объединять системы между собой, строить «сеть сетей». Это позволяет изменить бизнес-модели отраслей и даже экономики целых стран. [2]

Менее чем за 20 лет Интернет вещей стал трендом рынка информационных технологий. Аналитики прогнозируют колоссальное количество IoT устройств через несколько лет — свыше 50 миллиардов. Развитие производства электронных компонентов позволяет «штамповать» миллионы дешевых чипов для всевозможных устройств. От радиочипов, нанесенных на складские коробки, IoT трансформировался в глобальную «интернетизацию» окружающих нас предметов, воспринимаемый людьми как глобальная «оцифровка» реальности.

В недалеком будущем интернет вещей станет тем инструментом, с помощью которого можно дешево, быстро и масштабно решать конкретные бизнес-задачи в конкретных отраслях. [2]

Промышленный IoT (Industrial IoT, IIoT) объединяет концепцию межмашинного общения, использование BigData и проверенные технологии автоматизации производства. Ключевая идея IIoT в превосходстве «умной» машины над человеком в точном, постоянном и безошибочном сборе

информации. Интернет вещей повысит уровень контроля качества продукции, выстроит процесс бережливого и экологичного производства, обеспечит надежные поставки сырья и оптимизирует работу заводского конвейера. [2]

В отличие от традиционного «человеческого» Интернета, Интернет вещей применяется для рационального и практичного подхода. Его ключевая задача — автоматизация, оптимизация, сокращение материальных и временных затрат.

Применение IoT в промышленной индустрии и транспорте сокращает затраты за счет снижения аварийности, уменьшения потерь сырья и количества использованных ресурсов. В сфере энергетики — повышает эффективность выработки и распределения электроэнергии. [2]

Интернет вещей экономит не только деньги, но и время: машины заменили человека на рутинной работе и освободили от выполнения рискованных или стандартных задач. Интеллектуальные системы следят за промышленным конвейером, считают товар на складах и регулируют движение вместо человека. В любую погоду, круглосуточно и без выходных. [2]

1.1 Введение в облачные сервисы и их классификация

В самом простом понимании облачный сервис представляет собой не что иное, как выделяемое дисковое пространство на удаленном сервере, где пользователь может сохранить свои файлы и папки, загрузить их заново на свой компьютер, производить простейшие действия по управлению ими или открыть общий доступ для того, чтобы ими мог воспользоваться кто-то другой. Также облачный сервис может использоваться просто для передачи информации без ее сохранения. [3]

1.1.1 SaaS - software as a service

Данное обозначение расшифровывается как Software as a Service и представляет собой модель работы с бизнес-приложениями в виде Интернет-сервисов. Если говорить о цели SaaS, что это такое и как реализуется, необходимо уделить вниманию и специфике функционирования. Непосредственно само приложение находится на сервере провайдера, а пользователи получают к нему доступ посредством Интернет-браузера. Ещё одной характерной особенностью является то, что программный продукт не покупают, а берут в аренду. Таким образом, решается сразу два момента. Первый – экономят на программном обеспечении. Второй момент – поскольку приложение постоянно обновляется, есть доступ к самому наилучшему функционалу и широким возможностям. На плечах провайдера лежит забота о работоспособности приложения и технической поддержке пользователей. Благодаря свободному времени силы компании можно сосредоточить на достижении поставленных целей. [4]

Основные преимущества SaaS:

- Низкая стоимость владения;
- Небольшие сроки внедрения;
- Низкий порог требований для реализации (часто дополняется ещё и возможностью ограниченного периода бесплатного использования);
- Технические аспекты вроде поддержки и обновления ложатся на плечи провайдера;
- Мобильность пользователя в плане использования приложения;
- Удобно использовать географически удалённым подразделениям компаний;
- Невысокие требования к характеристикам компьютера;
- Универсальность работы на любой соответствующей технике.

Есть ли недостатки у SaaS?

Сервис этот, увы, не полностью совершенен. И для полноты картины необходимо уделить внимание и недостаткам. Как правило, в качестве таковых

называют ненадёжность доступа из-за помех Интернета, невысокое быстродействие и передачу коммерчески важных данных сторонней компании. В качестве средств компенсации этих недостатков называют широкополосной доступ к сети Интернет, качественные технологии шифрования и хороший имидж провайдера. Что тут поделаешь. Это не отшлифованное оружие вроде Umarex Colt SAA, которое должно работать без осечки. Но всё же от программного обеспечения может зависеть качество экономической жизни. [4]

1.1.2 IaaS - Infrastructure-as-a-Service

Инфраструктура как услуга (IaaS) — это схема, при которой различные компоненты облачной инфраструктуры вроде серверов, хранилищ данных, операционных систем и сетевых ресурсов предоставляются в качестве подключаемой услуги. Вместо того, чтобы покупать оборудование, софт, стойки в дата-центре, компании просто по мере необходимости приобретают эти ресурсы (и в дальнейшем отказываются от них) у сторонних провайдеров. Существует несколько подкатегорий IaaS — получение услуг может осуществляться с помощью публичного или частного облака, а также комбинации этих подходов. [5]

1.1.2.1 Характеристики и провайдеры IaaS

Понятие инфраструктуры как услуги включает в себя несколько основных характеристик:

- Ресурсы распространяются в качестве услуги;
- Существует возможность динамического расширения (и сокращения) объёмов потребляемых ресурсов;
- Реализованы гибкие модели оплаты (например, оплата только за фактически потребленные ресурсы — модель pay as you go);

- Как правило, с одним физическим «железным» ресурсом работают несколько пользователей. [5]

В мире существует огромное количество провайдеров IaaS — среди самых известных, к примеру, Amazon Web Services, помимо этого на региональных рынках присутствуют свои сильные игроки.

Когда стоит использовать IaaS?

В том случае, если у специалистов конкретной компании есть опыт создания и администрирования ИТ-инфраструктуры, в определенных условиях использование IaaS будет оптимальным вариантом. Вот несколько наиболее распространенных случаев:

- Потребность компании в ИТ-ресурсах не одинакова в разные моменты времени — бывают сильные всплески, когда потребности вырастают, но они не длятся долго;
- Компания — это стартап, у которого просто нет денег на покупку оборудования и создания собственной инфраструктуры;
- Организация быстро растет, и возникает проблема постоянного масштабирования инфраструктуры;
- Существует потребность в снижении операционных издержек, в том числе на оборудование;
- При запуске нового экспериментального направления бизнеса, которое нужно протестировать или которое не требует постоянного выделения ИТ-ресурсов.

Когда не стоит использовать IaaS?

Несмотря на гибкость и масштабируемость IaaS, у этой технологии есть определенные ограничения, и существуют ситуации, когда ее использование проблематично:

- Если компания является игроком регулируемой отрасли, правила которой не разрешают хранение данных на серверах, не принадлежащих компании (и часто находящихся в другой стране);

- IaaS может не подойти тем компаниям, которым требуется высочайший уровень производительности — его проще достигнуть с помощью использования выделенных инфраструктурных ресурсов (hosted infrstructure). [5]

1.1.3 PaaS - Platform as a Service

PaaS (Platform as a Service) – это модель предоставления облачных услуг, которая дает потребителю возможность разворачивать в облаке приложения, разработанные с использованием конкретных языков программирования, программных библиотек, сервисов и инструментов, поддерживаемых облачным провайдером. Потребителю PaaS не требуется контролировать уровень облачной инфраструктуры IaaS (Infrastructure as a Service), но он имеет все возможности для управления, конфигурирования и развертывания разработанных приложений, которыми может пользоваться сам или предлагать своим конечным пользователям по модели SaaS (Software as a Service). [6]

Основные преимущества PaaS лежат в плоскости стоимости и практического удобства:

- Не нужно самим создавать и конфигурировать виртуальные машины, вы используете портал или скрипты, чтобы настроить базы данных. Если вам нужно хранилище определенного объема, вы просто двигаете слайдером на портале или запускаете соответствующий скрипт, и нужное хранилище практически сразу готово к использованию;
- Зачастую провайдер предоставляет возможность выбрать для PaaS конкретное решение (сделать высокоуровневый выбор), при этом низкоуровневая детализация все равно остается скрытой от вас;
- Автоматическое обновление СУБД провайдерами;

- Не требуется никаких усилий для масштабирования и обеспечения высокой доступности сервиса, всю ответственность берет на себя провайдер PaaS;
- Не нужно платить за лицензии СУБД, это включено в оплату сервиса;
- Вы платите только за то, что используете.

При использовании PaaS разработчики концентрируются на написании кода, а потом все отдают «на откуп» управляющему движку PaaS, который обеспечивает выполнение разработанных сервисов, их масштабирование и управление ими. [6]

Для разработчиков здесь важны:

- Автоматическая интеграция и согласованность разработки и развертывания;
- Большой выбор средств для разработки, готовых API и другого кода;
- Множество уже реализованных приложений, которые можно использовать при разработке собственного.

PaaS может предоставлять разработчикам различные интерфейсы доступа и управления: командную строку, интегрированную среду разработки IDE, доступ через веб-браузер или через REST API (REpresentational State Transfer). Наиболее продвинутые сервисы PaaS предоставляют все четыре опции. [6]

Использование PaaS приводит к упрощению и стандартизации процесса разработки, дает возможность приобретать готовые компоненты в магазине приложений облачного провайдера, поддерживающего данный PaaS, или на открытом рынке у компаний, которые используют средства разработки, совместимые с этим PaaS. Все это минимизирует инвестиции на этапе запуска проекта, сокращает время вывода облачного продукта на рынок и позволяет потом гибко подстраиваться под реальные потребности рынка. Такая ситуация идеальна для стартапов. [6]

1.1.3.1 Ключевые критерии выбора PaaS

При выборе платформы PaaS очень важно с самого начала учесть специфику и потребности будущих проектов, а также возможности разработчиков, чтобы не менять платформу потом, когда уже имеется много наработок. [6]

Итак, какие критерии следует рассматривать при выборе PaaS?

1) Уровень автоматизации.

   PaaS могут значительно отличаться друг от друга в части автоматизации. Очень часто именно быстрота предоставления элементов платформы в автоматическом режиме, без необходимости что-то делать самому или привлекать для этого провайдера, является основным фактором при выборе PaaS.

2) Гибкость настройки.

   Ряду проектов требуется серьезная кастомизация, т.е. настройка PaaS под эти проекты. Естественно, PaaS различаются количеством компонентов, которые могут быть кастомизированы, и способами, которыми могут быть сделаны изменения в компонентах.

3) Библиотеки шаблонов.

   При построении приложений, экземпляры которых могут различаться, существенно наличие в PaaS готовых библиотек шаблонов, учитывающих эти различия. Возможность использовать существующие шаблоны, чтобы подстраиваться под пользовательские запросы, заметно облегчает жизнь разработчикам.

4) Возможности миграции.

   Существенный фактор при выборе PaaS – возможность переноса приложений в облако. Разные PaaS имеют свои ограничения по миграции и предъявляют определенные требования к переносимым приложениям. Приложения могут существовать либо только в бинарных модулях, когда нет никакого доступа к исходному коду, либо могут быть жестко завязаны на API и платформу конкретного

вендора, в этом случае задача миграции может оказаться непростой. Если используются стандартизованные средства уровня middleware и runtime, и если к тому же это open source, то миграция значительно упрощается. А если приложения построены в микросервисной архитектуре в расчете на тот или иной формат контейнеризации, то миграция, как прямая, так и обратная, становится простой стандартной процедурой.

5) Универсальность (частный и публичный сервис).

Если есть жесткие требования информационной безопасности и приложение разрабатывается для нужд конкретной компании, то правильный выбор – частный PaaS. Частные облака обеспечивают полный контроль над всеми ресурсами PaaS. Если важнее стоимость и гибкая масштабируемость решения, то обычно проще использовать публичный PaaS. Иногда более удобна гибридная модель, когда одна и та же PaaS-платформа используется и в облаке провайдера, и развернута внутри компании. [6]

Глава 2. Базы данных

2.1 Введение в базы данных

Базой данных называют некое структурированное хранилище информации. БД - это также информационная модель, способная вмещать в себе некие данные, с условием, что они обязательно будут упорядочены. Каждый из нас работал с БД хоть раз, но мог об этом даже и не догадываться, например, вводя поисковый запрос, мы обращаемся к масштабной базе данных за конкретными сведениями. СУБД - это очередная аббревиатура, которую расшифровывают как "система управления базами данных". В общем смысле они представляют собой различные программные решения, при помощи которых можно организовывать данные БД. Под этим понимается заполнение базы информацией, упорядочивание ее, удаление, копирование, анализ и многое другое. [7]

Виды БД:

1) Реляционные базы данных (от английского слова relation, что переводится как "связь") - характеризируются отношениями и выражены в совокупности взаимосвязанных сущностей. Последние представлены в виде табличек, в которых содержатся данные БД. Это наиболее распространенный тип баз данных;
2) Иерархические - связи на уровне "предок-потомок", "начальник-подчиненный";
3) Сетевые - ответвление от предыдущего вида;
4) Объектно-ориентированные, которые напрямую работают с соответствующей методологией программирования (ООП). [7]

2.2 База данных SQLite

SQLite – это библиотека, которую можно встраивать в приложения. Полноценная БД на основе файлов SQLite предлагает широкий набор инструментов для обработки всех видов данных и накладывает намного меньше ограничений, чем другие реляционные базы данных. [8]

Приложения, использующие SQLite, не взаимодействуют с помощью интерфейса (портов, сокетов), а отправляют прямые запросы в файл, в котором хранятся данные (например, БД SQLite). Благодаря этому приложение SQLite очень быстрое и производительное. [8]

Типы данных SQLite:

- NULL: пустое значение;
- INTEGER: целочисленное значение (зависимо от объёма значение хранится в 1, 2, 3, 4, 6 или 8 байтах);
- REAL: число с плавающей точкой, хранится в виде 8-байтного IEEE;
- TEXT: текстовая строка, хранится в зашифрованном виде (UTF-8, UTF-16BE или UTF-16LE);
- BLOB: бинарные данные, хранятся в том виде, в котором были введены.

