МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий
Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практическим работам 9-12

по дисциплине

«Технологические основы Интернета вещей»

Выполнили: студенты группы ИВБО-02-19

К. Ю. Денисов
И. А. Кремнев
А. М. Сосунов
Д. Н. Федосеев

Принял: ассистент

Ю. А. Воронцов

Москва 2021

Содержание

1 Практическая работа №9 **3**
1.1 Создание виртуальных устройств в облаке 3
1.2 Отправка данных в облако 4

2 Дополнительное задание №9 **6**
2.1 Выбор облачного решения 6
2.2 Реализация отправки данных 6

3 Практическая работа №10 **7**
3.1 Реализация сценария управления вентилятором 8
3.2 Реализация сценария управления лампами 11

4 Дополнительное задание № 10 **14**
4.1 Требования к интерфейсу пользователя 14
4.2 Макеты интерфейса приложения 15

5 Практическая работа № 11 **18**
5.1 Обработка тревожных сигналов вентилятора 18
5.2 Обработка тревожных сигналов блока ламп 23

6 Дополнительное задание № 11 **27**
6.1 Описание технологического стека проекта 27
6.2 Реализация пользовательского интерфейса 28

7 Практическая работа №12 **31**
7.1 Отправка Email сообщений. Умный вентилятор 31
7.2 Отправка Email сообщений о работе блока умных ламп 34

8 Дополнительное задание № 12 **36**
8.1 Тестирование разрабатываемого приложения 36
8.2 Описание процесса развертывания приложения 36

Выводы о проделанной работе **38**

1 Практическая работа №9

ThingsBoard имеет тестовый сервер в сборке Community Edition для проверки доступных функций платформы и тестирования своих приложений. Для регистрации на платформе необходимо перейти по данной ссылке.

Зарегистрируемся на платформе ThingsBoard для выполнения данных практических работ.

1.1 Создание виртуальных устройств в облаке

Создадим в облаке следующие виртуальные устройства для получения данных:

1. Датчик качества воздуха;
2. Датчик освещенности;
3. Датчик напряжения.

Создадим для каждого устройства свой профиль (виртуальное устройство), соответствующий передаваемым на устройство данным. В качестве протокола для профилей устройств используем **MQTT** (см. Рисунок 1,2).

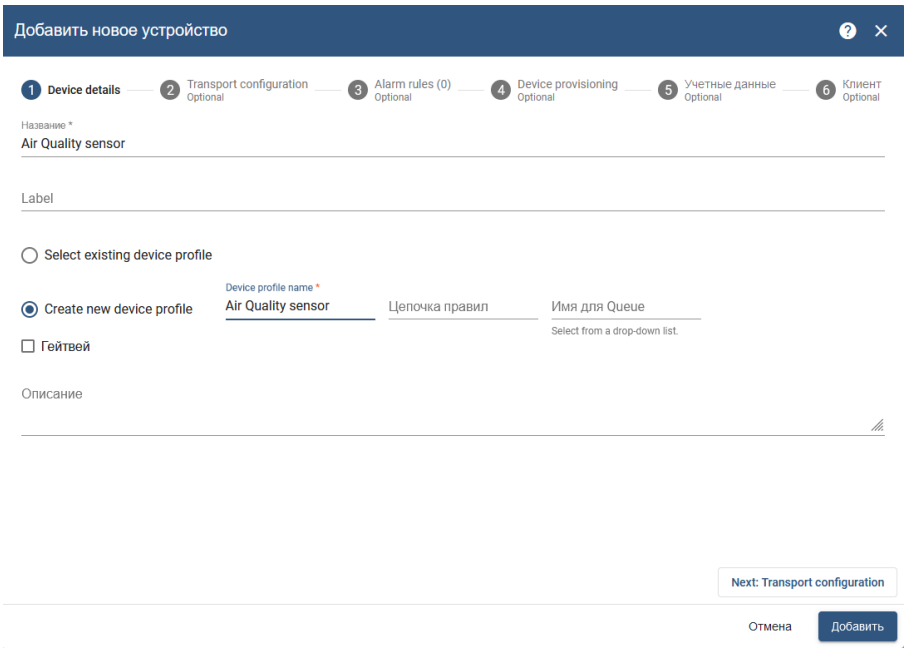


Рисунок 1 — Создание устройства на платформе ThingsBoard

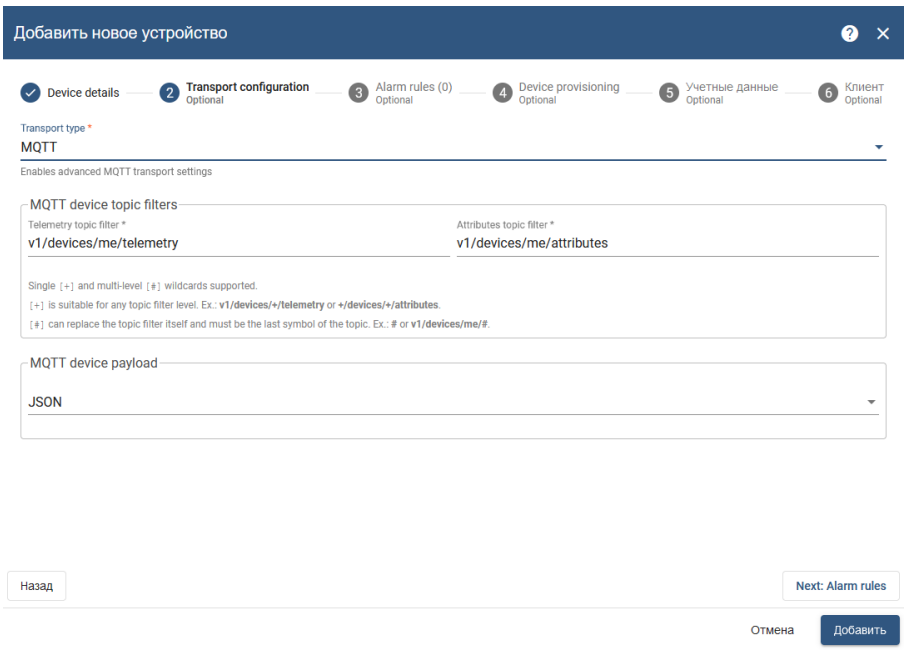


Рисунок 2 — Настройка устройства на платформе ThingsBoard

1.2 Отправка данных в облако

Выполним передачу тестовых данных в каждое из созданных устройств, список которых приведен на Рисунке 3.

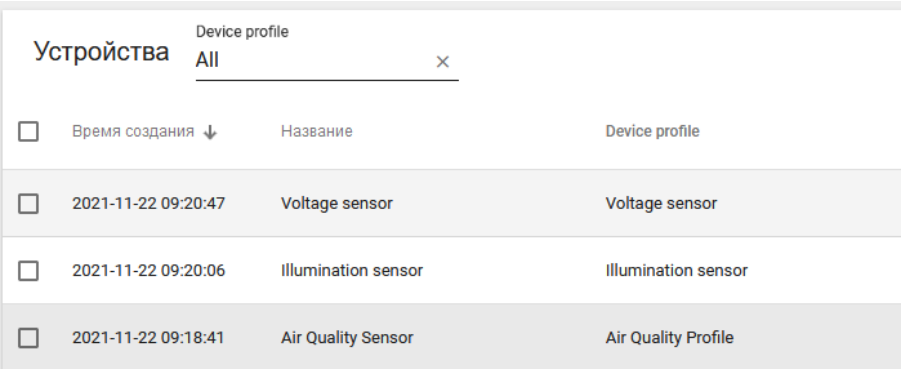


Рисунок 3 — Список зарегистрированных устройств

Приведем команду, с помощью которой осуществляется процесс ответа на сообщение с телеметрией в топик устройства с параметром "motion"(см. Рисунок (см. Рисунок 4).

Рисунок 4 — Отправка данных на устройства

После отправки, тестовые данные отображаются в веб-интерфейсе платформы ThingsBoard в виде представленном на Рисунках 5–7.

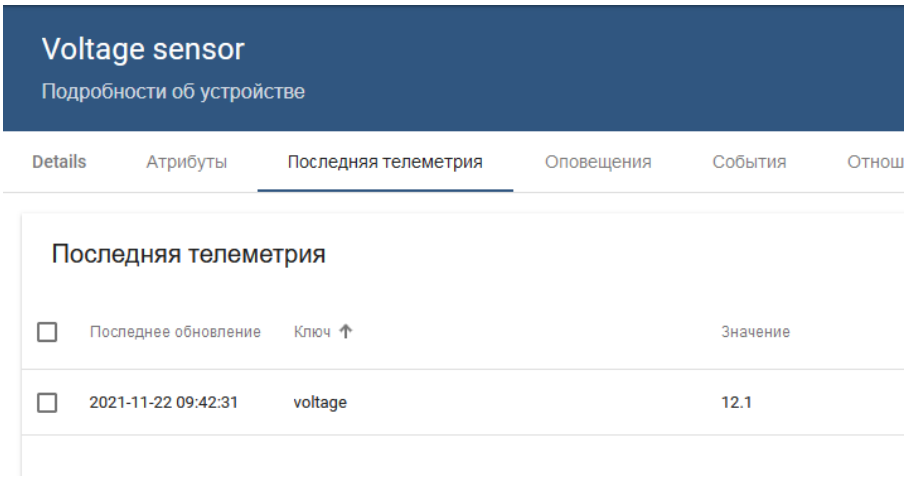


Рисунок 5 — Данные с датчика напряжения

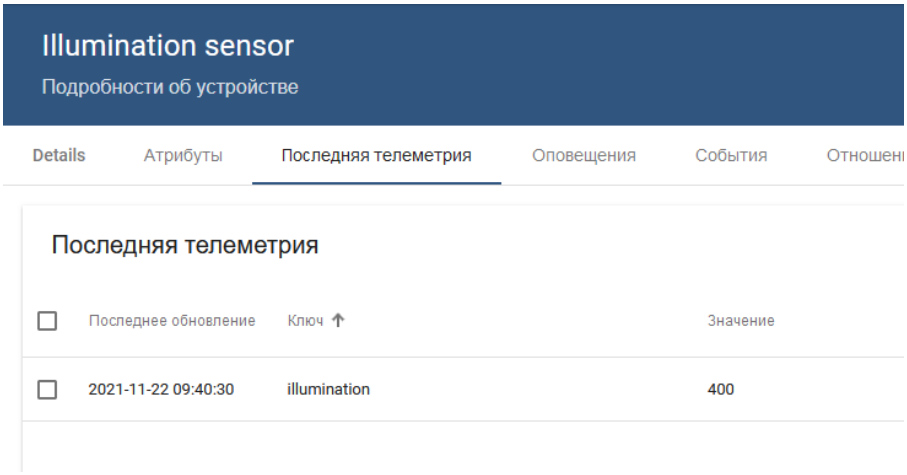


Рисунок 6 — Данные с датчика освещения

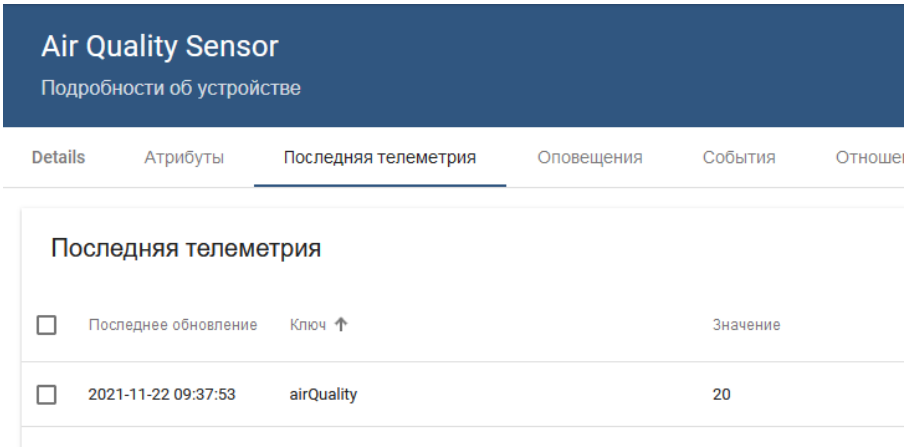


Рисунок 7 — Данные с датчика качества воздуха

Данные соответствуют типу устройства и передаются при помощи утилиты `mosquito_pub`.