Преимущества SQLite:

1) Простое строение на основе файлов: вся база данных состоит всего из одного файла, что увеличивает её портативность;
2) Стандарты: несмотря на простоту, система SQLite основана на SQL. Некоторые функции опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), однако вместо них добавлены другие;
3) SQLite отлично подходит для разработки или тестирования. На этих этапах почти всегда необходимо простое, но масштабируемое решение.

Недостатки SQLite:

1) Нет управления пользователями. Более сложные СУБД поддерживают управление пользователями (их взаимосвязями, привилегиями и т.п.). Простая СУБД SQLite такой функции не предоставляет;
2) Невозможно повысить производительность. Библиотека SQLite проста в настройке и в использовании. Однако она разработана таким образом, что не позволяет путём тонкой настройки получить дополнительную производительность. То есть сделать SQLite более производительной технически невозможно.

Когда лучше использовать SQLite?

- Простые встроенные приложения, которым нужна портативность, например, однопользовательские локальные приложения, мобильные приложения, игры;
- Замена диска. Обычно приложения, которым необходимо читать или записывать файлы на диск, могут использовать SQLite для получения дополнительных функций;
- Тестирование.

Когда лучше не использовать SQLite?

- Многопользовательские приложения. Если приложение построено таким образом, что большое количество клиентов одновременно использует одну БД, то в такое приложение лучше внедрить полнофункциональную реляционную СУБД (например, MySQL);
- Приложения, записывающие большое количество данных. операция записи является одним из ограничений SQLite. Эта СУБД позволяет выполнять только одну операцию записи за один момент времени, следовательно, она ограничивает пропускную способность. [8]

2.3 База данных MongoDB

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных. [9]

Длительное время было обычным делом хранить все данные в реляционных базах данных (MS SQL, MySQL, Oracle, PostgresSQL). При этом было не столь важно, а подходят ли реляционные базы данных для хранения данного типа данных или нет. [9]

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать. Но, даже учитывая все недостатки традиционных баз данных и достоинства MongoDB, важно понимать, что задачи бывают разные и методы их решения тоже бывают разные. В какой-то ситуации MongoDB действительно улучшит производительность вашего приложения, например, если надо хранить сложные по структуре данные. В другой же ситуации лучше будет использовать традиционные реляционные базы данных. Кроме того, можно использовать смешенный подход: хранить один тип данных в MongoDB, а другой тип данных - в традиционных БД. [9]

Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Функциональность MongoDB позволяет расположить несколько баз данных на нескольких физических серверах, и эти базы данных смогут легко обмениваться данными и сохранять целостность. [9]

2.3.1 Формат данных в MongoDB

Одним из популярных стандартов обмена данными и их хранения является JSON (JavaScript Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в

MongoDB применяется формат, который называется BSON (БиСон) или сокращение от binary JSON. [9]

BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью. [9]

2.3.2 Кроссплатформенность

MongoDB написана на C++, поэтому ее легко портировать на самые разные платформы. MongoDB может быть развернута на платформах Windows, Linux, MacOS, Solaris. Можно также загрузить исходный код и самому скомпилировать MongoDB, но рекомендуется использовать библиотеки с официального сайта. [9]

2.3.3 Документы вместо строк

Если реляционные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений.

Ключ представляет простую метку, с которым ассоциировано определенный кусок данных. Однако, при всех различиях есть одна особенность, которая сближает MongoDB и реляционные базы данных. В реляционных СУБД встречается такое понятие как первичный ключ. Это понятие описывает некий столбец, который имеет уникальные значения. В MongoDB для каждого документа имеется уникальный идентификатор, который называется _id. И

если явным образом не указать его значение, то MongoDB автоматически сгенерирует для него значение. [9]

Каждому ключу сопоставляется определенное значение. Но здесь также надо учитывать одну особенность: если в реляционных базах есть четко очерченная структура, где есть поля, и если какое-то поле не имеет значение, ему (в зависимости от настроек конкретной бд) можно присвоить значение NULL. В MongoDB все иначе. Если какому-то ключу не сопоставлено значение, то этот ключ просто опускается в документе и не употребляется. [9]

2.3.4 Коллекции

Если в традиционном мире SQL есть таблицы, то в мире MongoDB есть коллекции. И если в реляционных БД таблицы хранят однотипные жестко структурированные объекты, то в коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств. [9]

2.3.5 Репликация

Система хранения данных в MongoDB представляет набор реплик. В этом наборе есть основной узел, а также может быть набор вторичных узлов. Все вторичные узлы сохраняют целостность и автоматически обновляются вместе с обновлением главного узла. И если основной узел по каким-то причинам выходит из строя, то один из вторичных узлов становится главным. [9]

2.3.6 Простота в использовании

Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему

значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов. [9]

2.3.7 GridFS

Одной из проблем при работе с любыми системами баз данных является сохранение данных большого размера. Можно сохранять данные в файлах, используя различные языки программирования. Некоторые СУБД предлагают специальные типы данных для хранения бинарных данных в БД (например, BLOB в MySQL). [9]

В отличие от реляционных СУБД MongoDB позволяет сохранять различные документы с различным набором данных, однако при этом размер документа ограничивается 16 мб. Но MongoDB предлагает решение - специальную технологию GridFS, которая позволяет хранить данные по размеру больше, чем 16 мб. [9]

Система GridFS состоит из двух коллекций. В первой коллекции, которая называется files, хранятся имена файлов, а также их метаданные, например, размер. А в другой коллекции, которая называется chunks, в виде небольших сегментов хранятся данные файлов, обычно сегментами по 256 кб. [9]

Глава 3. NODE-RED – инструмент визуального программирования для Промышленного Интернета Вещей (IIoT)

3.1 NODE.JS

Node.js — это платформа разработки программного обеспечения с открытым исходным кодом для выполнения кода JavaScript на стороне сервера. Node полезен для разработки приложений, которым требуется постоянное соединение браузера с сервером. Часто используется для приложений реального времени, таких как чат, новостные ленты и push-уведомления. Node.js предназначен для работы на выделенном HTTP-сервере и для использования одного потока с одним процессом за одну единицу времени. Начало работы в Node.js основано на событиях и выполняется асинхронно. Код, построенный на данной платформе, не соответствует традиционной модели приема, обработки, отправки, ожидания и получения. Вместо этого Node обрабатывает входящие запросы в стеке постоянных событий, отправляет небольшие запросы один за другим и не дожидается ответов. [10]

Начало Node.js — JavaScript.

Node базируется на JavaScript, в связи с чем понимание работы данной платформы невозможно без экскурса в JS. JavaScript — это интерпретируемый язык программирования или скриптов из Netscape. Языки сценариев JavaScript проще и быстрее кодировать, чем более структурированные и скомпилированные, такие как C и C++. Языки скриптов обычно занимают больше времени, чем компилируемые, но они очень полезны для коротких программ. JavaScript используется в разработке веб-сайта для выполнения таких задач, как автоматическое изменение форматированной даты на веб-

странице. Это причина, по которой связанная страница будет отображаться во всплывающем окне. [10]

Характерные особенности JS:

JavaScript обычно используется в веб-разработке. Он был первоначально разработан Netscape как средство для добавления динамических и интерактивных элементов на веб-сайты. Современный основан на ECMAScript — языке сценариев, разработанном Sun Microsystems. JavaScript — это язык сценариев на стороне клиента. Это означает, что исходный код обрабатывается веб-браузером клиента, а не веб-сервером. Функции могут запускаться после загрузки веб-страницы без связи с сервером. Например, функция JavaScript может проверять веб-форму перед отправкой, чтобы убедиться, что все необходимые поля заполнены. Код может выдать сообщение об ошибке, прежде чем какая-либо информация будет фактически передана на сервер. Подобно серверным скриптовым языкам, таким как PHP и ASP, код JavaScript может быть вставлен в любом месте HTML-страницы в веб. Однако в HTML отображается только вывод серверного кода, а код JavaScript остается полностью видимым в источнике веб-страницы. Его также можно найти в отдельном файле .JS, который также можно просмотреть в браузере. [10]

3.1.1 Преимущества и важные нюансы

Одним из основных преимуществ Node.js то, что он не блокирует ввод/вывод (I/O). Некоторые разработчики очень критично относятся к Node.js и отмечают, что если для одного процесса требуется значительное количество циклов процессора, приложение блокируется. Это может привести к сбою. Сторонники модели Node.js утверждают, что время обработки процессора меньше беспокоит из-за большого числа небольших процессов, на которых основан код узла. [10]

Популярность приложений JavaScript стремительно растет в последние несколько лет, а Node.js определенно способствует этому росту. Если мы посмотрим на статистические данные, то увидим, что в мире больше пакетов Node, чем аналогичных данных Ruby. Второй фактор: пакеты Node растут быстрее, чем Ruby, Python и Java. [10]

Что делает более популярным, чем Rails и другие альтернативы Node.JS? Node сам по себе является асинхронной платформой, основанной на событиях, построенной на базе движка JavaScript Chrome и предназначенной для создания масштабируемых сетевых приложений. Иными словами, Node.js — это JavaScript плюс C/C ++ в совокупности с файловой системой, запуском HTTP или TCP-серверов. [10]

Node однопоточен и использует модель параллелизма, основанную на цикле событий. Он не блокирует процессы и не заставляет программу ждать, но регистрирует обратный вызов и позволяет приложению продолжить работу. Это означает, что платформа может обрабатывать параллельные операции без нескольких потоков выполнения и эффективно масштабируется. [10]

Между последовательными языками и Node.js существуют большие различия:

- Node позволяет выполнять другие задачи, ожидая уведомления, когда ответ будет доступен;
- Приложение не буферизует данные в память, а вместо этого выводит его фрагментарно.

Хотя существуют другие системы циклов событий (например, библиотека EventMachine в Ruby или Twisted на Python), между ними и Node существует значительная разница.

В Node.JS все библиотеки были разработаны с нуля, чтобы быть неблокируемыми, чего нельзя сказать о других. [10]

3.2 NODE-RED

Node-RED - это мощный инструмент визуального программирования для создания приложений Internet of Things (IoT) с уделением особого внимания упрощению кода для выполнения задач. Он использует подход визуального программирования, который позволяет разработчикам подключать предопределенные кодовые блоки, известные как «узлы», вместе для выполнения задачи. Связанные узлы, как правило, комбинация входных узлов, узлов обработки и выходных узлов, когда они соединены вместе, составляют «потоки». Основной экран пользовательского интерфейса инструмента разделен на узлы, основной экран для их конфигурирования и блок отладки (рис.1). [11]

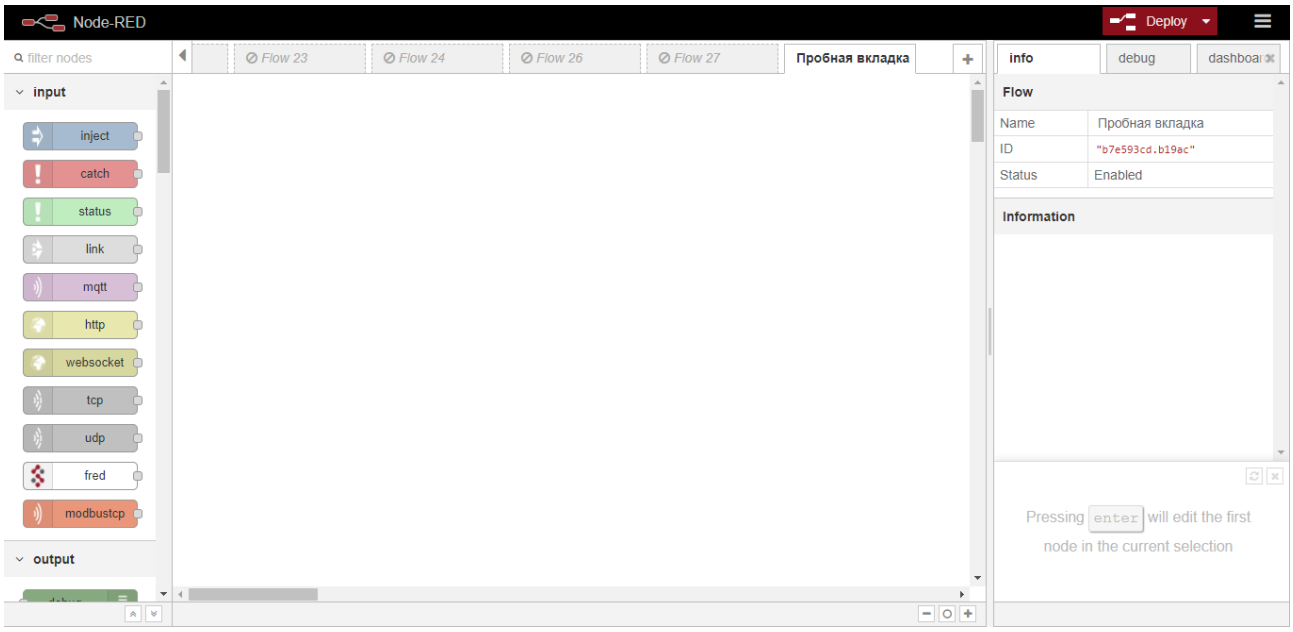

Рис.1. Графический интерфейс NODE-RED

Первоначально Node-RED был разработан как проект с открытым исходным кодом в IBM в конце 2013 года, чтобы удовлетворить потребность в быстром подключении оборудования и устройств к веб-сервисам и другому программному обеспечению - как своего рода клей для IoT - он быстро превратился в инструмент универсального программирования для IoT. Важно отметить, что Node-RED быстро развил значительную и растущую базу пользователей и активное сообщество разработчиков, которые вносят новые

узлы, которые позволяют программистам повторно использовать код Node-RED для самых разных задач. [11]

Хотя Node-RED изначально был разработан для работы с Интернетом Вещей, то есть с устройствами, которые взаимодействуют и управляют реальным миром, со временем, он стал полезен для целого ряда приложений. [11]

3.2.1 Ключевые идеи

Когда вы программируете с помощью Node-RED, вы создаете потоки, которые представляют собой сети легких компонентов, называемых узлами, которые обмениваются сообщениями по заранее определенным соединениям или «проводам». [11]

3.2.1.1 Потоки

Программы или потоки Node-RED представляют собой набор узлов, соединенных вместе для обмена сообщениями. По сути поток состоит из списка объектов JavaScript, которые описывают узлы и их конфигурации, а также список нижестоящих узлов, к которым они подключены. [11]

3.2.1.2 Сообщения

Сообщения, передаваемые между узлами в Node-RED, являются, по соглашению, объектами JavaScript, называемыми msg, состоящими из набора именованных свойств. Эти сообщения часто содержат свойство msg.payload, то есть полезную нагрузку сообщения. Узлы могут связывать другие свойства с сообщением, которое может использоваться для переноса другой информации на следующий узел в потоке. Когда это произойдет, эти дополнительные свойства будут задокументированы в документации узла, которая появляется в

панели сведений о узле, когда вы выбираете узел в рабочем пространстве Node-RED.[11]

Сообщения - это основная структура данных, используемая в Node-RED, и в большинстве случаев это единственные данные, с которыми узел должен работать, когда он активирован.

Это одно из ключевых преимуществ языка, основанного на потоках. Поскольку узлы автономны и обычно взаимодействуют только с другими узлами с использованием сообщений, вы можете быть уверены, что у них нет непредвиденных побочных эффектов, и поэтому их можно безопасно повторно использовать при создании новых потоков. [11]

3.2.1.3 Узлы

Узлы являются основным строительным блоком потоков Node-RED. Когда поток работает, сообщения генерируются, потребляются и обрабатываются узлами. Узлы состоят из кода, который выполняется в службе Node-RED (javascript.js-файл) и файла HTML, состоящего из описания узла, так что он отображается в области узла с категорией, цветом, именем и значком, код для настройки узла и текст справки. Узлы могут иметь не более одного входа, и ноль или более выходов. [11]

Во время процесса инициализации узел загружается в службу Node-RED. Когда браузер обращается к редактору Node-RED, код для установленных узлов загружается на страницу редактора (рис.2).

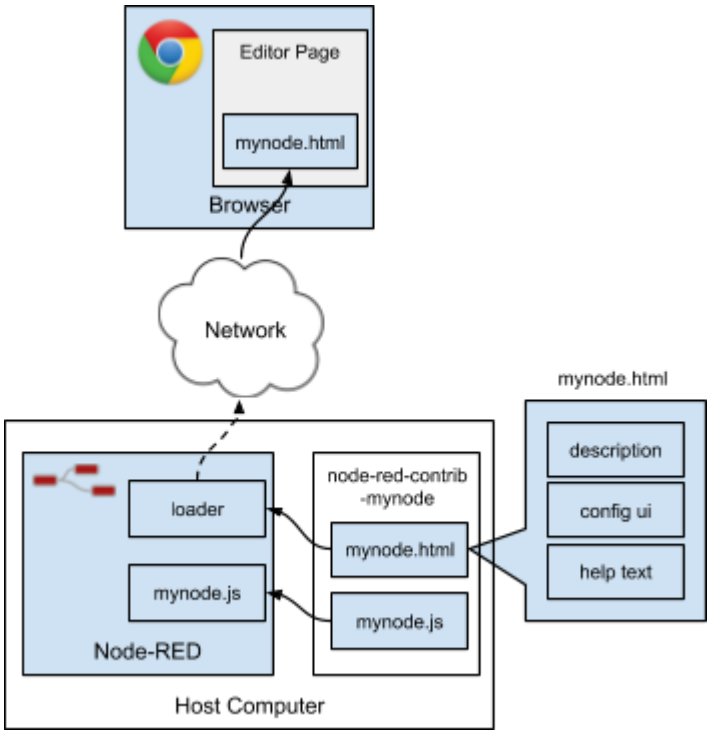

Рис.2. Схема работы NODE-RED

Существует три типа основных узлов:

1) Входные узлы – они генерируют сообщения для нижестоящих узлов;
2) Выходные узлы - потребляют сообщения, например, для отправки данных во внешнюю службу или для вывода на устройство и могут генерировать ответные сообщения;
3) Узлы обработки - узлы, которые каким-то образом обрабатывают данные.

В дополнение к этим основным типам существует еще две категории:

1) Учетные узлы - это узлы, которые содержат учетные данные, используемые одним или несколькими узлами для подключения к внешней системе или сервису, таким как платформа Sense Tecnic, MQTT или Pushbullet. Они отображаются только на панели учетных данных, а не на главной панели, и создаются по мере необходимости, когда вы настраиваете узел, требующий учетных данных, таких как ключ API или имя и пароль. После создания узел учетных данных может быть повторно использован другими узлами того же типа для

подключения к аналогичным протоколам или службам. Даже когда все узлы, использующие эти учетные данные, будут удалены из вашего потока, узел учетных данных останется, поэтому рекомендуется удалить неиспользуемые учетные данные, когда они больше не нужны;

2) Пользовательские узлы - программируемые узлы, такие как узлы функций или подпотоки, являются созданными вами узлами для выполнения некоторых пользовательских операций или повторного использования сегментов потока в других потоках. [11]

3.2.1.4 «Провода»

Провода определяют соединения между входными и выходными конечными точками узла в потоке. Они (обычно) соединяют выходные конечные точки узлов с входами последующих узлов, указывая, что сообщения, сгенерированные одним узлом, должны обрабатываться подключенным узлом. С помощью проводов можно подключить более одного узла к конечной точке. Когда несколько узлов подключены к конечной точке вывода, сообщения отправляются на каждый подключенный узел поочередно в том порядке, в котором они были подключены к выходу. Когда к входной конечной точке подключено более одного узла, сообщения с любого из этих узлов будут обрабатываться подключенным узлом, когда они прибудут. Также возможно подключить узлы, находящиеся далее по потоку, к восходящим узлам для формирования циклов. [11]

3.2.1.5 Контекст

Как правило, сообщения - единственный способ получить данные в узлах и из них. Хотя это, как правило, верно, есть одно исключение из этого правила, которое доступно для функционирования узлов. Функциональные узлы имеют

доступ к специальному объекту, называемому контекстом, который используется для хранения данных в памяти для передачи от одного сообщения к другому. Это важно для узлов, которым необходимо поддерживать индекс или подсчитывать или суммировать данные в сообщениях. В дополнение к этому локальному контексту доступен глобальный контекст context.global для обмена данными между всеми функциональными узлами потока. [11]

3.2.1.6 Функциональные узлы

Функциональный узел является «швейцарским армейским ножом» узлов, которые вы можете использовать, когда нет существующего узла, посвященного вашей задаче. Это отлично подходит для специализированной обработки данных или форматирования, например. Как следует из названия, функциональный узел предоставляет одну функцию JavaScript. Используя функциональный узел, вы можете написать свой собственный JavaScript-код, который выполняется против переданных сообщений, и возвращает ноль или более сообщений в нижестоящие узлы для обработки. Чтобы написать функциональные узлы, используется встроенный редактор кода (рис.3). [11]

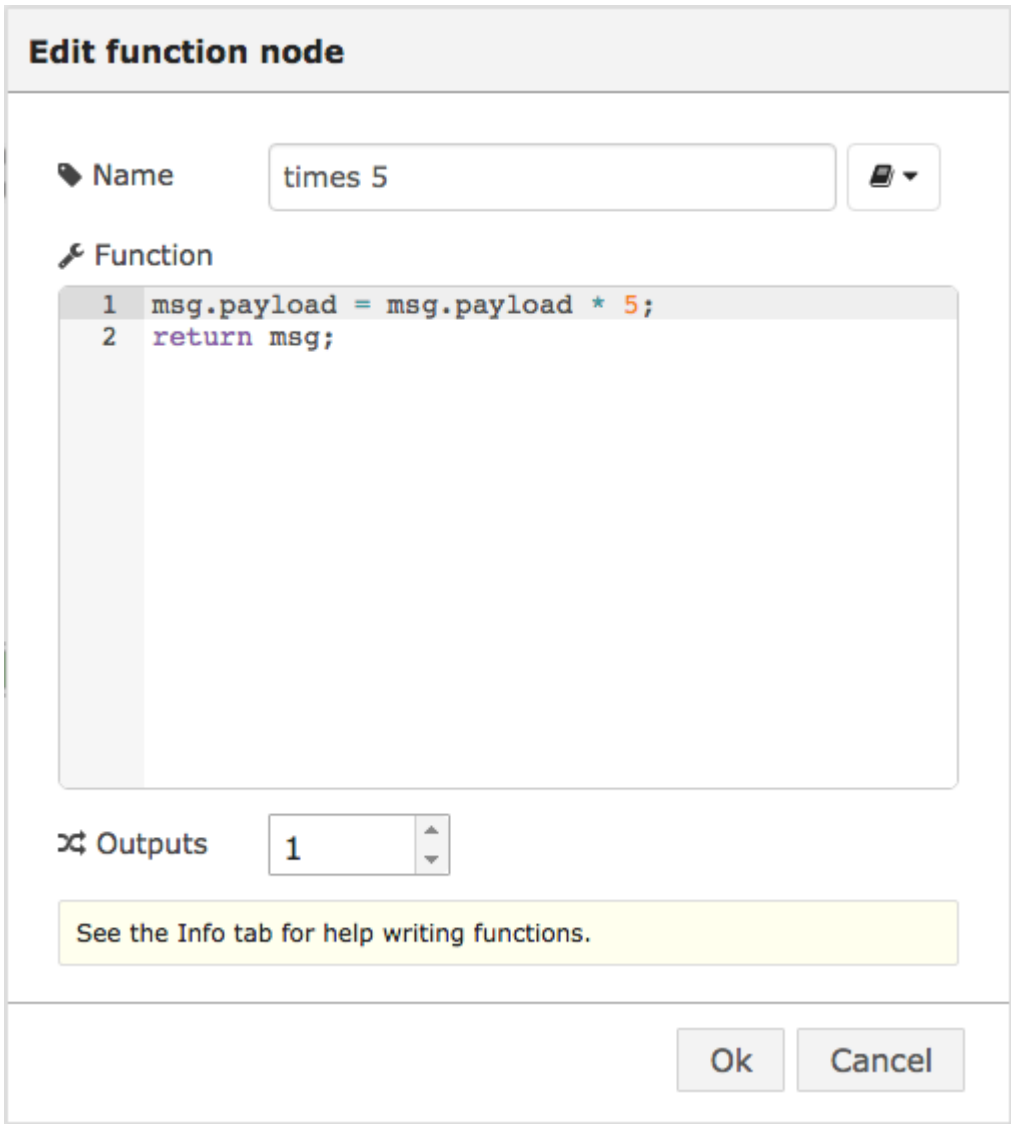

Рис.3. Пример функционального узла

Глава 4. Применение концепции Промышленного Интернета вещей (IIoT) для построения системы телемеханики и телеметрии распределенных промышленных объектов.

С помощью Промышленного Интернета вещей был разработан и создан комплекс удаленного мониторинга и управления технологическим оборудованием на примере сервисного обслуживания и эксплуатации холодильных агрегатов. Комплекс программ организован в виде Web-приложения, состоящего из 2 частей: серверной и клиентской, описание которых будет в последующих подпунктах данной главы.

4.1 Структурная схема проекта

На представленной ниже структурной схеме изображен весь путь получения и обработки данных (рис.4):

1) Передача значений с удаленного объекта мониторинга на контроллер, а именно температуры камеры, испарителя холодильной камеры, а также данных об ее состоянии;
2) Получение данных с контроллера при помощи облачного сервиса CloudMQTT через протокол mqtt;
3) Сохранение оперативных данных, полученных с облачного сервиса на стороне сервера NODE-RED;
4) Организация аварийного оповещения на сервере в виде отправки e-mail сообщений;
5) Отображение в Web-приложении данных реального времени в текстовом и графическом виде, а также расположение кнопок для управления состояниями агрегата;
6) Вывод в Web-приложении исторических данных в текстовом и графическом виде;

7) Вывод в Web-приложении исторических данных об аварийных режимах.