2 Дополнительное задание №9

2.1 Выбор облачного решения

В качестве облачного решения была выбрана платформа ThingsBoard. Данный выбор был сделан по следующим причинам:

- Понятный пользовательский интерфейс;
- Популярность платформы;
- Возможность установить платформу локально или использовать готовую облачную среду.

2.2 Реализация отправки данных

Реализуем отправку данных с программного эмулятора реального физического устройства в облачную платформу ThingsBoard. Приведем листинг скрипта на языке программирования Python (см. Листинг 1):

Листинг 1 — Отправка данных в облачную платформу

```
import paho.mqtt.client as paho
import sys
import json
import schedule
import datetime
import time
import random

# Connection parameters to the MQTT broker
broker="demo.thingsboard.io"
port=1883

USERNAME = "i1Ohu4cIi2Q7OqUAHf21" # Login to connect to the broker


def job():
  now = datetime.datetime.now()
  motion = random.randint(20, 800)
  noise = random.randint(20, 800)
  doorState = random.randint(0, 1)
```

```
  print("[" + now.strftime("%H:%M %d.%m.%Y") + "] data: " + "motion: " + str(
   motion) + ", noise: " + str(noise) + ", door: " + ("Closed" if doorState == 0
    else "Open"))

  data = {
    "timestamp" : now.isoformat(),
    "motion" : motion,
    "noise": noise,
    "door_open" : doorState
  }

  pahoClient.publish("v1/devices/me/telemetry", json.dumps(data))


def main():
  # Creating and configuring an instance of the Client class to connect to the
   MQTT broker
  global pahoClient
  pahoClient  = paho.Client("control1")
  pahoClient.username_pw_set(USERNAME)
  pahoClient.connect(broker,port)

  schedule.every(2).seconds.do(job)

  while 1:
    schedule.run_pending()
    time.sleep(1)


if __name__ == "__main__":
    main()
```

В результате запуска данного скрипта происходит соединение с MQTT-брокером, создание экземпляра класса Client для MQTT-брокера, с последнующей передачей данных в облако (см. Рисунки 8, 9).

3 Практическая работа №10

Реализуем следующие сценарии из практической работы №3 при помощи цепочек правил ThingsBoard.

1. Включение и выключение вентилятора по датчику движения;
2. Включение и выключения индикации зеленым и красным светом комбинированного датчика по кнопкам.

```
[10:17 22.11.2021] data: motion: 190, noise: 409, door: Open
[10:17 22.11.2021] data: motion: 103, noise: 418, door: Closed
[10:17 22.11.2021] data: motion: 572, noise: 447, door: Open
[10:17 22.11.2021] data: motion: 633, noise: 440, door: Closed
[10:17 22.11.2021] data: motion: 318, noise: 551, door: Closed
[10:17 22.11.2021] data: motion: 248, noise: 639, door: Open
[10:17 22.11.2021] data: motion: 360, noise: 561, door: Open
[10:17 22.11.2021] data: motion: 761, noise: 30, door: Open
[10:17 22.11.2021] data: motion: 370, noise: 692, door: Open
[10:17 22.11.2021] data: motion: 295, noise: 221, door: Closed
[10:17 22.11.2021] data: motion: 513, noise: 610, door: Open
[10:18 22.11.2021] data: motion: 741, noise: 365, door: Closed
[10:18 22.11.2021] data: motion: 171, noise: 185, door: Closed
[10:18 22.11.2021] data: motion: 134, noise: 458, door: Closed
[10:18 22.11.2021] data: motion: 177, noise: 561, door: Open
[10:18 22.11.2021] data: motion: 391, noise: 378, door: Closed
[10:18 22.11.2021] data: motion: 152, noise: 264, door: Closed
[10:18 22.11.2021] data: motion: 28, noise: 389, door: Open
[10:18 22.11.2021] data: motion: 292, noise: 717, door: Open
[10:18 22.11.2021] data: motion: 363, noise: 625, door: Closed
[10:18 22.11.2021] data: motion: 108, noise: 514, door: Closed
[10:18 22.11.2021] data: motion: 398, noise: 508, door: Open
[10:18 22.11.2021] data: motion: 698, noise: 442, door: Open
[10:18 22.11.2021] data: motion: 67, noise: 35, door: Open
[10:18 22.11.2021] data: motion: 198, noise: 266, door: Open
[10:18 22.11.2021] data: motion: 326, noise: 271, door: Closed
[10:18 22.11.2021] data: motion: 648, noise: 281, door: Open
[10:18 22.11.2021] data: motion: 292, noise: 650, door: Closed
[10:18 22.11.2021] data: motion: 622, noise: 50, door: Closed
```

Рисунок 8 — Оправка телеметрических данных с устройств

Timeseries table

Режим реального времени - Последние 5 минут

Timestamp	door_open	motion	noise
2021-11-22 10:19:58	0	622	50
2021-11-22 10:19:56	0	292	650
2021-11-22 10:19:54	1	648	281
2021-11-22 10:19:52	0	326	271
2021-11-22 10:19:50	1	198	266
2021-11-22 10:19:48	1	67	35
2021-11-22 10:19:46	1	698	442
2021-11-22 10:19:44	1	398	508
2021-11-22 10:19:42	0	108	514
2021-11-22 10:19:40	0	363	625

Рисунок 9 — Прием телеметрических данных облачной платформой

3.1 Реализация сценария управления вентилятором

На платформе ThingsBoard создадим виртуальное устройство, которое будет прообразом реального вентилятора (см. Рисунок 10).

Устройства

Device profile: All

Время создания	Название	Device profile
2021-11-23 16:57:18	Smart fan	Smart fan

Рисунок 10 — Виртуальный вентилятор

Создадим цепочку правил для контроля за состоянием вентилятора. Когда значения, передаваемые датчиком движения превышают 700 условных единиц, вентилятор должен включаться. При уменьшении значения ниже 700, вентилятор должен выключаться. Приведенная на Рисунке 11 цепочка правил описывает данный сценарий управления устройством.

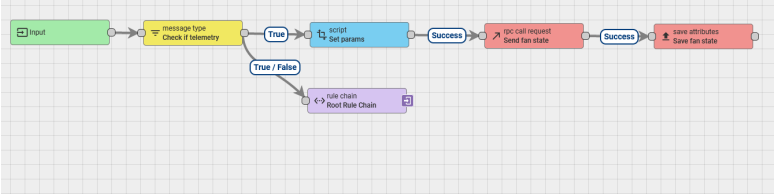


Рисунок 11 — Цепочка правил для управления вентилятором

Узел формирования параметров вентилятора

Узел трансформации данных при помощи скрипта позволяет переформировать объект, содержащий в себе данные приходящего сообщения: его основную полезную нагрузку, метаданные, а также тип сообщения.

Поведение узла описывается при помощи Java Script. Изначально в приходящей телеметрии предполагается наличие параметра `motion`. На основании этого параметра вычисляет новое состояние увлажнителя и формируется новый объект сообщения с этим состоянием.

Данный объект содержит в себе несколько свойств: *method* — это наименование метода, при помощи которого можно будет идентифицировать необходимое действие на устройстве, а также свойство *params*, содержащее как раз состояние устройства, в которое его необходимо привести.

В дальнейшем этот объект будет отправлен на конечное устройство для смены его состояния. Изменим тип события на событие загрузки атрибутов устройства — POST_ATTRUBUTE_REQUEST. Узел возвращает объект, содержащий в себе основное сообщение, метаданные, а также тип сообщения. Полный код приведен в Листинге 2.

Листинг 2 — Отправка объекта на устройство

```
function getNewFanState(motion){
    return motion > 700;
}

let newMsg = {};
let newMsgType = '';
```

```
newMsg = {
    "method" : "setFanState",
    "params":{
        "state": getNewFanState(msg.motion)
    }
};

newMsgType = "POST_ATTRIBUTES_REQUEST";


return {msg: newMsg, metadata: metadata, msgType: newMsgType};
```

После реализации узла формирования параметров можно приступать к тестированию созданной цепочки.

Подпишемся на топик запросов (v1/devices/me/rpc/request/+) созданного для данного виртуального вентилятора. Воспользуемся для этого следующую командой:

```
mosquitto_sub -v -h "demo.thingsboard.io" -t "v1/devices/me/rpc/request/+" -u
  "nS83gBMMgHxwPq4yXVW9"
```

После чего опубликуем сообщение с телеметрией в топик устройства с параметром `motion`:

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/telemetry" -u "
   nS83gBMMgHxwPq4yXVW9" -m "{"motion": 800}"
```

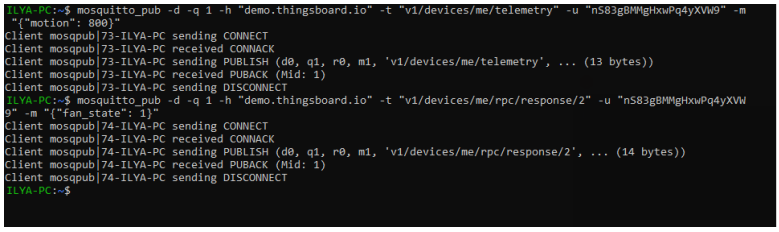


Рисунок 12 — Публикация сообщения с телеметрией вентилятора

Чтобы отправить ответ на опубликованный запрос на смену состояния, воспользуемся также утилитой `mosquito_pub`.

Для отправки ответа на созданный запрос необходимо послать сообщение в топик *v1/devices/me/rpc/response/2*. Воспользуемся для этого следующей командой:

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/rpc/response
 /2" -u "nS83gBMMgHxwPq4yXVW9" -m "{"fan_state": 1}"
```

После этого можно проверить в облаке поступившую телеметрию и аргументы устройства, посланные в ответ на запрос. Результаты представлены на Рисунках 13 и 14.

Последняя телеметрия

Последнее обновление	Ключ	Значение
2021-11-23 17:28:18	motion	800

Рисунок 13 — Телеметрия облачного устройства «вентилятор»

Клиентские атрибуты

Контекст атрибутов объекта: Клиентские атрибуты

Последнее обновление	Ключ	Значение
2021-11-23 17:28:30	fan_state	1

Рисунок 14 — Атрибуты облачного устройства «вентилятор»

3.2 Реализация сценария управления лампами

На платформе ThingsBoard создадим виртуальное устройство, которое будет прообразом реальных блока светодиодных ламп (см. Рисунок 15).

Устройства

Device profile: All

Время создания	Название	Device profile
2021-11-23 17:41:02	Smart lamp	Smart lamp

Рисунок 15 — Виртуальный блок ламп

Создадим цепочку правил для контроля за состоянием блока ламп. Световые индикаторы должны сигнализировать о включении и выключении комбинированного датчика. При включении датчика следует включить зеленую лампу, а при выключении — красную.

Приведенная на Рисунке 16 цепочка правил описывает данный сценарий управления устройством.