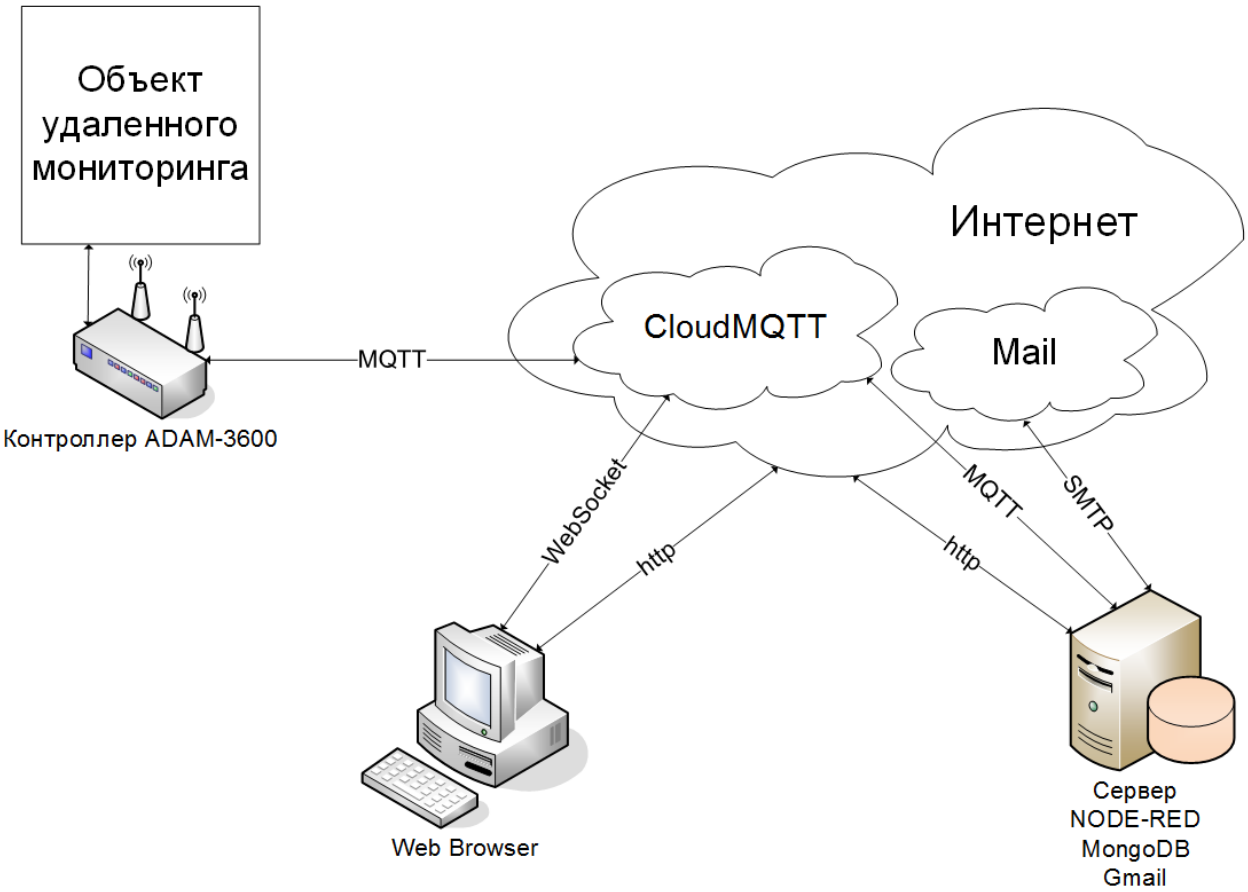

Рис.4. Структурная схема проекта

4.2 Имитационная модель технологического объекта

Для отладки приложения в качестве объекта мониторинга использовалась имитационная модель холодильной камеры на основе электротепловой аналогии. Модель была создана в рамках курсовой работы студентом Марченко Н.А.

Электротепловая аналогия – это метод расчёта тепловых систем, сводящий их расчёт к расчёту эквивалентных линейных электрических схем.

Для дальнейшего построения имитационной модели холодильной камеры необходимо создать ее математическую модель, то есть записать дифференциальные уравнения, описывающие изменения параметров камеры во времени.

Можно произвести электротепловую аналогию для данного проекта (рис.5) и составить математическую модель холодильной камеры, то есть составить дифференциальные уравнения.

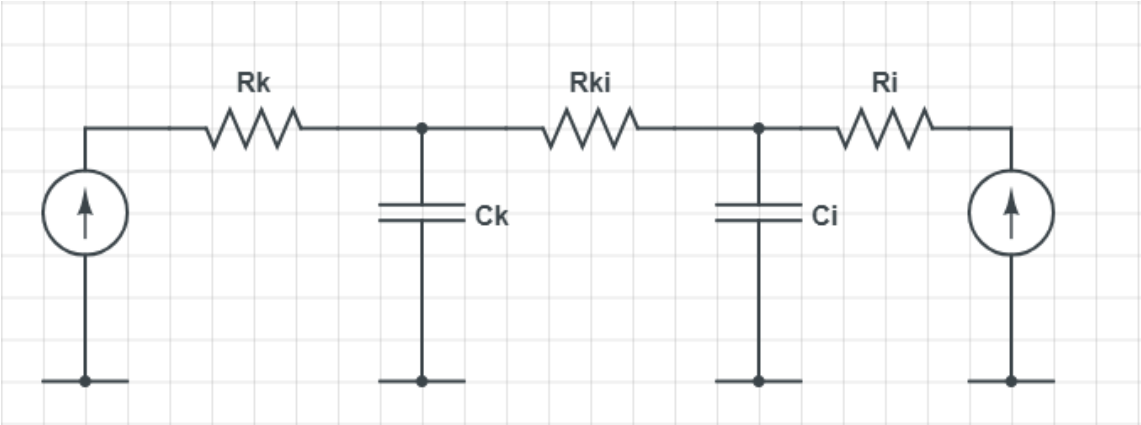

Рис.5. Электротепловая аналогия

Обозначения Ck и Ci – это теплоемкость камеры и теплоемкость испарителя соответственно.

Параметры, которые будут полностью описывать нашу систему - это температура камеры и температура испарителя, то есть напряжения на Rk и Ri соответственно. Ниже составлена система, состоящая из дифференциальных уравнений (1) и (2):

$$\frac{\mathrm{dTk}}{\mathrm{dt}} = \frac{\mathrm{Yk}}{\mathrm{Ck}}(\mathrm{T0} - \mathrm{Tk}) + \frac{\mathrm{Yki}}{\mathrm{Ck}}(\mathrm{Ti} - \mathrm{Tk}); (1)$$

$$\frac{\mathrm{dTi}}{\mathrm{dt}} = \frac{\mathrm{Yki}}{\mathrm{Ci}}(\mathrm{Tk} - \mathrm{Ti}) + \frac{\mathrm{Yi}}{\mathrm{Ci}}(\mathrm{Tisp} - \mathrm{Ti}); (2)$$

В уравнениях сопротивления R заменены на проводимости Y. Данная система уравнений в дальнейшем решалась непрерывно в реальном масштабе времени на контроллере ADAM-3600 методами Эйлера и Рунге-Кутты четвертого порядка.

4.3 Серверная часть Web-приложения

Вся серверная часть Web-приложения создана при помощи инструмента NODE-RED. В ней организовано:

- Получение оперативных данных от CloudMQTT и их последующая архивация СУБД MongoDB;
- Обработка запросов на вывод исторических данных из БД;
- WEB - сервер, предназначенный для загрузки браузером HTML, CSS и Java Script файлов;
- Формирование аварийных оповещений.

4.3.1 Начало работы с NODE-RED

Как говорилось ранее, NODE-RED это мощный инструмент для работы с Интернетом Вещей, где особое внимание уделяется упрощению блоков кода для выполнения задач.

Рассмотрим этот инструмент и его основной экран подробнее (рис.6).

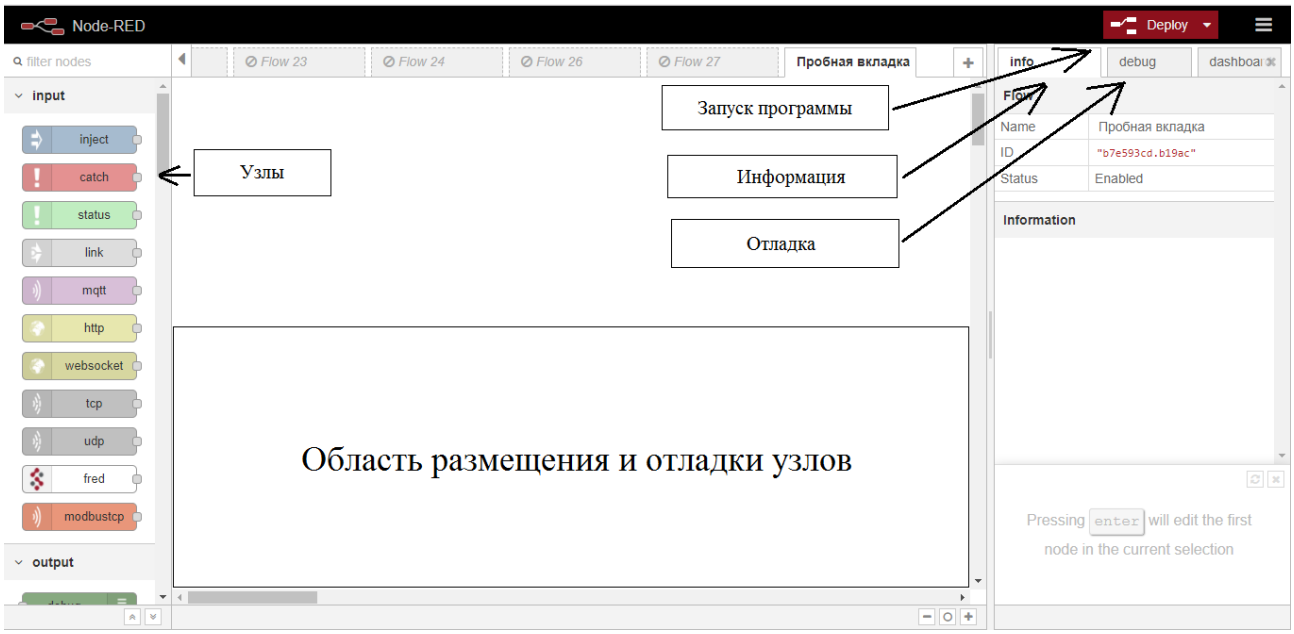

Рис.6. Основной интерфейс NODE-RED

1) Основным преимуществом NODE-RED являются «узлы». Это некие блоки, которые запрограммированы на языке JavaScript для упрощения написания той или иной задачи. В стандартный набор входит очень много полезных узлов, которых будет достаточно для написания первой и простой программы. Если же возможностей этих узлов будет недостаточно, то существуют специальные узлы, в которых можно писать свой код. Так же мы можем дополнительно установить нужные нам узлы. Подробно способ установки описан в пункте 4.3.2;
2) Размещение узлов происходит на большом выделенном пространстве по середине NODE-RED;
3) Во вкладке «info» можно посмотреть информацию о том или ином узле, просто нажав на него;
4) Во вкладке «debug» можно посмотреть информацию при отладке программы. Если при ее выполнении возникнут какие-либо ошибки, то там будет написано что именно не работает и почему показывает ошибку;
5) При нажатии клавиши «Deploy» программа, составленная из узлов, начнет свою работу.

4.3.2 Дополнительная установка узлов

1) Нажать на иконку выплывающего меню справа от кнопки «Deploy» и после нажать на Manage palette (рис.7);

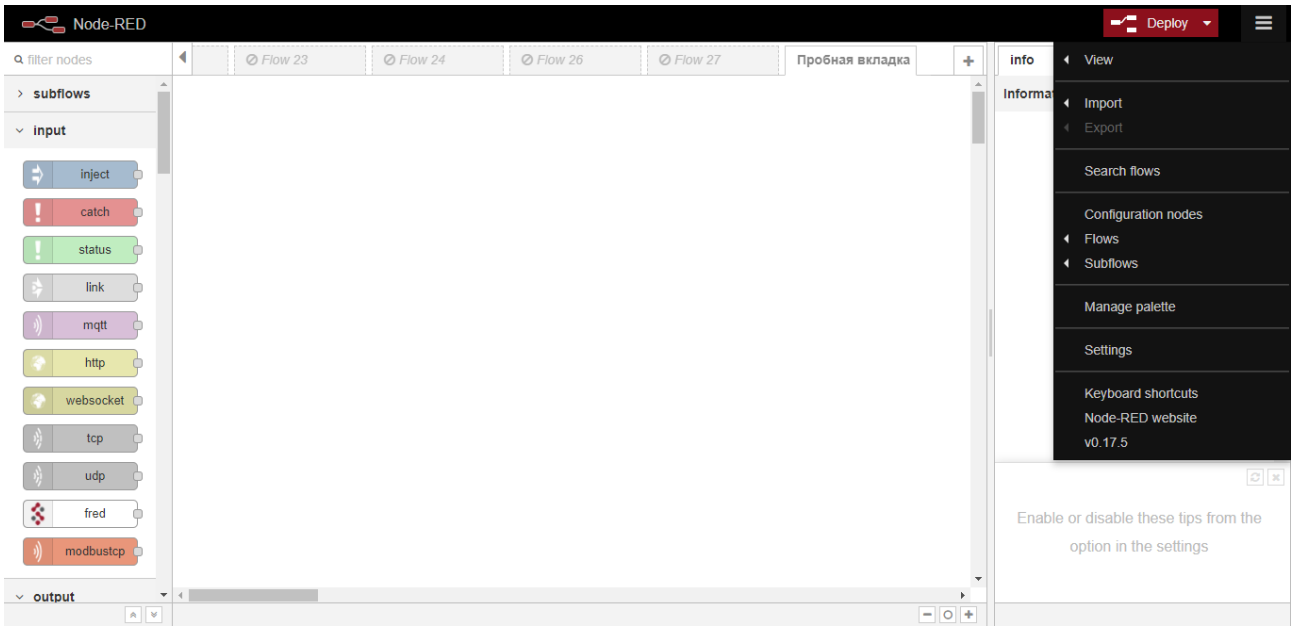

Рис.7. Пример установки дополнительного узла mongodb (1)

2) Нажать на вкладку Install и в окне поиска написать название нужного узла (в данном примере mongodb) (рис.8);
3) Далее выбрать нужный узел и нажать на кнопку install для установки узла (рис.8);
4) После установки узла перезапустить NODE-RED.

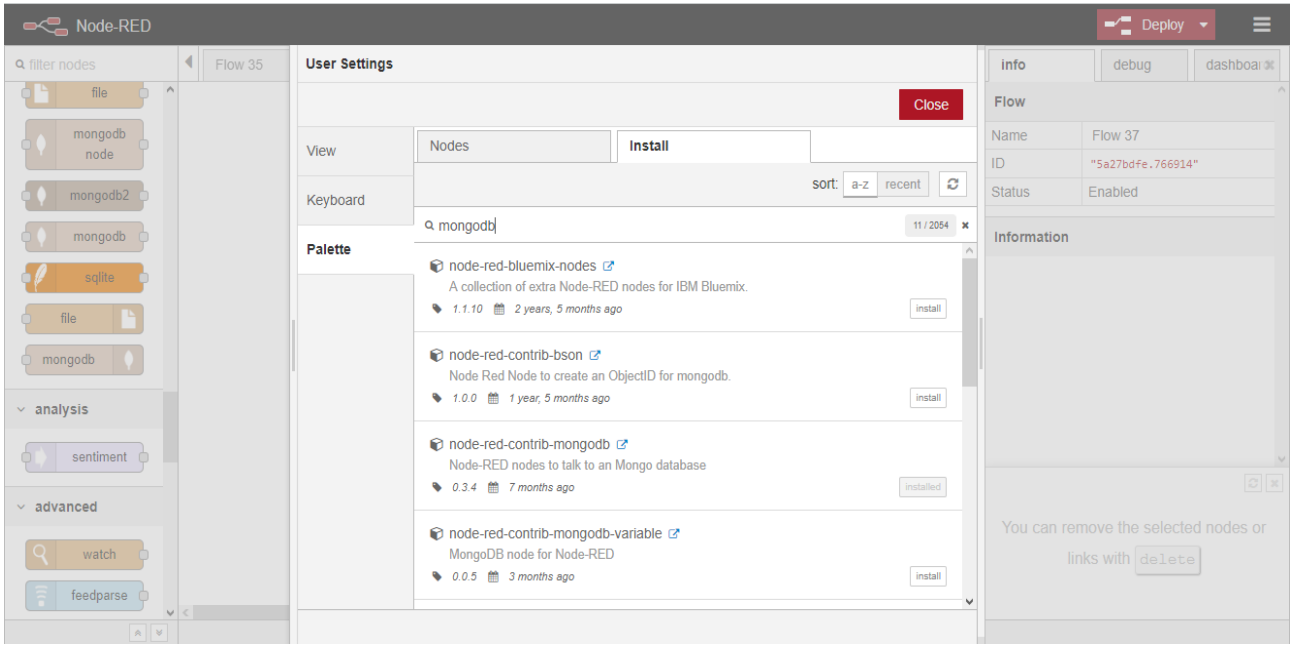

Рис.8. Пример установки дополнительного узла mongodb (2)

4.3.3 Архивация данных

Первой функциональной подсистемой является получение и архивация оперативных данных. Осуществляется это с использованием облачного сервиса CloudMQTT при помощи протокола mqtt и базой данных MongoDB.

Данная подсистема состоит из 4 узлов (рис.9).

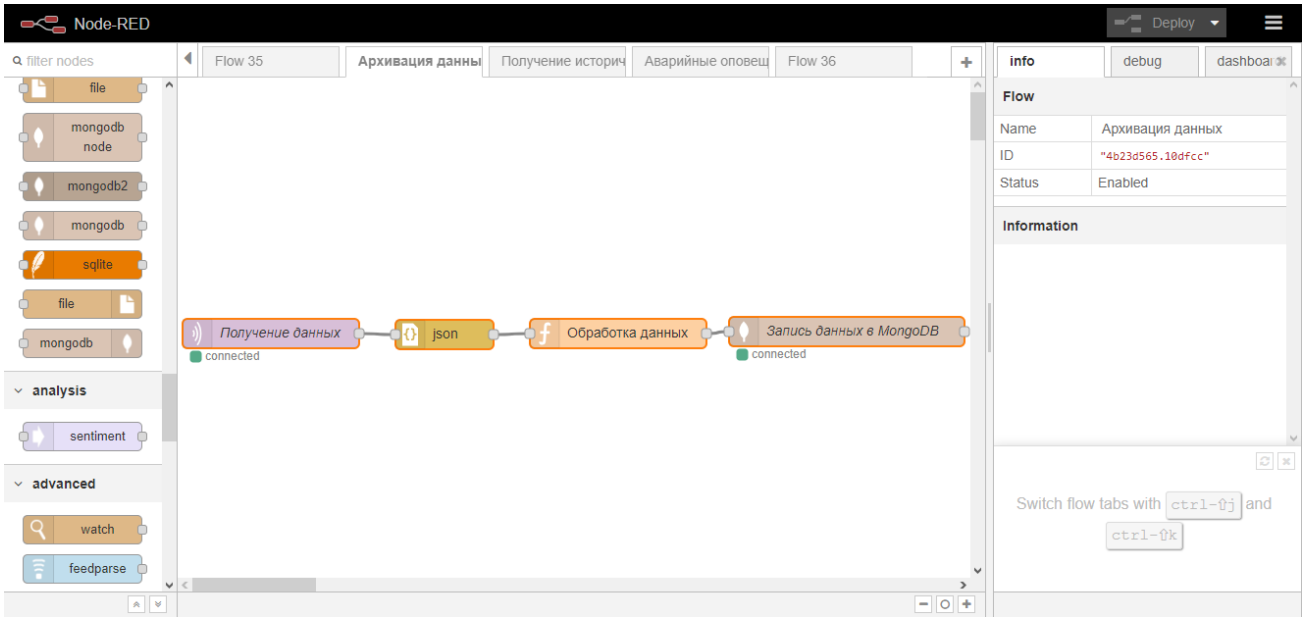

Рис.9. Программа для архивации оперативных данных

- Получение данных осуществляется с использованием облачного сервиса CloudMQTT при помощи протокола mqtt. Поля в узле заполнены так, как это было установлено на контроллере, то есть учетные записи и топики, использованные при передаче данных, должны совпадать;
- Далее через узел JSON мы осуществляем перевод строки, которую получили с сервиса, в объект и совершаем с ним нужные нам операции;
- В функциональном узле отбираем данные из общего массива и формируем новый объект, который в дальнейшем будет записываться в базу данных MongoDB.

4.3.4 Получение исторических данных

Второй функциональной подсистемой является получение исторических данных из базы по запросу клиентского приложения (рис.10). Данная подсистема осуществляет вывод исторических данных и аварийных событий, случившихся когда-либо с агрегатом.

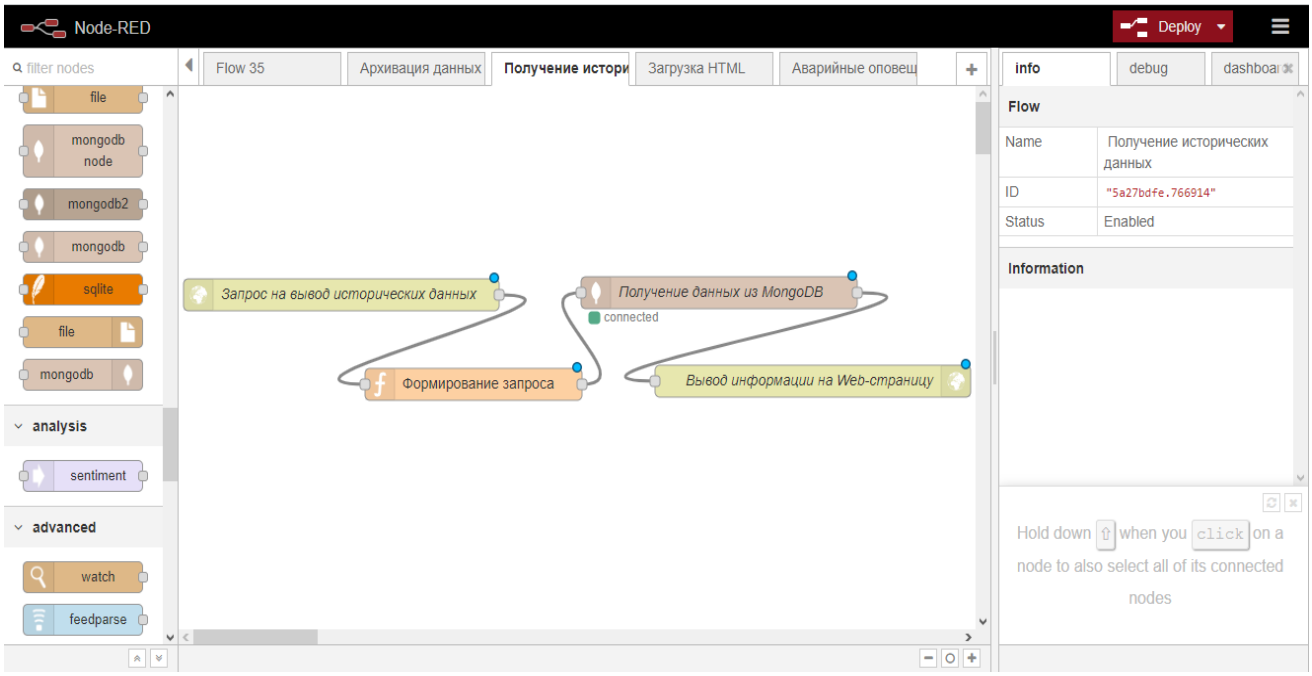

Рис.10. Программа для вывода исторических данных

- На Web-странице клиентской части приложения создается запрос, а именно, вводится дата и время, в пределах которых мы хотим получить данные из БД;
- Далее в функциональном узле идет обработка полученного запроса, то есть формируется временной интервал, который мы посылаем в БД, где составляется выборка из данных, которые соответствуют введенному запросу;
- Последний узел принимает отобранные данные из всей базы и публикует их на Web-странице.

Нужно отметить то, что в клиентской части приложения запрос осуществляется при помощи объекта XMLHttpRequest. Это позволяет из

JavaScript делать HTTP-запросы к серверу без перезагрузки страницы. Но для этого HTML страница должна иметь тот же IP-адрес и порт, по которому осуществляется запрос. Следовательно, все HTML страницы мы должны подгружать через сервер, чтобы не возникло проблем с кросс-доменностью (рис.11).

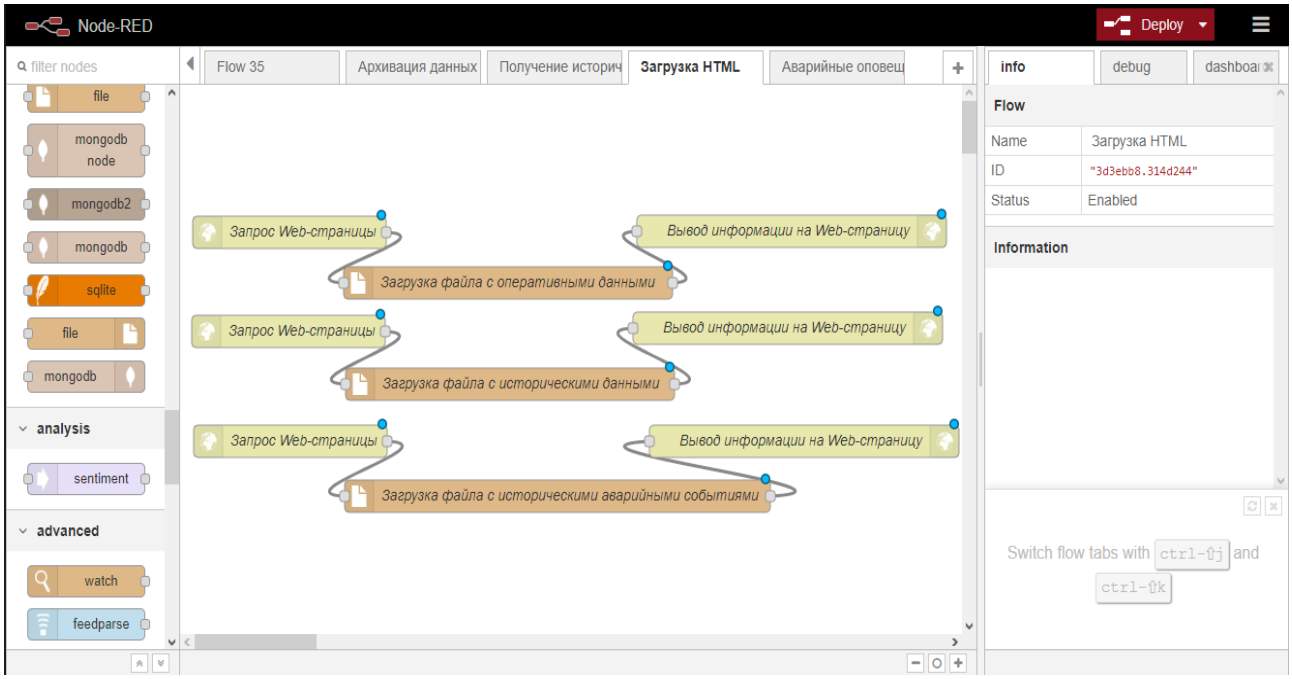

Рис.11. Загрузка HTML-страниц через сервер

4.3.5 Аварийные оповещения

Третьей функциональной подсистемой является осуществление аварийных оповещений (рис.12). Эта программа позволяет отправлять письма на электронную почту, где будет указано что именно случилось с агрегатом.