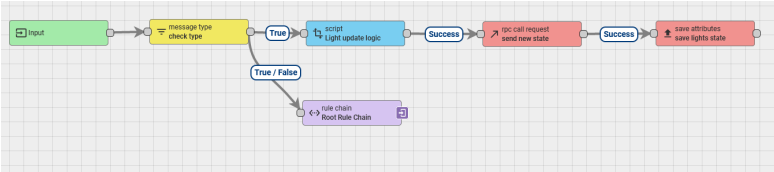


Рисунок 16 — Цепочка правил для управления блоком ламп

Узел формирования параметров вентилятора

Полный код, описывающий поведение узла формирования параметров блока умных ламп приведен в Листинге 3.

Листинг 3 — Блок умных ламп

```
function GetRedLightState(msg){
    return msg.redButton == 1 && msg.greenButton == 0;
}

function GetGreenLightState(msg){
    return msg.redButton == 0 && msg.greenButton == 1;
}

let newMsg = {};
let newMsgType = '';

newMsg = {
    "method" : "setLightsState",
    "params":{
        "redLight": GetRedLightState(msg),
        "greenLight": GetGreenLightState(msg)
    }
};

newMsgType = "POST_ATTRIBUTES_REQUEST";

return {msg: msg, metadata: metadata, msgType: msgType};
```

После реализации узла формирования параметров можно приступать к тестированию созданной цепочки.

Подпишемся на топик запросов (v1/devices/me/rpc/request/+) созданного для данного виртуального контроллера. Воспользуемся для этого следующую командой:

```
mosquitto_sub -v -h "demo.thingsboard.io" -t "v1/devices/me/rpc/request/+" -u "
    RxcWZHP60DTvKzcgWB0T"
```

После чего опубликуем сообщение с телеметрией в топик устройства с параметрами `redButton` и `greenButton`:

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/telemetry" -u "
    RxcWZHP60DTvKzcgWB0T" -m "{"redButton": 1, "greenButton": 0}"
```

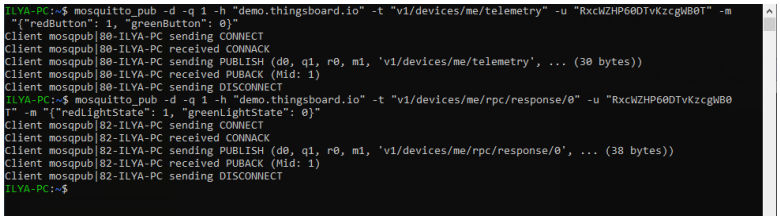


Рисунок 17 — Публикация сообщения с телеметрией блока ламп

Чтобы отправить ответ на опубликованный запрос на смену состояния, воспользуемся также утилитой `mosquito_pub`.

Для отправки ответа на созданный запрос необходимо послать сообщение в топик *v1/devices/me/rpc/response/2*. Воспользуемся для этого следующей командой:

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/rpc/response/2"
    -u "RxcWZHP60DTvKzcgWB0T" -m "{"redLightState": 1, "greenLightState": 0}"
```

После этого можно проверить в облаке поступившую телеметрию и аргументы устройства, посланные в ответ на запрос. Результаты представлены на Рисунках 18 и 19.

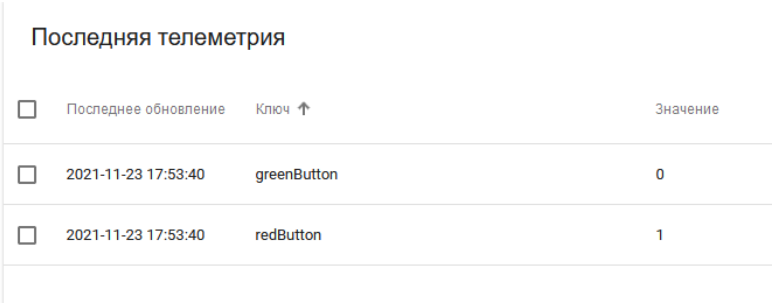


Рисунок 18 — Телеметрия облачного устройства «блок светодиодных ламп»

Клиентские атрибуты

Контекст атрибутов объекта
Клиентские атрибуты

Последнее обновление	Ключ	Значение
2021-11-23 17:54:25	greenLightState	0
2021-11-23 17:54:25	redLightState	1

Рисунок 19 — Атрибуты облачного устройства «блок светодиодных ламп»

4 Дополнительное задание № 10

Описание взаимодействия пользователя с интерфейсом

Опишем взаимодействие пользователя с интерфейсом реализуемой системы при помощи диаграммы Use-case в нотации UML (см. Рисунок 20).

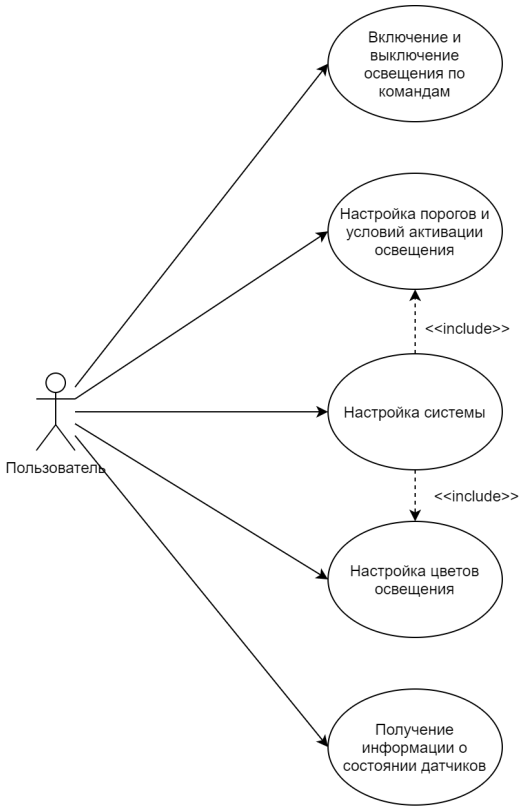


Рисунок 20 — Взаимодействие пользователя с системой

4.1 Требования к интерфейсу пользователя

Для разрабатываемого нами программного решения был выбран интерфейс telegram–бота. Взаимодействие с данным интерфейсом состоит

в отправке текстовых сообщений в мессенджере и основано на принципе «запрос-ответ» — данные предоставляются пользователю по требованию.

Для удобного взаимодействия с информационной системой данные должны предоставляться пользователю в графическом, табличном и текстовом виде. Это упросит взаимодействие с программным инструментом.

Следует реализовать возможность выбора отслеживаемого на данный момент датчика, регистрации нового датчика (прибора), получения сведений о состоянии измерительного прибора, получении текущей телеметрии и статистических данных.

Сведения о передаваемой датчиками телеметрии следует представлять в виде таблиц, графиков или структурированного текста. Также нужно предоставить пользователю возможность получать сведения о данных за определенный временной промежуток, проводить их агрегирование.

4.2 Макеты интерфейса приложения

После проведения анализа существующих решений было сформировано представление о том, как следует выстроить взаимодействие с пользователем.

Пользователю предлагается набор команд, которые понимает telegram–бот, их список приведен над полем ввода. На каждую команду бот отвечает установленным образом, предоставляя пользователю информацию о состоянии элементов системы.

Решения использующие аналогичный интерфейс взаимодействия приведены на Рисунках 21–23.

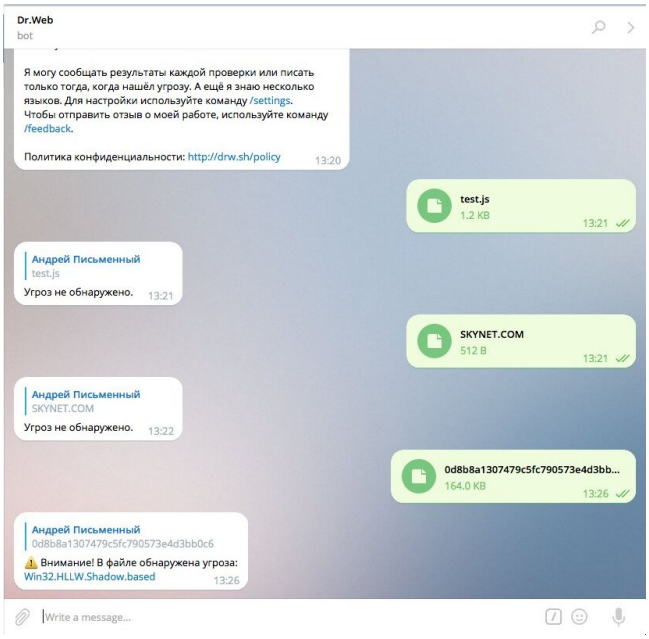


Рисунок 21 — Макет 1

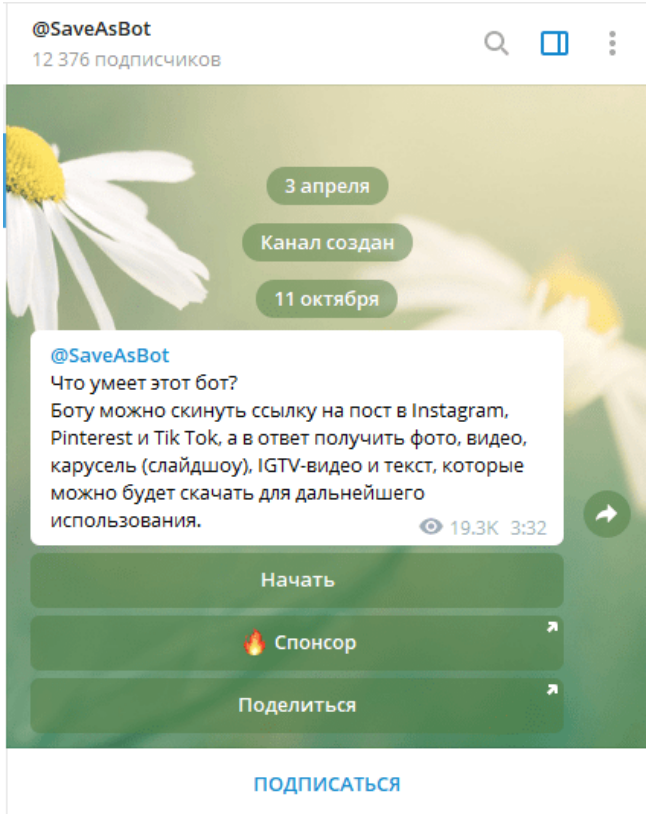


Рисунок 22 — Макет 2

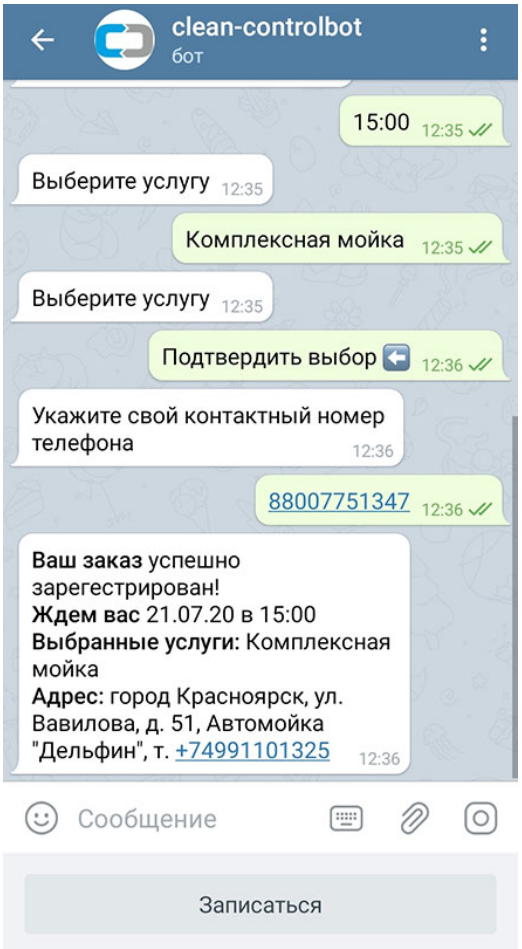


Рисунок 23 — Макет 3

5 Практическая работа № 11

5.1 Обработка тревожных сигналов вентилятора

Добавим в цепочки правил управления вентилятором тревогу при выходе приходящего параметра за допустимые границы, а также тревогу при поступлении неверного ответа от физического устройства.