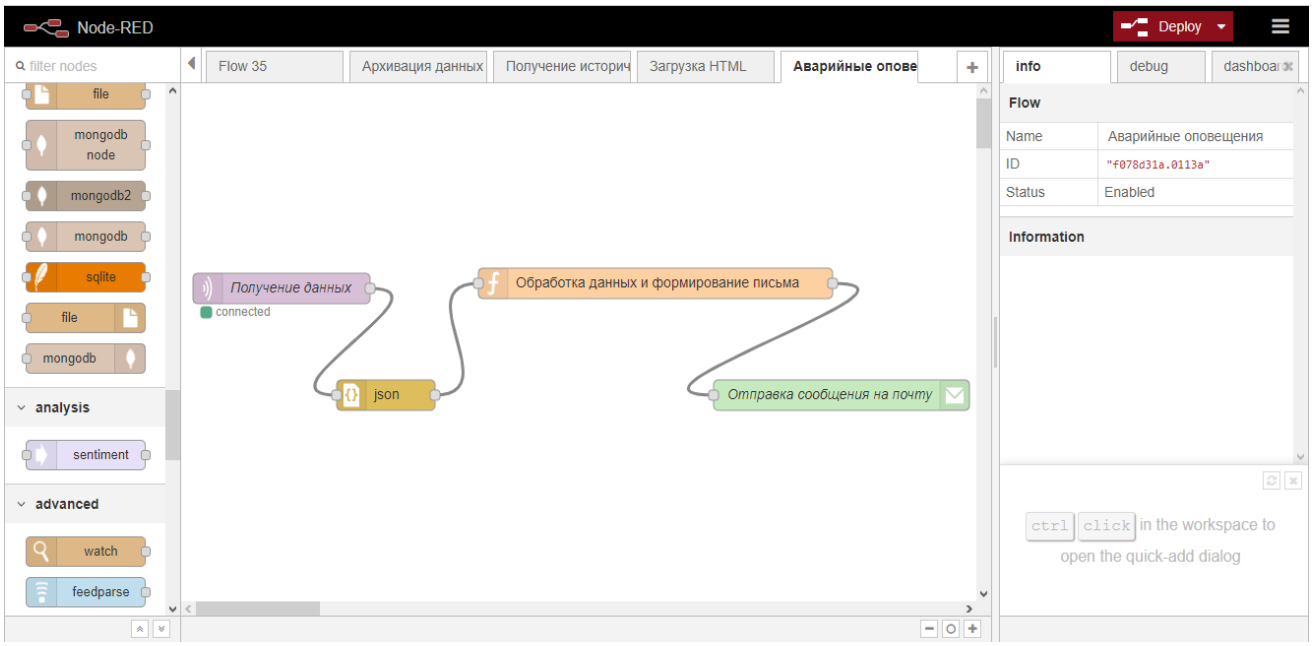

Рис.12. Программа для аварийных оповещений

- Получение данных осуществляется идентичным способом, как и при архивации оперативных данных, с использованием облачного сервиса CloudMQTT при помощи протокола mqtt;
- После через узел JSON мы осуществляем перевод строки, которую получили с сервиса, в объект и совершаем с ним нужные нам операции;
- Далее в функциональном узле на языке JavaScript запрограммированы некие проверки данных, которые сравнивают определенные переменные. Состояния об ошибках передаются в виде логических «0» и «1». То есть, если соответствующая переменная равна логической единице, то это говорит о том, что агрегат работает не в штатном режиме, и в соответствии с этой переменной формируется сообщение, после чего через специальный узел отправляется на почту, где указана точная ошибка и ее параметры.

4.4 Клиентская часть Web-приложения

Вся клиентская часть Web-приложения, организованного в виде HTML страниц, состоит из трех функциональных подсистем:

1) Управление и мониторинг;
2) Вывод исторических данных;
3) Вывод исторических аварийных событий.

4.4.1 Управление и мониторинг

Данная страница разделена на 3 блока (рис.13):

1) Мониторинг наличия связи с облачным сервисом CloudMQTT и контроллером, а также управление режимами работы холодильного агрегата;
2) Техническая схема работы агрегата;
3) Основной блок, предназначенный для наблюдения за состоянием камеры.

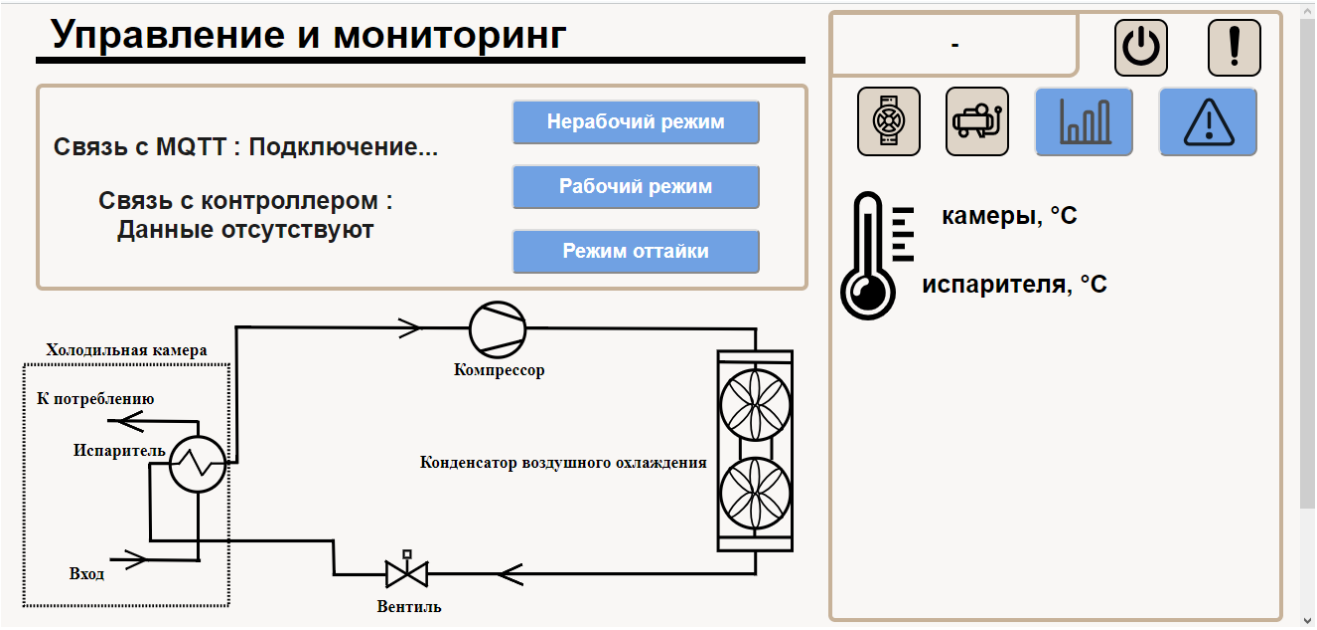

Рис.13. Общий вид страницы с управлением и мониторингом

- При первоначальной загрузке страницы, программа автоматически начинает подключение к облачному сервису CloudMQTT. Как только подключение завершится, в окне появится соответствующая надпись «Подключено». Далее для выбора режима работы агрегата необходимо нажать на соответствующую кнопку, то есть «Нерабочий режим», «Рабочий режим» или же «Режим оттайки». При нажатии на кнопку начинает работать функция, созданная на JavaScript, которая отправляет необходимые данные для включения определенного режима через облачный сервис. Если режим включается без каких-либо либо ошибок и сбоев, то соответствующая кнопка загорится зеленым цветом, иначе цвет кнопки будет красным (рис.14).

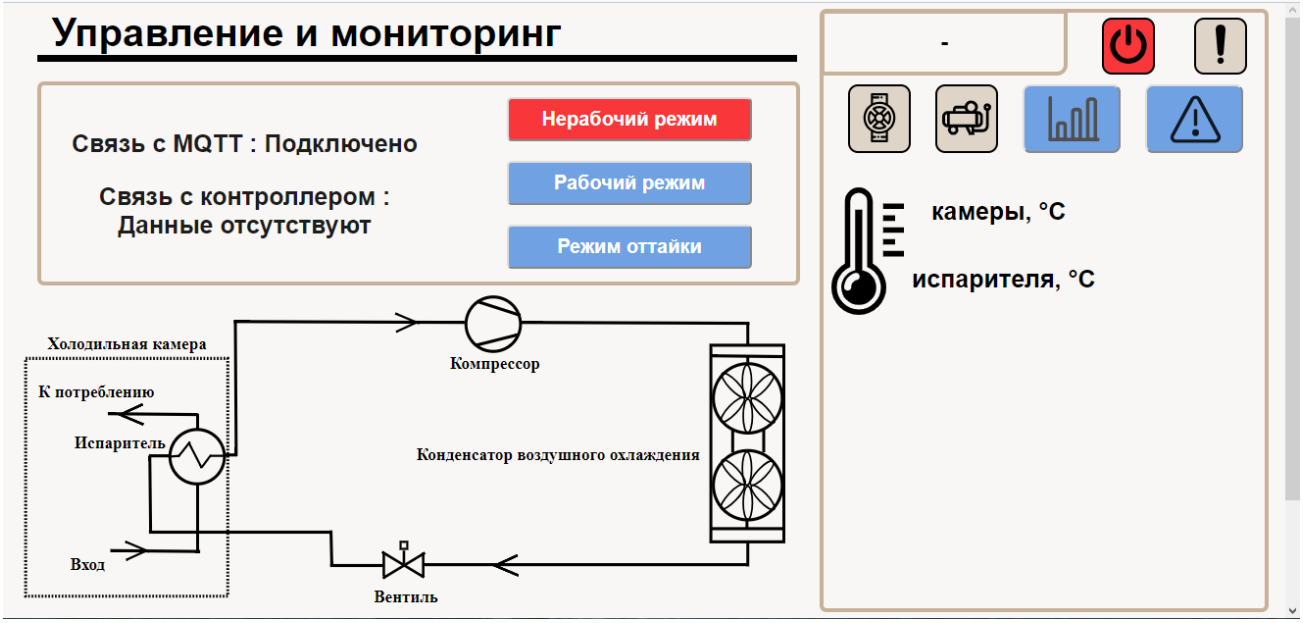

Рис.14. Пример включения режима

- При включении определенного режима активируется окно для мониторинга. В нем расположены индикаторы: состояние работы режима, индикация аварийных состояний, отображение состояния вентиля и компрессора соответственно. Также в данном окне располагается отображение режима работы, температура камеры и испарителя, кнопки для перехода между страницами. Более того, во второй половине окна находится график оперативных данных.

- Ниже представлен пример работы программы (рис.15). На нем мы можем наблюдать то, что у нас присутствует связь с облачным сервисом и непосредственно с самим контроллером. То есть, если данные с контроллера не поступают в течении 20 секунд, именно такой повтор установлен у функции в программе, то мы увидим сообщение о том, что связи с контроллером нет. Суть программы заключается в том, что при получении данных с контроллера записывается время, когда именно они поступили с облачного сервиса. Далее при начале работы функции данное время присваивается еще одной переменной. После 20 секунд функция повторяется автоматически и, если две переменные совпадают, то программа понимает, что данные не поступают и сообщает о том, что связь с контроллером отсутствует. Если же в период между повтором функции данные с контроллера все же поступали, то переменные с записанным временем будут отличаться друг от друга, тем самым показывая, что связь с контроллером присутствует. Также невозможно не заметить, что кнопка включения режима и индикатор его работы горит зеленым цветом. Это говорит о том, что камера находится в рабочем режиме. Индикаторы состояния вентиля и компрессора так же горят зеленым цветом и обозначают то, что вентиль находится в открытом состоянии, а компрессор включен. Технология их работы очень проста. При получении данных программа «разбирает» некий массив, откуда выделяет сведения о состояниях вентиля и компрессора, которые представлены в виде логических «0» и «1». Соответственно, при «0» программа установит CSS стиль в виде красного цвета подсветки, а при «1» зеленого. Также ниже расположено отображение температуры камеры и испарителя как в числовом, так и в графическом виде. Получение оперативных данных осуществляется при помощи библиотеки MQTT по протоколу

«WebSocket», предназначенного для общения между браузером и web-сервером в режиме реального времени.

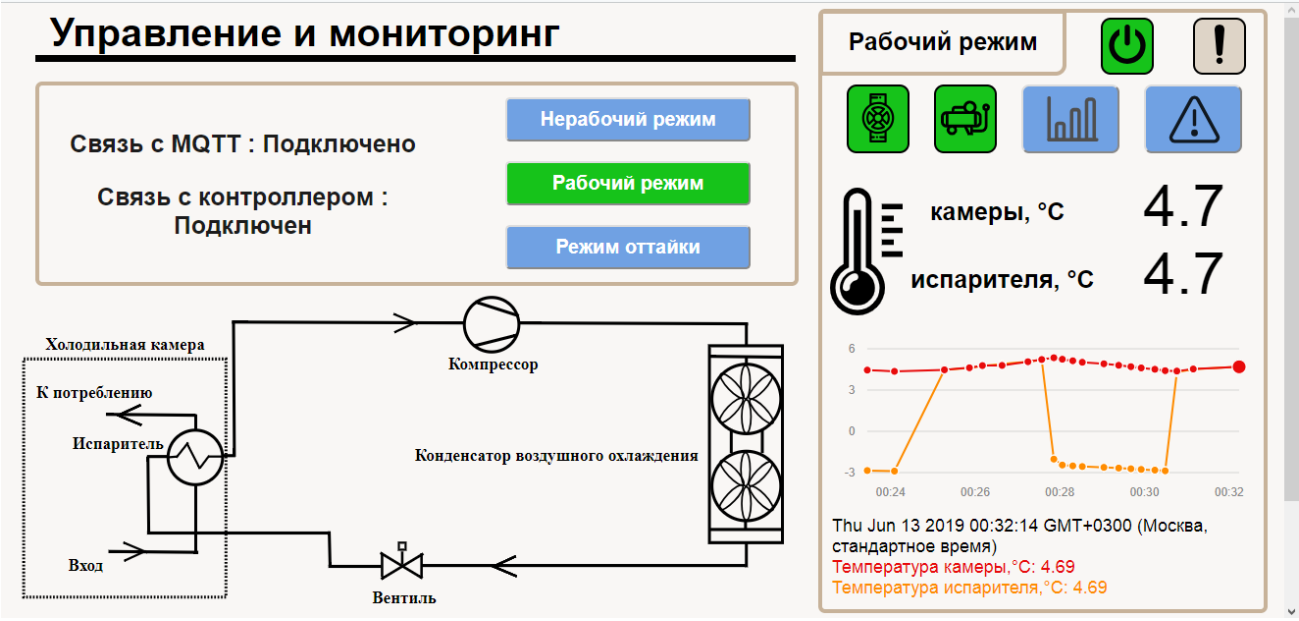

Рис.15. Пример работы программы

- В случае неполадок предусмотрена система аварийной индикации и оповещения. В данной программе отслеживаются интервалы температур. В случае выхода за их пределы, сразу же появится сообщение об аварийном режиме, то есть, при получении переменной, предназначенной для указания аварийных событий, со значением логической «1», специальная функция программы сформирует и выведет на экран сообщение с состоянием ошибки (рис.16).

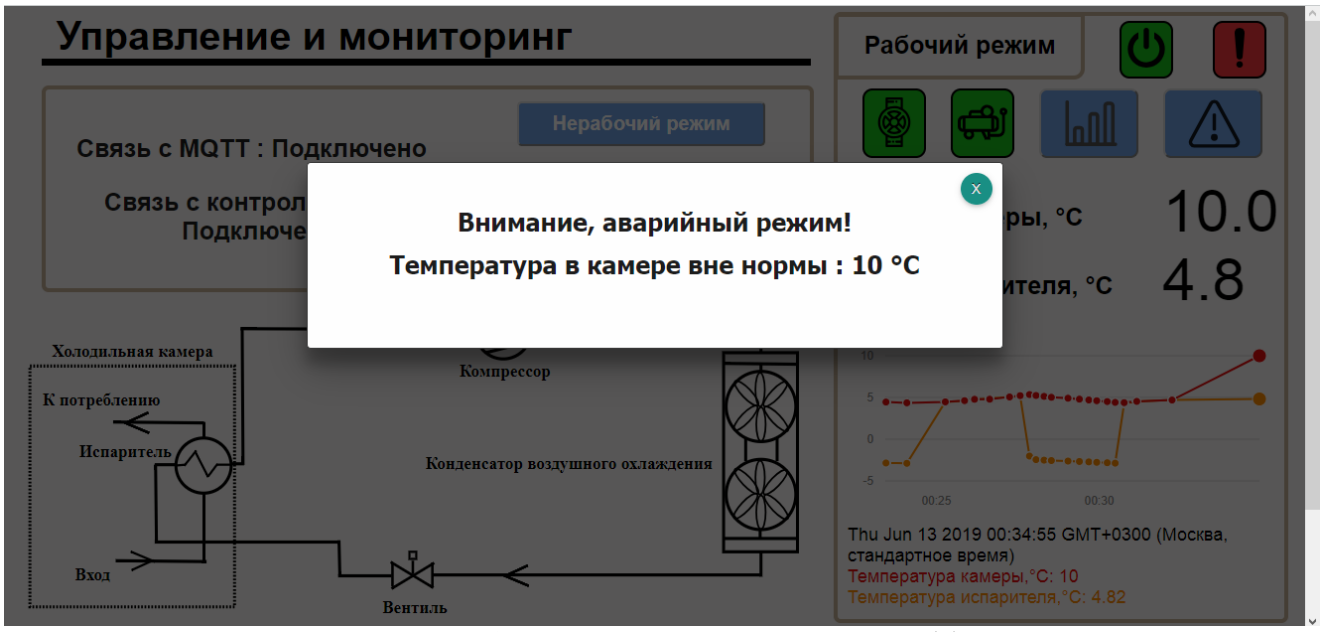

Рис.16. Пример аварийного режима (1)

- При закрытии основного окна с отображением аварийного режима, до устранения неполадок будет моргать индикатор «!», где при наведении на него так же можно прочесть о случившейся ошибке (рис.17).

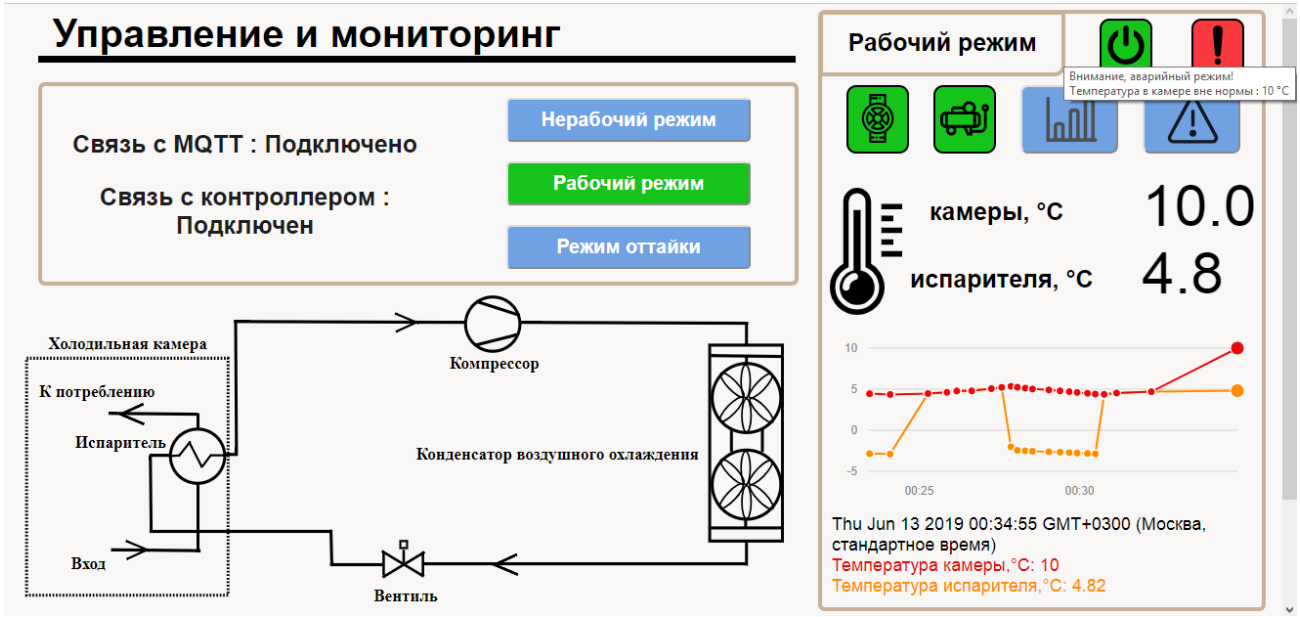

Рис.17. Пример аварийного режима (2)

- Также, как уже говорилось ранее, на серверной части установлена система аварийного оповещения при помощи E-mail сообщений, то есть при появлении аварийного режима на электронную почту сразу

же будет отправлено сообщение с указанием случившейся ошибки (рис.18).

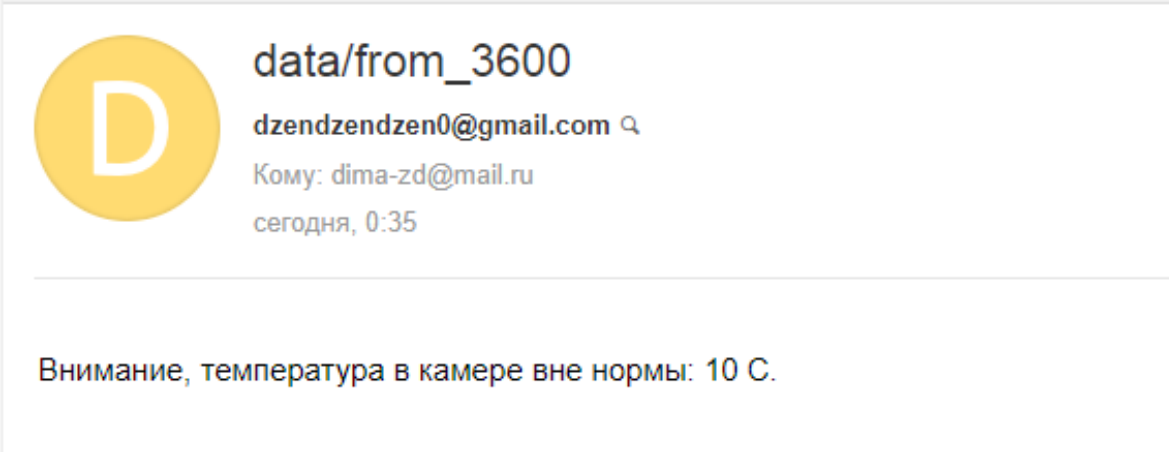

Рис.18. Пример аварийного оповещения

4.4.2 Вывод исторических данных

Для вывода информации необходимо ввести дату, чтобы определить временной интервал, в пределах которого мы хотим осуществить вывод данных (рис.19).

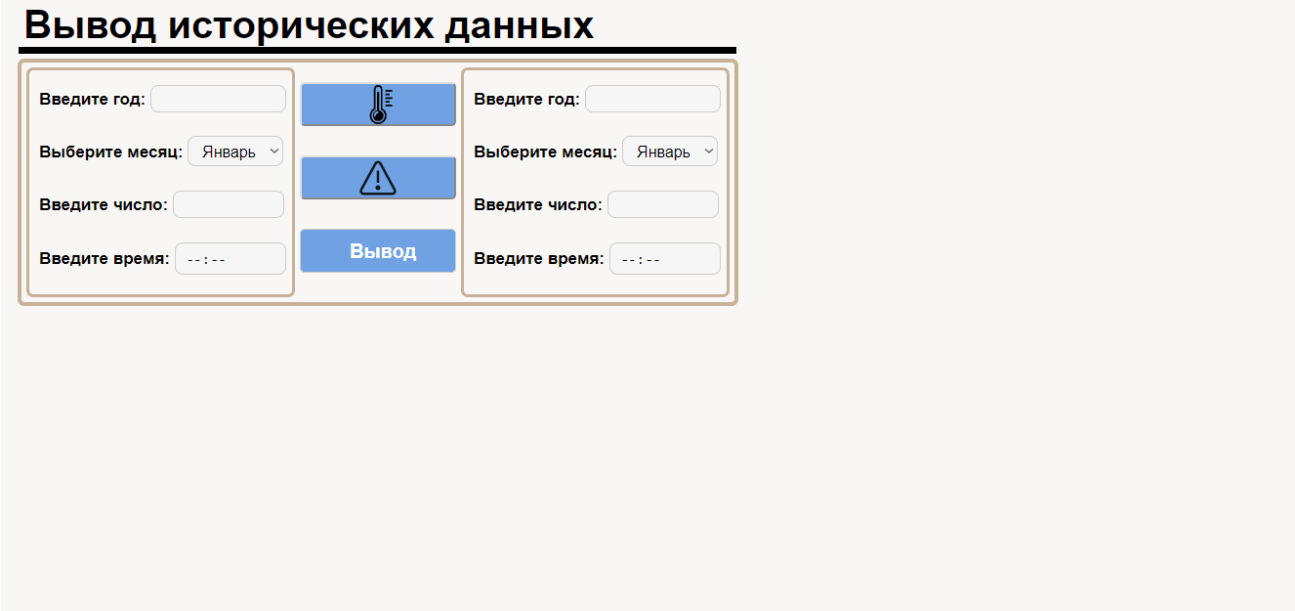

Рис.19. Общий вид страницы с выводом исторических данных

- В левой части окна поочередно вводятся данные для определения нижней границы времени. В правой же части осуществляется ввод для

определения верхней границы времени. По середине располагаются кнопки для перехода на страницу с мониторингом и управлением, а также на страницу с выводом исторических аварийных режимов соответственно. Кнопка «Вывод» предназначена для того, чтобы после ввода времени отправить данные на сервер, где они будут обработаны. Далее составляется определенная выборка из базы данных в пределах заданного временного интервала, которая состоит из температуры камеры и испарителя, а также даты и режима работы, который был задан в то или иное время. Далее она отправляется на Web-страницу, где отображается в численном, а также в графическом виде (рис.20).