Для реализации механизма тревог внесем изменения в цепочку правил для управления умным вентилятором — добавим узел сохранения атрибутов запроса *(Save request atributes)* перед узлом отправки запроса и узел получения атрибутов запроса *(Get request atributes)* вместо последнего узла сохранения атрибутов ответа.

Также добавим два узла проверки атрибутов и статуса возвращаемого от устройства ответа *(Check request answer)*. Осуществим проверку при помощи двух скриптов из раздела фильтрующих блоков. Приведем скрипты проверки параметров (см. Листинги 4, 5):

Листинг 4 — Проверка абсолютных значений приходящего параметра

```
return msg.motion >= 10 && msg.motion <= 2000;
```

Листинг 5 — Проверка корректности значений приходящего параметра

```
let request_params = JSON.parse(metadata.ss_params);

return msg.fan_state === request_params.state;
```

Поскольку параметры для запроса были записаны в JSON формате необходимо распарсить JSON строку в объект JS для извлечения параметров при помощи функции JSON.parse. Подключим получившийся узел к узлу получения атрибутов запроса.

Далее создадим два узла активации тревоги в случае, если проверка не будет пройдена. Создадим узел *Create alarm*, отвечающий за проверку соответствия параметров заданному диапазону и зададим ему следующий скрипт в качестве обработчика (см. Листинг 6).

Листинг 6 — Формирование тревоги в случае выхода значений параметра за установленные границы

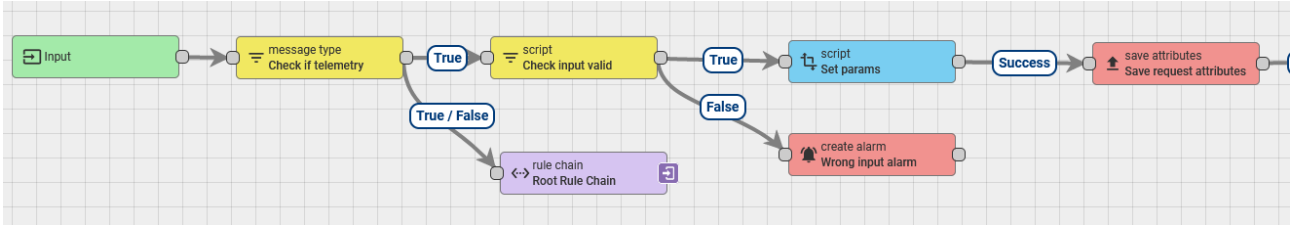
```
var details = {};
if (metadata.prevAlarmDetails) {
    details = JSON.parse(metadata.prevAlarmDetails);
    //remove prevAlarmDetails from metadata
    delete metadata.prevAlarmDetails;
    //now metadata is the same as it comes IN this rule node
}

details.recievedMotion = msg.motion;

return details;
```

Создадим узел *Create alarm*, отвечающий за проверку корректности параметров и зададим ему следующий скрипт в качестве обработчика (см. Листинг 7).

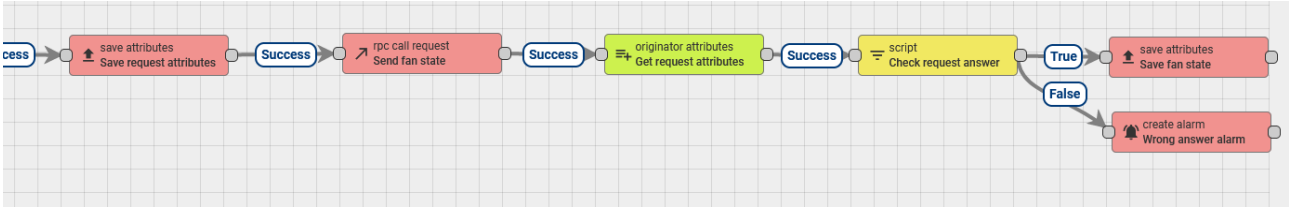
Листинг 7 — Формирование тревоги по случаю некорректности приходящих параметров

```
var details = {};
var request_params = JSON.parse(metadata.ss_params);

if (metadata.prevAlarmDetails) {
    details = JSON.parse(metadata.prevAlarmDetails);
    //remove prevAlarmDetails from metadata
    delete metadata.prevAlarmDetails;
    //now metadata is the same as it comes IN this rule node
}

details.send_status = request_params.state;
details.answer_status = msg.fan_state;

return details;
```

Приведем измененную цепочку правил (см. Рисунок 24).



а)



б)

Рисунок 24 — Измененная цепочка управления умным вентилятором

Тестирование цепочки правил

Проведем тестирование созданной цепочки при помощи утилит mosquitto. Подпишемся на топики виртуального устройства (см Рисунок 25).

```
ILYA-PC:~$ mosquitto_sub -v -h "demo.thingsboard.io" -t "v1/devices/me/rpc/request/+" -u "nS83gBMMgHxwPq4yXVW9"
v1/devices/me/rpc/request/4 {"method":"setFanState","params":{"state":true}}
```

Рисунок 25 — Подписка на топик умного вентилятора

Протестируем цепочку при отправке параметром, выходящих за границы определенного диапазона (см. Рисунок 26).

```
ILYA-PC:~$ mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/telemetry" -u "nS83gBMMgHxwPq4yXVW9" -m
 "{"motion": -800}"
Client mosqpub|64-ILYA-PC sending CONNECT
Client mosqpub|64-ILYA-PC received CONNACK
Client mosqpub|64-ILYA-PC sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (14 bytes))
Client mosqpub|64-ILYA-PC received PUBACK (Mid: 1)
Client mosqpub|64-ILYA-PC sending DISCONNECT
ILYA-PC:~$
```

Рисунок 26 — Отправка некорректной телеметрии

Вслед за эти ThingsBoard отправит оповещение уровня *Critital* (см. Рисунок 27).

Приведем подробные сведения о тревоге (см. Рисунок 28).

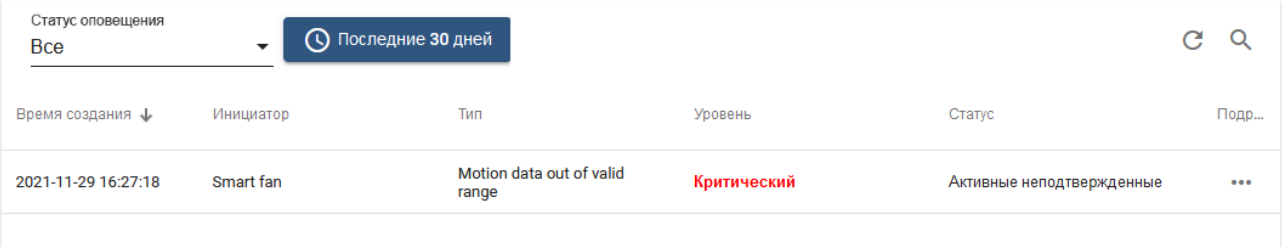


Рисунок 27 — Тревога. Данные выходят за границы заданного диапазона

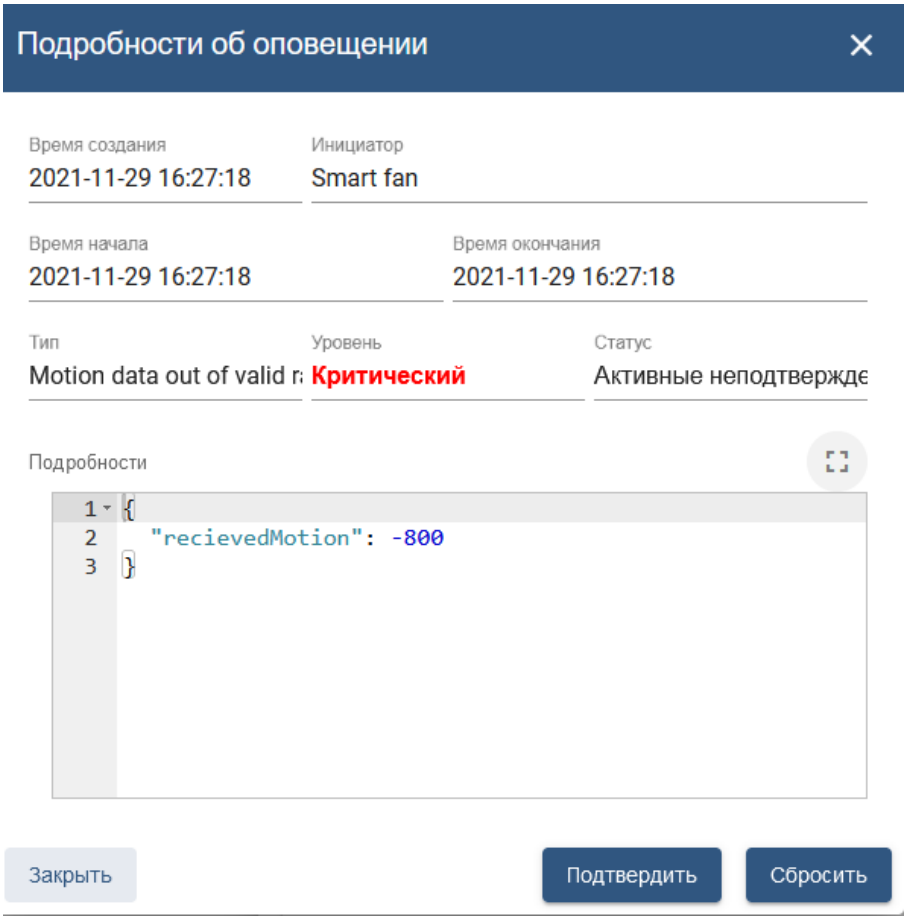


Рисунок 28 — Подробные сведения о тревоге

Протестируем цепочку при получении неверного ответа от устройства. Отправим данные об уровне движения в диапазоне, в котором должно происходить включение вентилятора. Вместе с этим отправим сообщение о выключении вентилятора, что противоречит ожидаемому поведению прибора (см. Рисунок 29).

Платформа ThingsBoard уведомит о некорректности отправленного сообщения, сравнив его с ожидаемым результатом и генерирует соответствующий тревожный сигнал (см. Рисунок 30).

```
ILYA-PC:~$ mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/telemetry" -u "nS83gBMMgHxwPq4yXVW9" -m
 "{\"motion\": 800}"
Client mosqpub|98-ILYA-PC sending CONNECT
Client mosqpub|98-ILYA-PC received CONNACK
Client mosqpub|98-ILYA-PC sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (15 bytes))
Client mosqpub|98-ILYA-PC received PUBACK (Mid: 1)
Client mosqpub|98-ILYA-PC sending DISCONNECT
ILYA-PC:~$ mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/rpc/response/4" -u "nS83gBMMgHxwPq4yXVW
9" -m "{\"fan_state\": false}"
Client mosqpub|101-ILYA-PC sending CONNECT
Client mosqpub|101-ILYA-PC received CONNACK
Client mosqpub|101-ILYA-PC sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/rpc/response/4', ... (20 bytes))
Client mosqpub|101-ILYA-PC received PUBACK (Mid: 1)
Client mosqpub|101-ILYA-PC sending DISCONNECT
ILYA-PC:~$
```

Рисунок 29 — Подробные сведения о тревоге

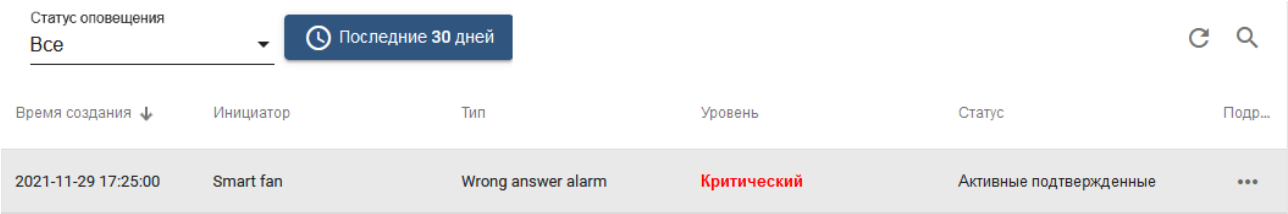


Рисунок 30 — Тревога. Некорректные данные

Приведем подробности тревоги (см. Рисунок 31).

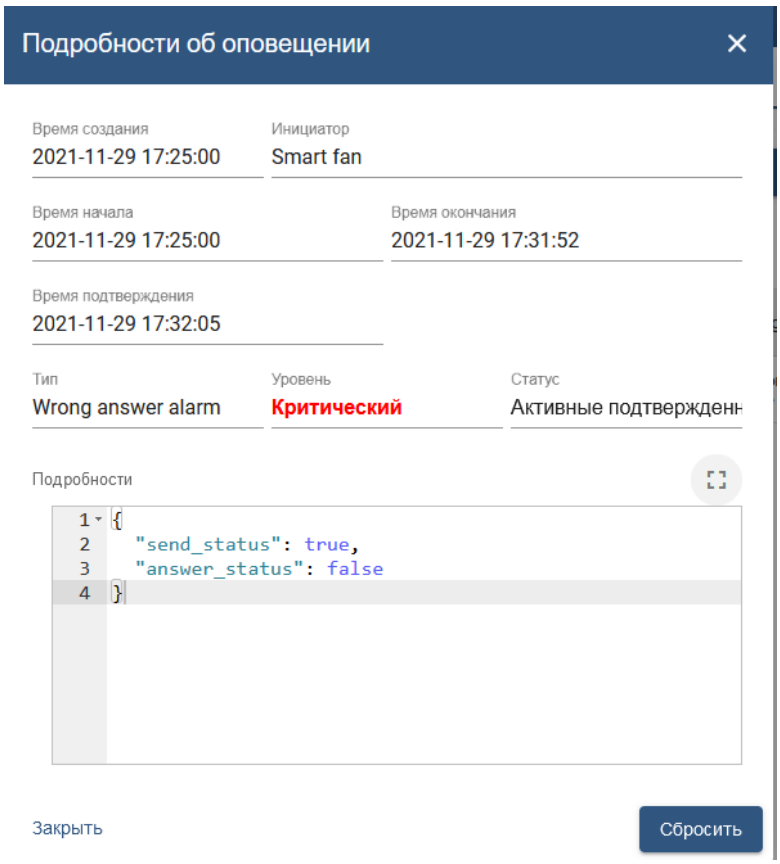


Рисунок 31 — Подробные сведения о тревоге

Сымитировав оба сценария, мы убедились в том, что цепочка правил

управления умным вентилятором отработала корректно. Тревоги сработали в соответствии с установленными правилами.

5.2 Обработка тревожных сигналов блока ламп

Для цепочки управления блоком ламп добавим тревогу при отсутствии ожидаемого параметра в приходящем сообщении, а также тревогу при поступлении неверного ответа от физического устройства.

Проделаем аналогичные шаги. Создадим узел проверки атрибутов и статуса возвращаемого от устройства ответа *(Check request answer)*, отвечающий за проверку наличия необходимых параметров (см. Листинг 8).

Листинг 8 — Проверка наличия требуемых параметров

```
return msg.redButton != undefined && msg.greenButton != undefined;
```

Создадим узел проверки атрибутов и статуса возвращаемого от устройства ответа *(Check request answer)*, отвечающий за проверку корректности приходящих параметров (см. Листинг 9).

Листинг 9 — Проверка корректности значений приходящего параметра

```
let request_params = JSON.parse(metadata.ss_params);

return msg.redLightState === request_params.redLight && msg.greenLightState ===
    request_params.greenLight;
```

Поскольку параметры для запроса были записаны в JSON формате необходимо распарсить JSON строку в объект JS для извлечения параметров при помощи функции JSON.parse. Подключим получившийся узел к узлу получения атрибутов запроса.

Далее создадим узел активации тревоги в случае, если проверка не будет пройдена. Создадим узел *Create alarm*, инициирующий тревогу в случае отсутствия необходимых параметров и зададим ему следующий скрипт в качестве обработчика (см. Листинг 10).

Листинг 10 — Формирование тревоги в случае отсутствия необходимых

```
var details = {};
if (metadata.prevAlarmDetails) {
    details = JSON.parse(metadata.prevAlarmDetails);
    //remove prevAlarmDetails from metadata
    delete metadata.prevAlarmDetails;
```

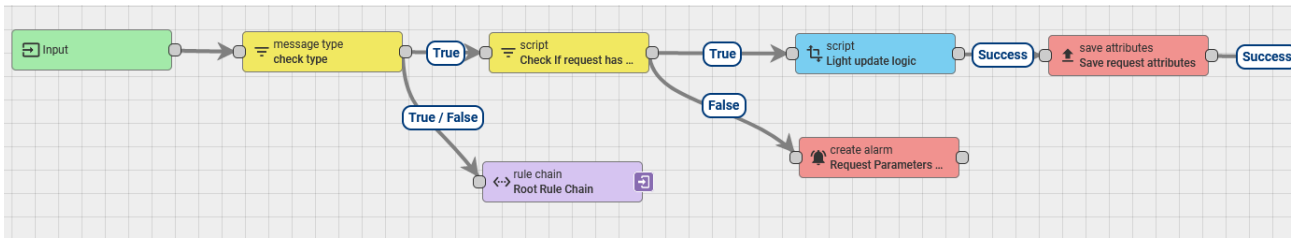
```
    //now metadata is the same as it comes IN this rule node
}

details.received_message = msg;

return details;
```

Создадим узел *Create alarm*, отвечающий за проверку корректности параметров и зададим ему следующий скрипт в качестве обработчика (см. Листинг 11).

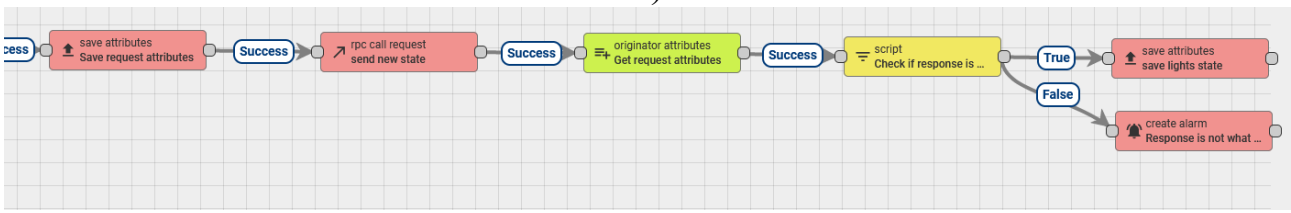
Листинг 11 — Формирование тревоги по случаю некорректности приходящих параметров

```
var details = {};
var request_params = JSON.parse(metadata.ss_params);

if (metadata.prevAlarmDetails) {
    details = JSON.parse(metadata.prevAlarmDetails);
    //remove prevAlarmDetails from metadata
    delete metadata.prevAlarmDetails;
    //now metadata is the same as it comes IN this rule node
}

details.expected = request_params;
details.received = msg;

return details;
```

Приведем измененную цепочку правил (см. Рисунок 32).



а)



б)

Рисунок 32 — Измененная цепочка управления блоком умных ламп

Тестирование цепочки правил

Проведем тестирование созданной цепочки при помощи утилит mosquitto.

Подпишемся на топики виртуального устройства (см Рисунок 33).

```
ILYA-PC:~$ mosquitto_sub -v -h "demo.thingsboard.io" -t "v1/devices/me/rpc/request/+" -u "RxcWZHP60DTvKzcgWB0T"
v1/devices/me/rpc/request/0 {"method":"setLightsState","params":{"redLight":true,"greenLight":false}}
```

Рисунок 33 — Подписка на топик блока умных ламп

Протестируем цепочку при отправке сообщения с неверным количеством параметров (см. Рисунок 34).

```
ILYA-PC:~$ mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/telemetry" -u "RxcWZHP60DTvKzcgWB0T" -m
 "{\"redButton\": 1}"
Client mosqpub|120-ILYA-PC sending CONNECT
Client mosqpub|120-ILYA-PC received CONNACK
Client mosqpub|120-ILYA-PC sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (16 bytes))
Client mosqpub|120-ILYA-PC received PUBACK (Mid: 1)
Client mosqpub|120-ILYA-PC sending DISCONNECT
ILYA-PC:~$
```

Рисунок 34 — Отправка телеметрии с отсутствующими параметрами

Вслед за эти ThingsBoard отправит оповещение уровня *Critital* (см. Рисунок 35).

Статус оповещения
Все
Последние 30 дней

Время создания	Инициатор	Тип	Уровень	Статус	Подр...
2021-11-29 18:27:05	Smart lamp	Request Parameters missing alarm	Критический	Активные неподтвержденные	•••

Рисунок 35 — Тревога. Необходимый параметр отсутствует

Приведем подробные сведения о тревоге (см. Рисунок 36).

Протестируем цепочку при получении неверного ответа от устройства. Отправим данные об уровне движения в диапазоне, в котором должно происходить включение вентилятора. Вместе с этим отправим сообщение о выключении вентилятора, что противоречит ожидаемому поведению прибора. (см. Рисунок 37).

Платформа ThingsBoard уведомит о некорректности отправленного сообщения, сравнив его с ожидаемым результатом и сгенерирует соответствующий тревожный сигнал (см. Рисунок 38).