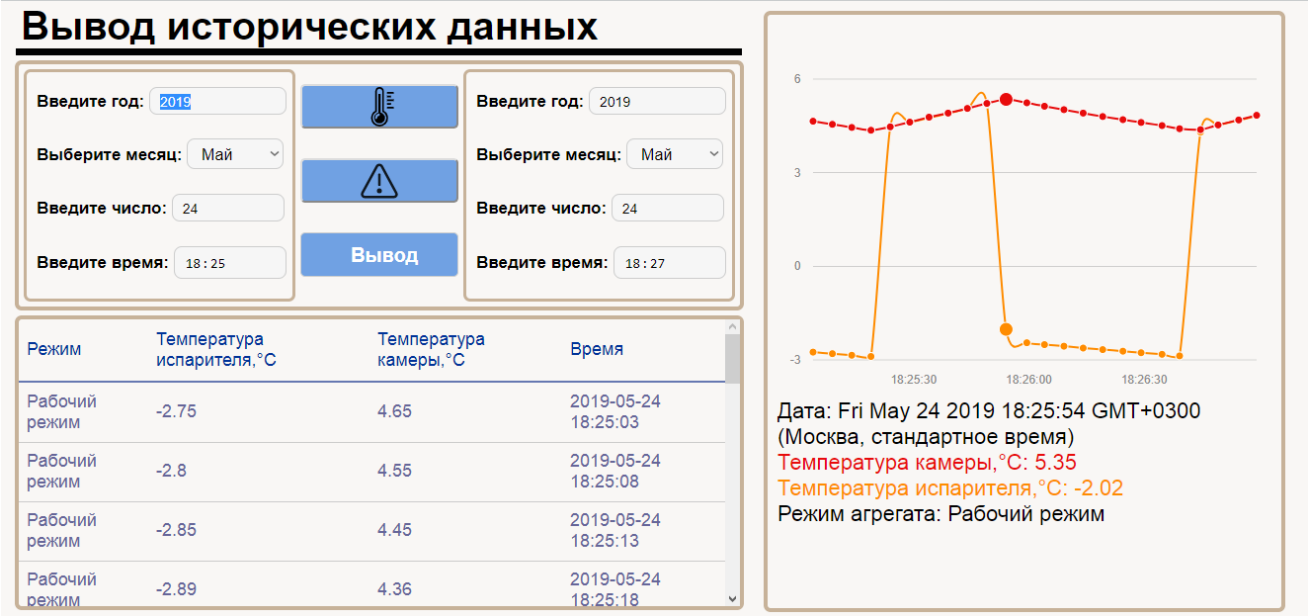

Рис.20. Пример вывода исторических данных

4.4.3 Вывод аварийных состояний

Аналогично предыдущему пункту, для вывода информации необходимо ввести дату, чтобы определить временной интервал, в пределах которого мы хотим просмотреть сообщения об ошибках (рис.21).

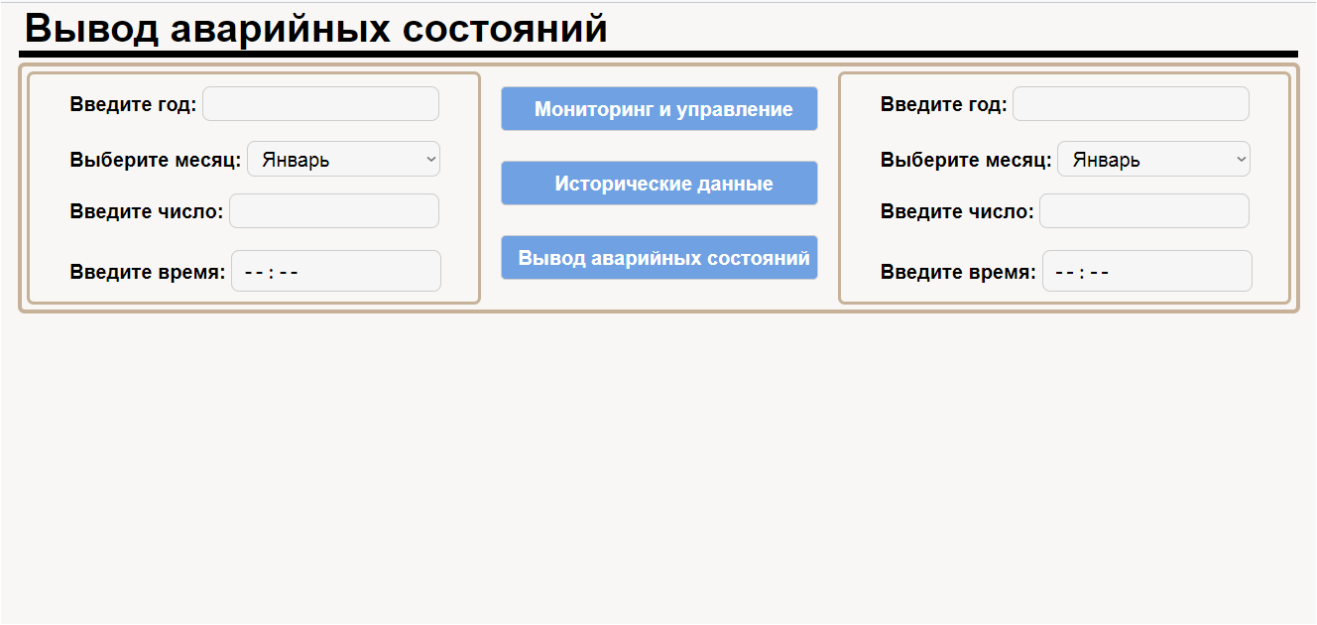

Рис.21. Общий вид страницы с выводом исторических аварийных состояний

- Для определения временного интервала необходимо осуществить ввод даты для определения его нижней и верхней границы. По середине так же располагаются кнопки для перехода между остальными страницами, а именно на страницу мониторинга и управления и на страницу для вывода исторических данных соответственно. В самом низу расположилась кнопка «Вывод аварийных состояний», предназначение которой заключается в том, чтобы после ввода времени отправить данные на сервер, где они будут обработаны. При получении всех необходимых параметров, сервер составит выборку из базы данных, после чего отправит ее на Web-страницу, где она будет «разобрана» и отображена в табличном виде (рис.22).

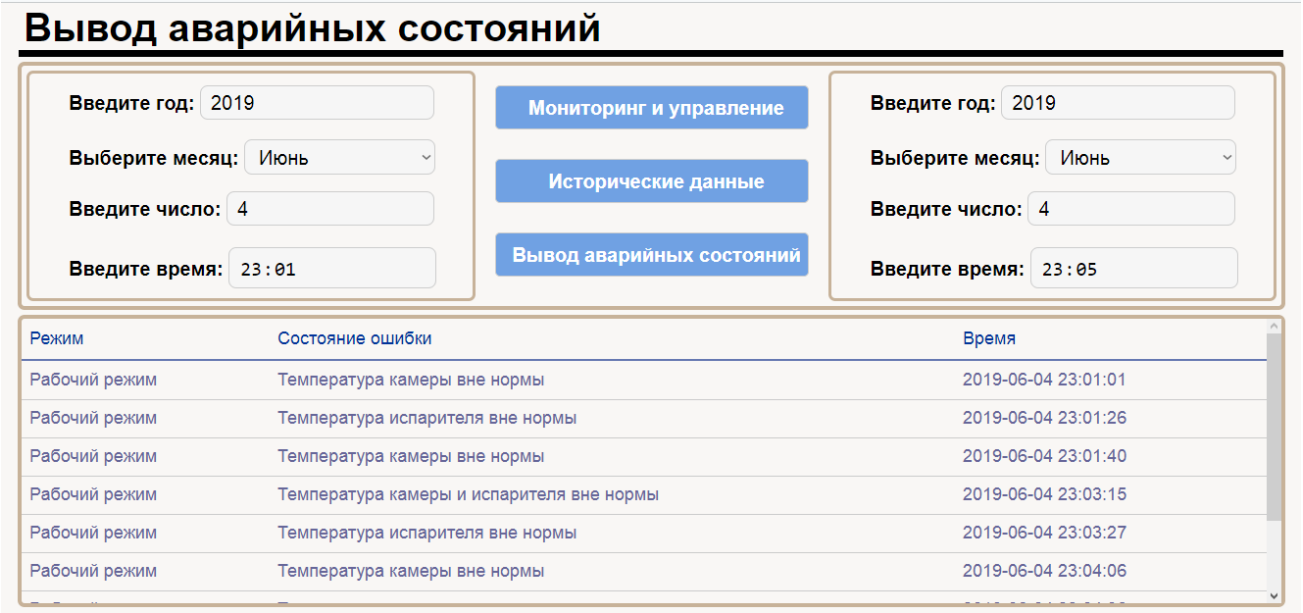

Рис.22. Пример вывода исторических аварийных состояний

Заключение

В результате проделанной работы было разработано Web-приложение, предназначенное для удаленного мониторинга и управления технологическим оборудованием с использованием современных технологий Промышленного Интернета Вещей. Данное приложение составлено из серверной и клиентской частей. Была реализована система для записи и вывода информации из базы данных, а также система аварийного оповещения. Для разработки Web-приложения были освоены и применены: инструмент NODE-RED, язык

программирования JavaScript, язык разметки HTML, каскадные таблицы стилей CSS. [12, 13, 14]

Комплексное тестирование приложения было проведено на имитационной модели холодильной камеры. По завершению тестирования, можем сделать заключение, что программа полностью отлажена и готова к работе.

Список литературы

1. Что такое IoT, или интернет вещей [Электронный ресурс]. – Режим доступа: https://coinspot.io/beginners/chto-takoe-iot-ili-internet-veshhej (дата обращения: 10.05.19)
2. Что такое Интернет вещей: существующие технологии интернета вещей [Электронный ресурс]. – Режим доступа:

https://strij.tech/publications/tehnologiya/chto-takoe-internet-veschey.html (дата обращения: 10.05.19)

3. Облачные сервисы - это что такое? [Электронный ресурс]. – Режим доступа: http://fb.ru/article/290383/oblachnyie-servisyi---eto-chto-takoe (дата обращения: 10.05.19)
4. SaaS - что это такое? Программное обеспечение SaaS [Электронный ресурс]. – Режим доступа: https://www.syl.ru/article/310019/saas---chto-eto-takoe-programmnoe-obespechenie-saas (дата обращения: 10.05.19)
5. Просто о корпоративном IaaS: что это, для кого, и как оплачивается [Электронный ресурс]. – Режим доступа: https://habr.com/company/it-grad/blog/257295 (дата обращения: 10.05.19)
6. Лабиринты PaaS. Платформы облачной разработки [Электронный ресурс]. – Режим доступа: http://www.iksmedia.ru/articles/5291675-Labirinty-PaaS-Platformy-oblachnoj.html (дата обращения: 10.05.19)
7. БД - это... Виды и свойства БД [Электронный ресурс]. – Режим доступа: http://fb.ru/article/234783/bd---eto-vidyi-i-svoystva-bd (дата обращения: 10.05.19)
8. Обзор реляционных систем управления базами данных [Электронный ресурс]. – Режим доступа: https://www.8host.com/blog/kratkij-obzor-relyacionnyx-sistem-upravleniya-bazami-dannyx (дата обращения: 10.05.19)
9. Введение в MongoDB [Электронный ресурс]. – Режим доступа: https://metanit.com/nosql/mongodb/1.1.php (дата обращения: 10.05.19)
10. Node.js для начинающих: описание, руководство, особенности [Электронный ресурс]. – Режим доступа: http://fb.ru/article/368698/node-js-dlya-nachinayuschih-opisanie-rukovodstvo-osobennosti (дата обращения: 10.05.19)
11. Руководство по программированию NODE-RED [Электронный ресурс]. – Режим доступа: http://noderedguide.com (дата обращения: 10.05.19)
12. Современный учебник Javascript [Электронный ресурс]. – Режим доступа:

https://learn.javascript.ru (дата обращения: 10.05.19)

13. Макфарланд Д. Новая большая книга CSS. - СПб.: Питер, 2016. - 720 с.
14. Мак-Дональд М. Создание Web-сайта. Недостающее руководство. - 3-е изд.: Пер. с англ. - СПб.: БХВ-Петербург, 2013. — 624 с.