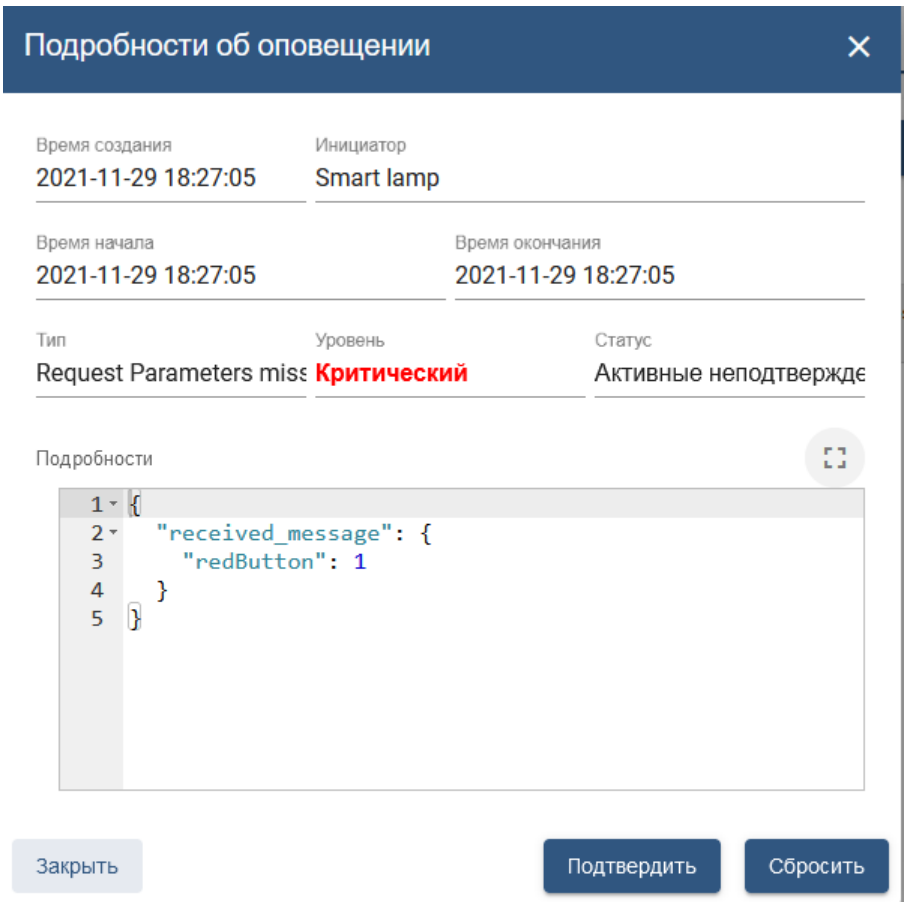


Рисунок 36 — Подробные сведения о тревоге

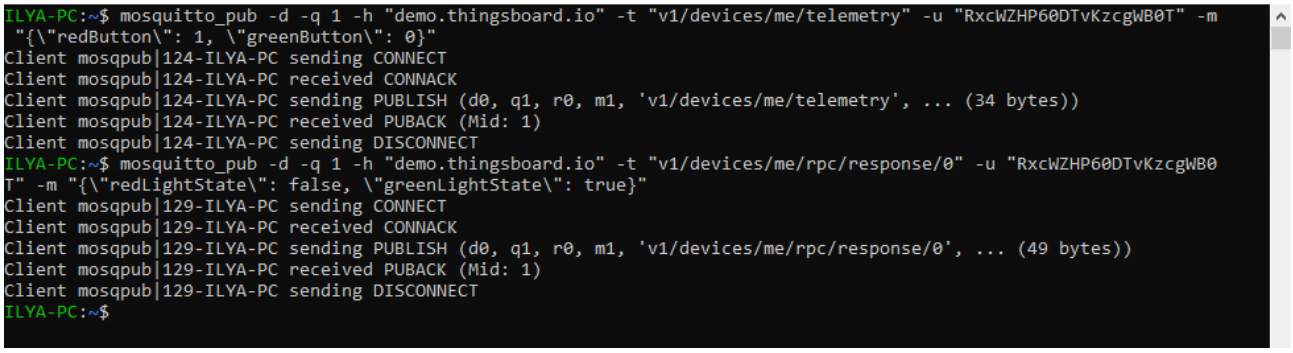


Рисунок 37 — Подробные сведения о тревоге

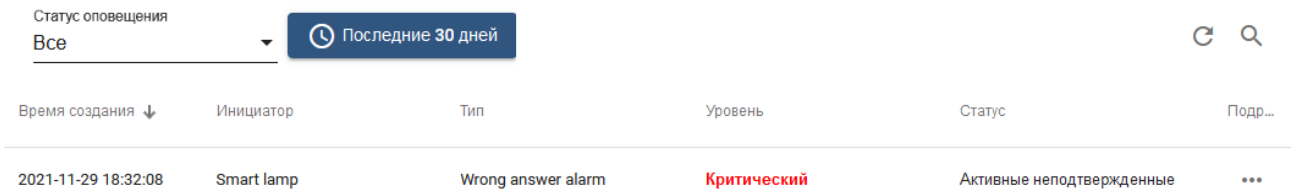


Рисунок 38 — Тревога. Некорректные данные

Приведем подробности тревоги (см. Рисунок 39).

Сымитировав оба сценария, мы убедились в том, что цепочка правил управления умным вентилятором отработала корректно. Тревоги сработали в

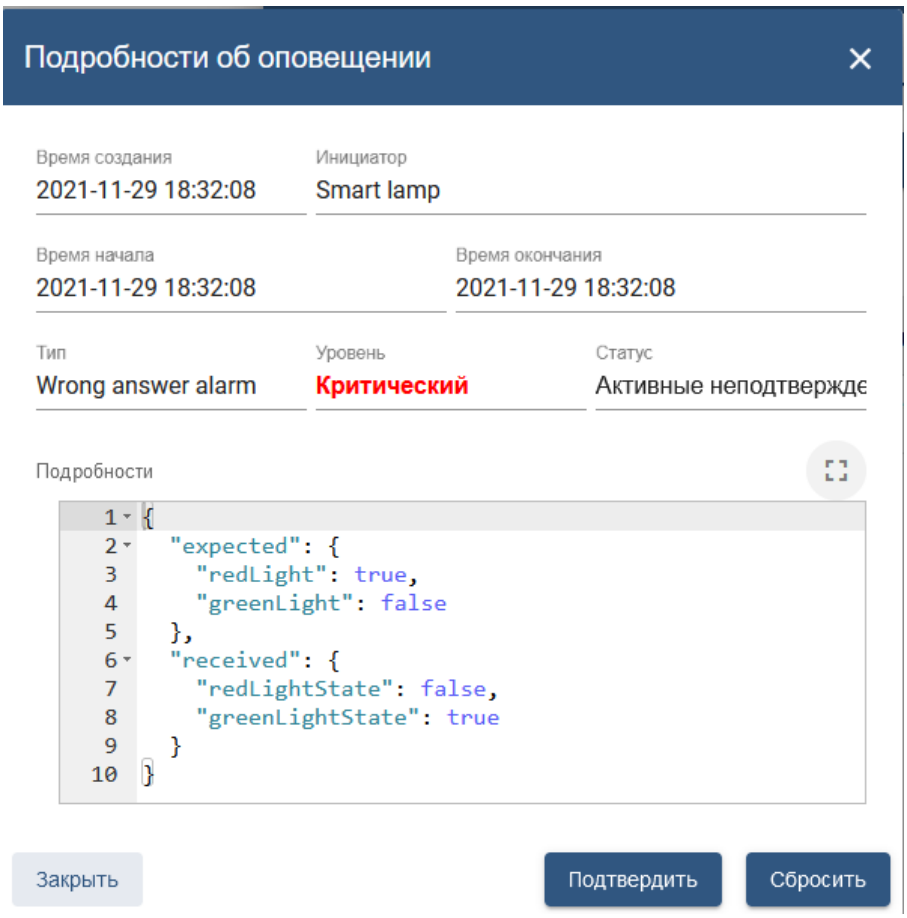


Рисунок 39 — Подробные сведения о тревоге

соответствии с установленными правилами.

6 Дополнительное задание № 11

6.1 Описание технологического стека проекта

Пользовательский интерфейс системы реализован в виде диалога с Telegram-ботом, поведение которого определяет скрипт написанный на языке программирования Python и программный интерфейс Telegram.

На языке Python также написаны скрипы, осуществляющие соединение с MQTT–брокером, эмуляцию физических устройств, а также реализующие процесс авторизации и соединения с локальной базой данных. Данный язык программирования был выбран с оглядкой на его широкую распространенность, наличие полной документации и универсальность.

Данные о зарегистрированных пользователях и привязанных к ним виртуальных и физических устройствах хранятся локально в базе данных

под управлением СУБД SQLite. Выбор в пользу данного решения сделан из–за простоты в обращении с ним, распространенности решения и его компактности.

В качестве облачного сервиса выбрана платформа ThingsBoard. Данная платформа имеет дружелюбный пользовательский интерфейс. Использование сервиса возможно на бесплатной основе, что также является весомым плюсом данной платформы.

Также в проекте используется язык программирования JavaScript. Это универсальный язык для веб приложений, который отлично подходит для реализации нашего проекта.

6.2 Реализация пользовательского интерфейса

Реализуем пользовательский интерфейс на основании макетов интерфейса приложения, разработанных в десятой практической работе.

Пользователь начинает взаимодействие с интерфейсом с авторизации в системе, затем ему следует произвести привязку приборов, указав их secret–key и имя прибора. После привязки устройств, пользователь может вывести их список, получить текущую телеметрию, изменить параметры устройств. Приведем полный список команд, распознаваемых системой (см. Таблицу 1 и Рисунок 40):

Таблица 1 — Список команд

Команда	**Параметры**	**Назначение**
/login	<login> <password>	Авторизация
/claim	<secret key>	Привязка нового устройства к аккаунту
/devices	–	Отобразить список устройств
/telemetry	–	Вывод телеметрии
/set_light	<on|off>	Включение/выключение света
/set_color	<html-color-code>	Установить цвет свечения лампы
/start_timer	<hh> <mm>	Установить таймер на включение света

Процесс авторизации отображен на Рисунке 41.

Процесс привязки устройства отображен на Рисунке 42.

После привязки устройства, оно отобразится при выполнении команды `/devices` (см. Рисунок 43).

Получить телеметрию устройства можно отправив боту команду `/telemetry` (см. Рисунок 44).

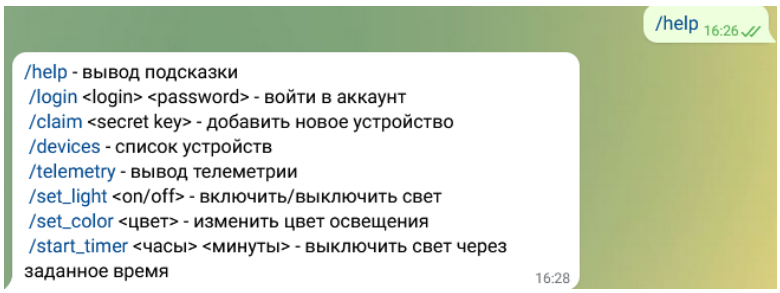


Рисунок 40 — Команды Telegram–бота

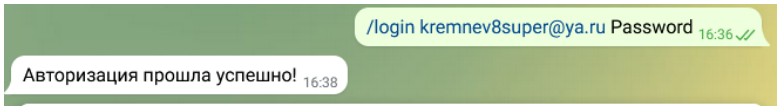


Рисунок 41 — Процесс авторизации

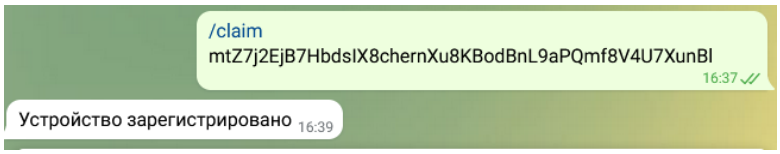


Рисунок 42 — Процесс привязки устройства

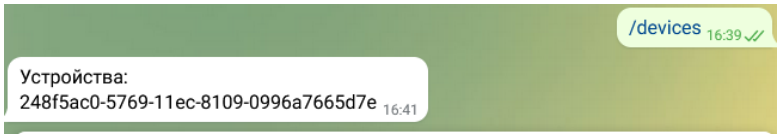


Рисунок 43 — Список устройств

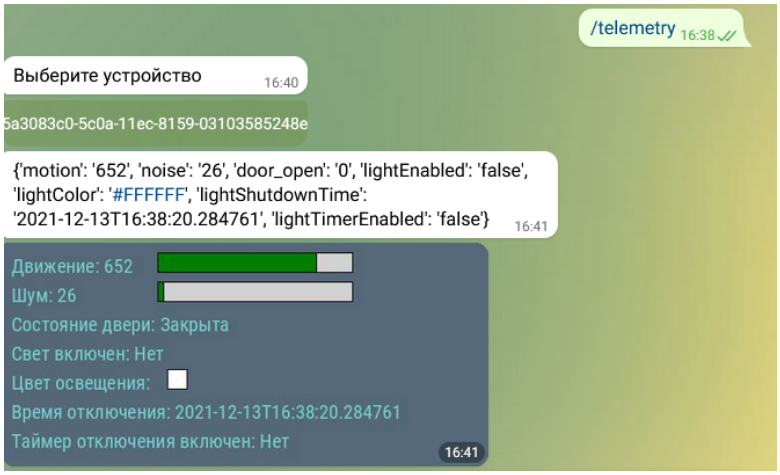


Рисунок 44 — Получение телеметрии устройства

Включение устройства производится выполнением команды `/set_on` с указанием secret–key устройства. Резутат выполнения команды можно видеть на Рисунке 45.

Если устройство предоставляет возможность смены параметра «цвет»,

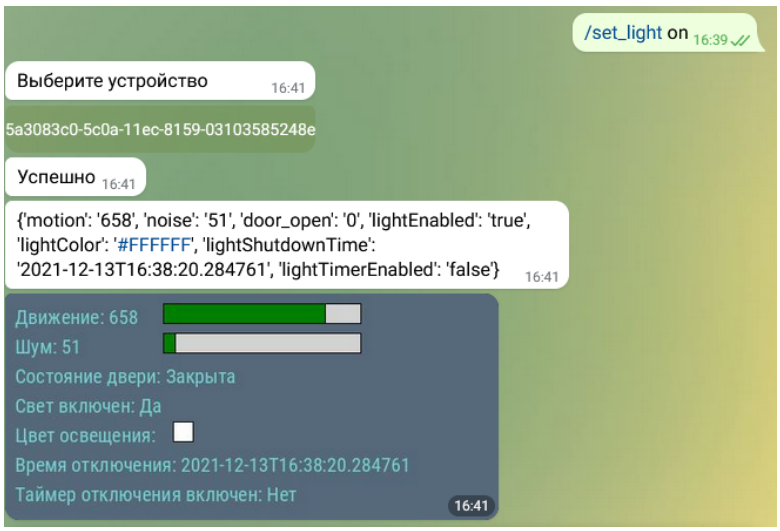


Рисунок 45 — Включение устройства

изменить его можно выполнив команду `/set_color`, указав в качестве параметра HTML код цвета (см. Рисунок 46).

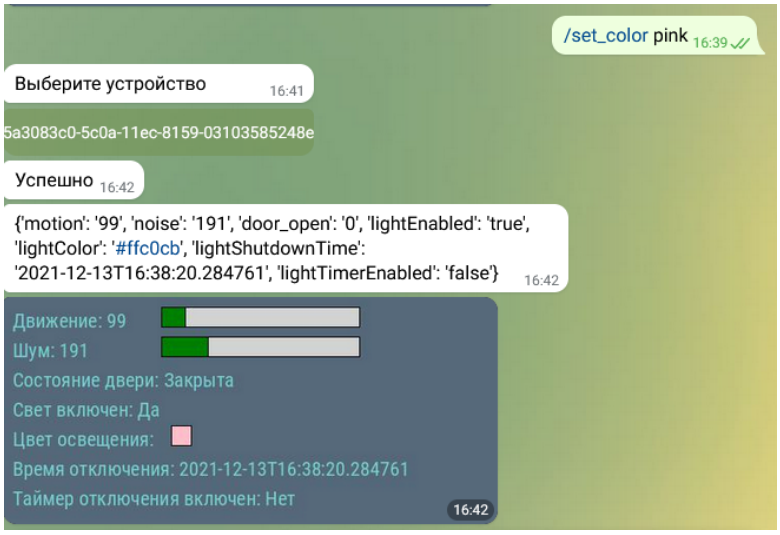


Рисунок 46 — Процедура смена цвета

Платформа предоставляет возможность отключения и включения устройства с задержкой. Данный функционал реализует команда `/start_timer`, где качестве параметров выступают часы и минуты (см. Рисунок 47).

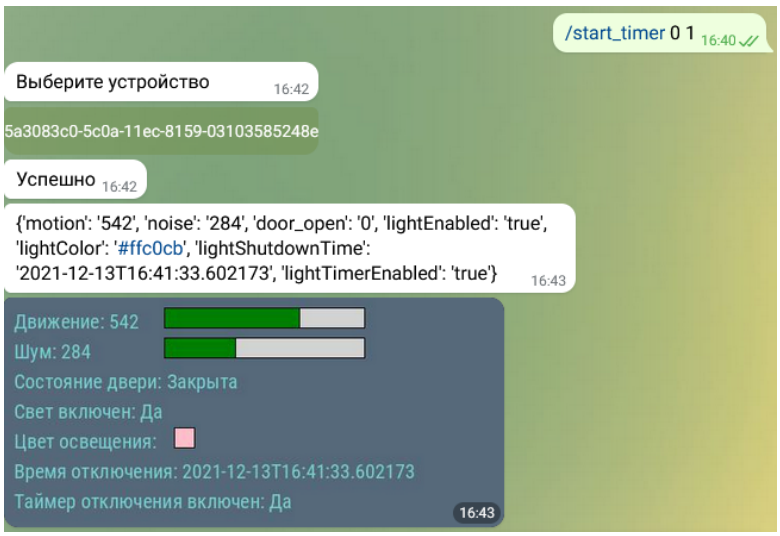


Рисунок 47 — Установка таймера

Данные команды позволяют пользователю управлять устройствами умного дома, подключенными к облачной платформе Интернета вещей, объединять их в группы, путем создания ролей и пользователей. Также возможно создание гостевой роли, предоставляющей ограниченный доступ к функционалу системы.

7 Практическая работа №12

Реализуем отправку Email сообщений из облачной платформы при возникновении тревог на узлах, созданных в практической работе №11. В качестве SMTP сервера для пересылки сообщений предлагается будем использовать сервера Yandex и Google.

7.1 Отправка Email сообщений. Умный вентилятор

Дополним цепочку управления работой умного вентилятора таким образом, чтобы после возникновения тревог критического уровня на электронный ящик администратора платформы поступало письмо с описанием проблемы.

Узел формирования Email сообщения

Для реализации возможности отправки сообщений добавим узел формирования Email сообщений To email. Установим необходимые

параметры — адрес отправителя, адрес получателя, заголовок письма, а также тело письма (см. Рисунок 48).

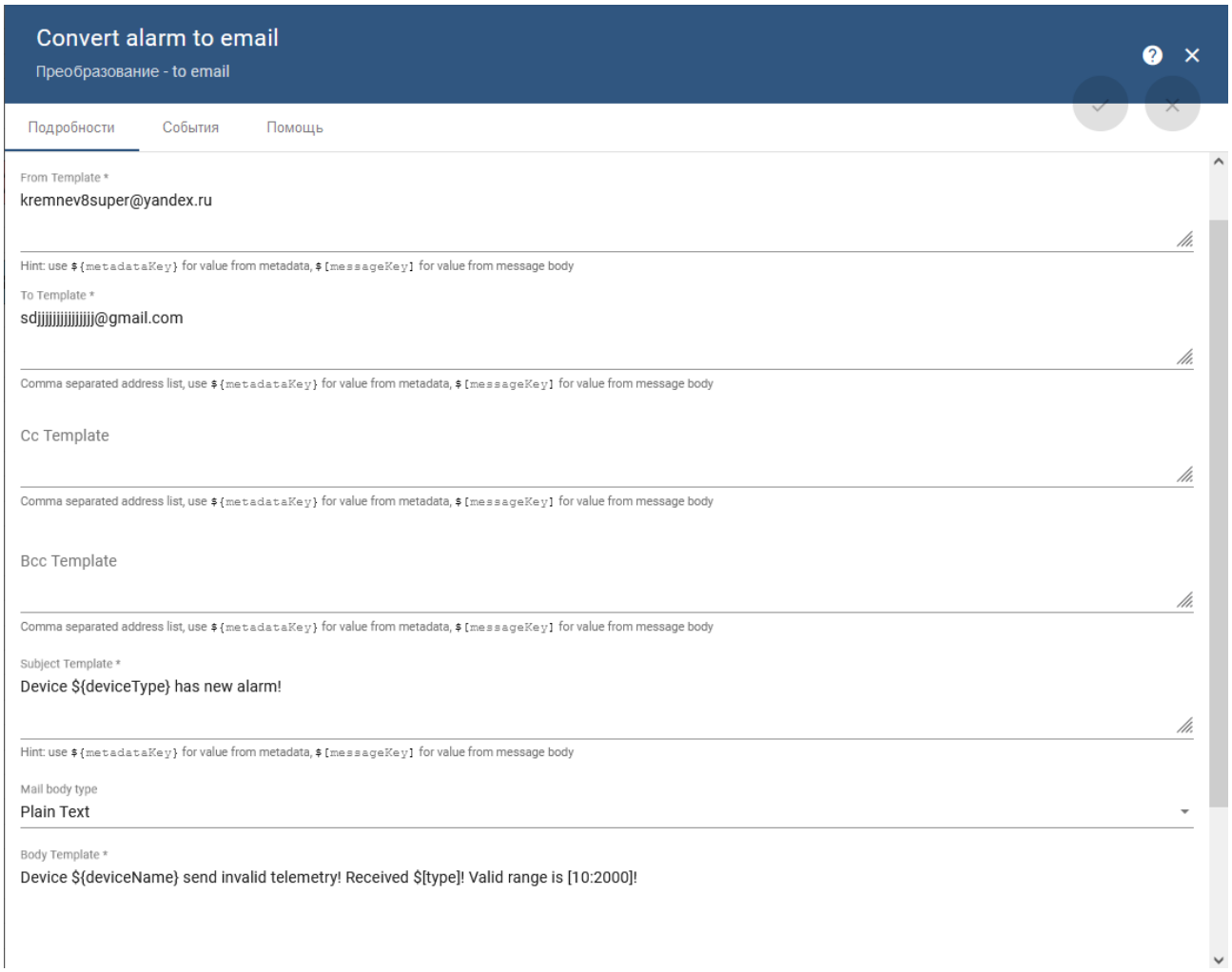


Рисунок 48 — Узел формирования Email сообщения

Добавим узлы в цепочку. Необходимо создать два идентичных узла формирования Email сообщения, которые следует соединить с узлами генерации тревог критического уровня.

Узел отправки Email сообщения

После формирования сообщения можно приступать к настройке узла отправки сообщения. В качестве протокола передачи данных выберем протокол SMTPS, поскольку современные почтовые сервисы передают данные только по шифрованным протоколам. В качестве STMP сервера укажем сервер Yandex с доменным именем smtp.yandex.ru. Порт 465. Таймаут установим равным 20 секундам.

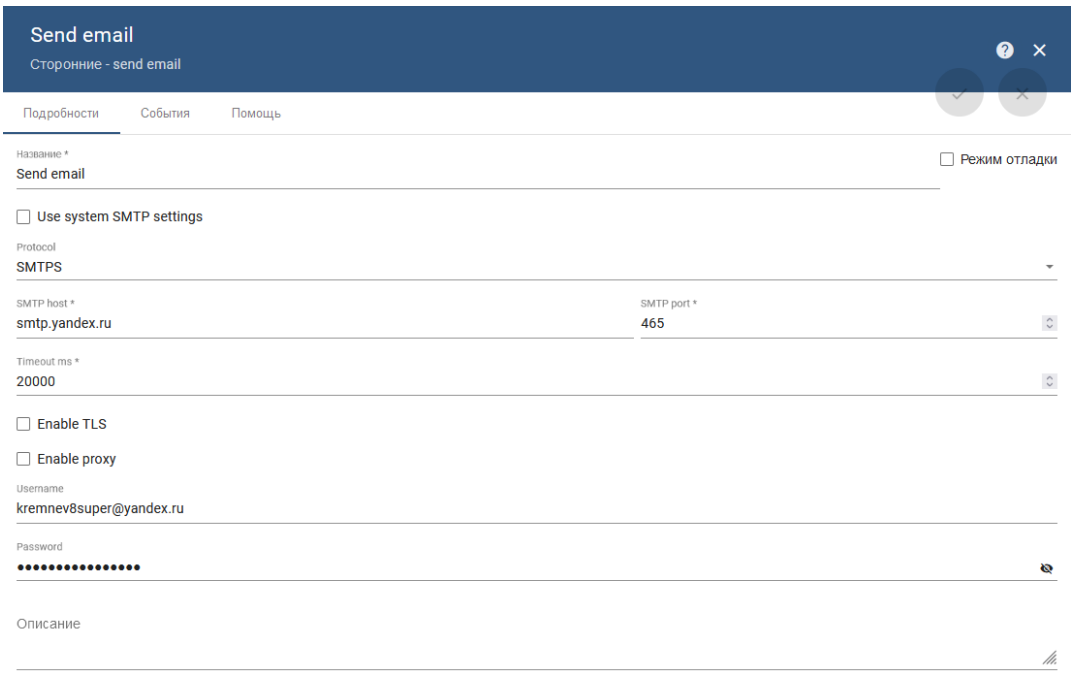
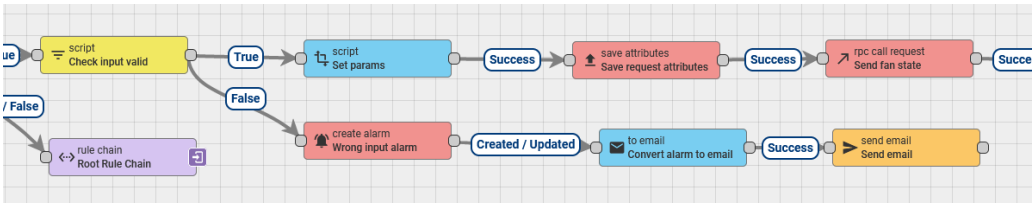


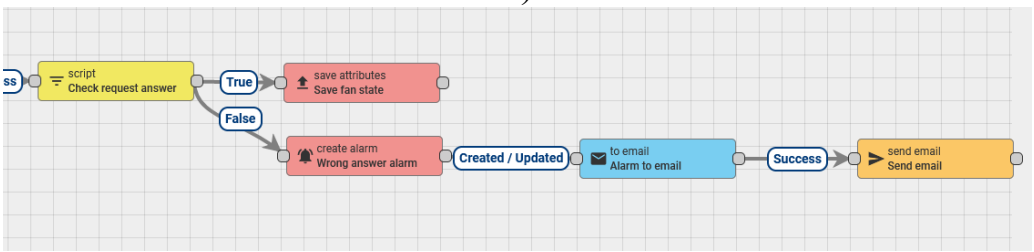
Рисунок 49 — Узел отправки Email сообщения

Два узла отправки сообщений соединим с уже добавленными узлами формирования Email сообщений.

Приведем измененную цепочку (см. Рисунок 50).



а)



б)

Рисунок 50 — Измененная цепочка управления вентилятором

Тестирование созданной цепочки

Проведем тестирование созданной цепочки при помощи утилит mosquitto.

Воспроизведем действия, описанные в разделе 5.1. Впоследствии мы должны обнаружить на почте, указанной в параметрах узла отправки Email сообщения письма, содержащие информацию о возникновении тревожных ситуаций (см. Рисунки 51, 52).

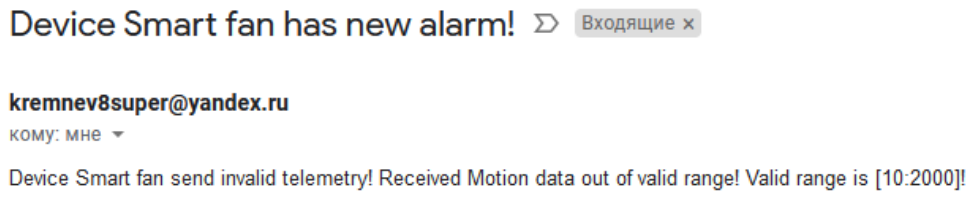


Рисунок 51 — Сообщение о несоответствии параметра заданному диапазону

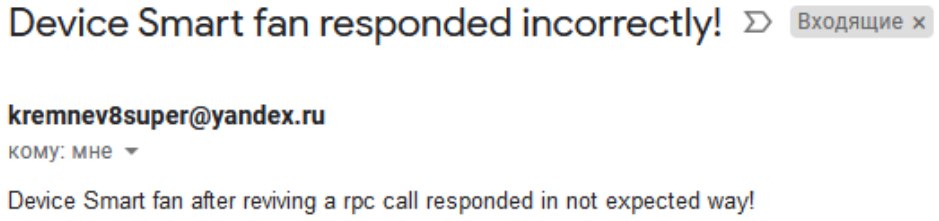


Рисунок 52 — Сообщение о некорректности отправленных параметров

Сымитировав аварийные случаи, мы убедились в работоспособности цепочки правил, корректности работы механизма email–оповещений.

7.2 Отправка Email сообщений о работе блока умных ламп

Дополним цепочку управления работой блока умных ламп таким образом, чтобы после возникновения тревог критического уровня на электронный ящик администратора платформы поступало письмо с описанием проблемы.

Узел формирования Email сообщения

Конфигурация узлов формирования Email сообщений аналогична рассмотренной в пункте 7.1.

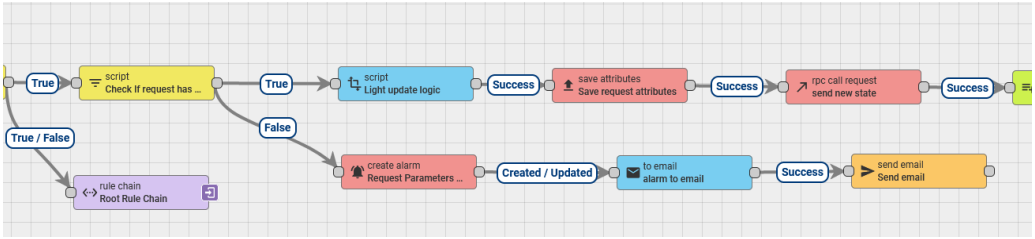
Добавим узлы в цепочку. Необходимо создать два идентичных узла формирования Email сообщения, которые следует соединить с узлами генерации тревог критического уровня.

Узел отправки Email сообщения

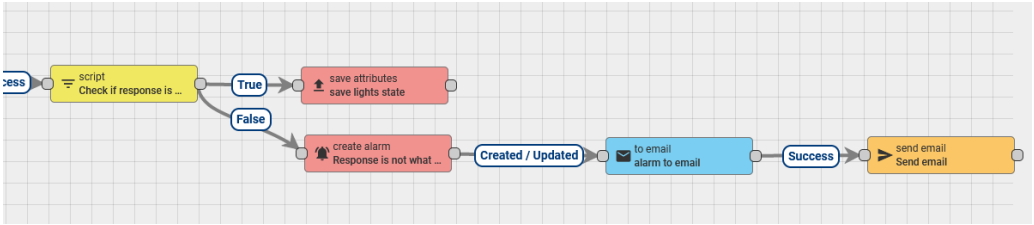
Конфигурация узлов формирования Email сообщений аналогична рассмотренной в пункте 7.1.

Два узла отправки сообщений соединим с уже добавленными узлами формирования Email сообщений.

Приведем измененную цепочку (см. Рисунок 53).



а)



б)

Рисунок 53 — Измененная цепочка управления блоком умных ламп

Тестирование созданной цепочки

Проведем тестирование созданной цепочки при помощи утилит mosquitto.

Воспроизведем действия, описанные в разделе 5.2. Впоследствии мы должны обнаружить на почте, указанной в параметрах узла отправки Email сообщения письма, содержащие информацию о возникновении тревожных ситуаций (см. Рисунки 54, 52).

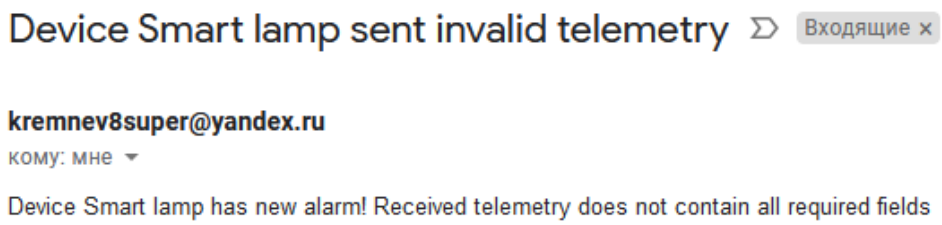


Рисунок 54 — Сообщение об отсутствии необходимых параметров

Рисунок 55 — Сообщение о некорректности отправленных параметров

Сымитировав аварийные случаи, мы убедились в работоспособности цепочки правил, корректности работы механизма email–оповещений.

8 Дополнительное задание № 12

8.1 Тестирование разрабатываемого приложения

С целью проверки работоспособности элементов разрабатываемого приложения было проведено тестирование скриптов, написанных на языке программирования Python. Для автоматизированного тестирования использовалась библиотека `pytest`. Также было проведено ручное тестирование для проверки корректности процедуры подписки на топики, регистрации пользователей и устройств.

Тестирование пользовательского интерфейса осуществлялось путем проверки реакции приложения на нелогичную последовательность команд и их некорректный ввод, а также на допущение ошибок в именах устройств и пользователей.

Было проведено тестирование подключения к системе с различных устройств, что позволило проверить компонент приложения, отвечающий за сетевое взаимодействие.

8.2 Описание процесса развертывания приложения

Для того, чтобы использовать приложение, которое было разработано нами в ходе выполнения практических работ, необходимо зарегистрироваться на облачной платформе ThingsBoard, затем использовать эту учетную запись для авторизации в системе умного дома с помощью Telegram–бота IoT SmartLight.

Для включения умных устройств в инфраструктуру Умного дома, необходимо, чтобы они были имели виртуальные устройства–клоны на облачной платформе ThingsBoard, а также были подключены к MQTT–брокеру для отправки и чтения сообщений из топиков.

Для удобства пользования код, описывающий логику работы Telegram–бота можно разместить в облачном окружении Yandex Cloud Solution. Данный сервис реализует подход serverless computing и позволяет запускать пользовательский код в виде функции в безопасном, отказоустойчивом и автоматически масштабируемом окружении без создания и обслуживания виртуальных машин.

Плюсами такого решения является возможность гибкого масштабирования, высокая доступность, возможность использовать различные языки программирования. Данное решение позволяет свести к минимуму объем ПО, которое должно присутствовать у пользователя локально.

База данных под управлением СУБД SQLite также может быть развернута на удаленной виртуальной машине с применением облачных решений.

Минусом данного решения является поминутная тарификация, зависимость стоимости аренды от требуемых вычислительных мощностей, отсутствие физического доступа к машинам, на которых развернуто программное обеспечение, а также риск того, что злоумышленники могут получить доступ к вашей конфиденциальной информации.

Выводы о проделанной работе

В результате выполнения данных практических работ нами были получены новые навыки работы с утилитами mosquitto, облачной платформой Интернета вещей ThingsBoard, программным интерфейсом Telegram API. Были получены знания, которые позволили разработать, смоделировать и создать приложение, реализующее пользовательский интерфейс для приложения, входящего в инфраструктуру Умного дома, разрабатываемого в рамках данного курса. Все полученные знания и навыки были отработаны и использованы на практике.