

Московский ордена Ленина, ордена Октябрьской Революции и ордена Трудового

Красного Знамени государственный технический университет имени Н. Э. Баумана

Факультет «Робототехника и комплексная автоматизация»

Кафедра «Компьютерные системы автоматизации производства»

«Документация по этапам №1-3 реализации кейсового задания»

ID: 254187039

05.02.1999

[dmasny@yandex.ru](mailto:dmasny@yandex.ru)

+79778781228

Бакалавриат 4 курс

Студент: Масный Дмитрий Игоревич

Группа: РК9-84Б

**Этап реализации кейса №1**

**Реализация эмуляторов устройств на платформе Rightech IoT Cloud**

Для реализации данной части задания необходимо прежде всего ознакомиться с исходными данными, предоставленными в таблице №1 задания финального тура олимпиады.

После ознакомления с материалами задания необходимо создать модель «Система управления защитным костюмом» на платформе Rightech IoT Cloud (далее просто RIC). Данная модель имеет следующие подсистемы: «Серверная информация», «Параметры костюма», «Координаты», «Окружающая среда», «Beacon», «Расстояния до маяков». После создания модели в ней создаются подсистемы, в которых в свою очередь создаются узлы (имя узла, его описание, идентификатор и тип данных указаны в таблице №1 задания). Кроме этого, стоит учитывать, что некоторые параметры подсистем генерируются ботом (для них указывается топик), который создается далее, некоторые являются конфигурационными («Емкость аккумулятора костюма», «Мощность в течение часа»), некоторые рассчитываются обработчиками («Запас заряда», расстояния до маяков и др.) (для них топик не указывается). Пример описания конфигурационного параметра представлен на рисунке 1.

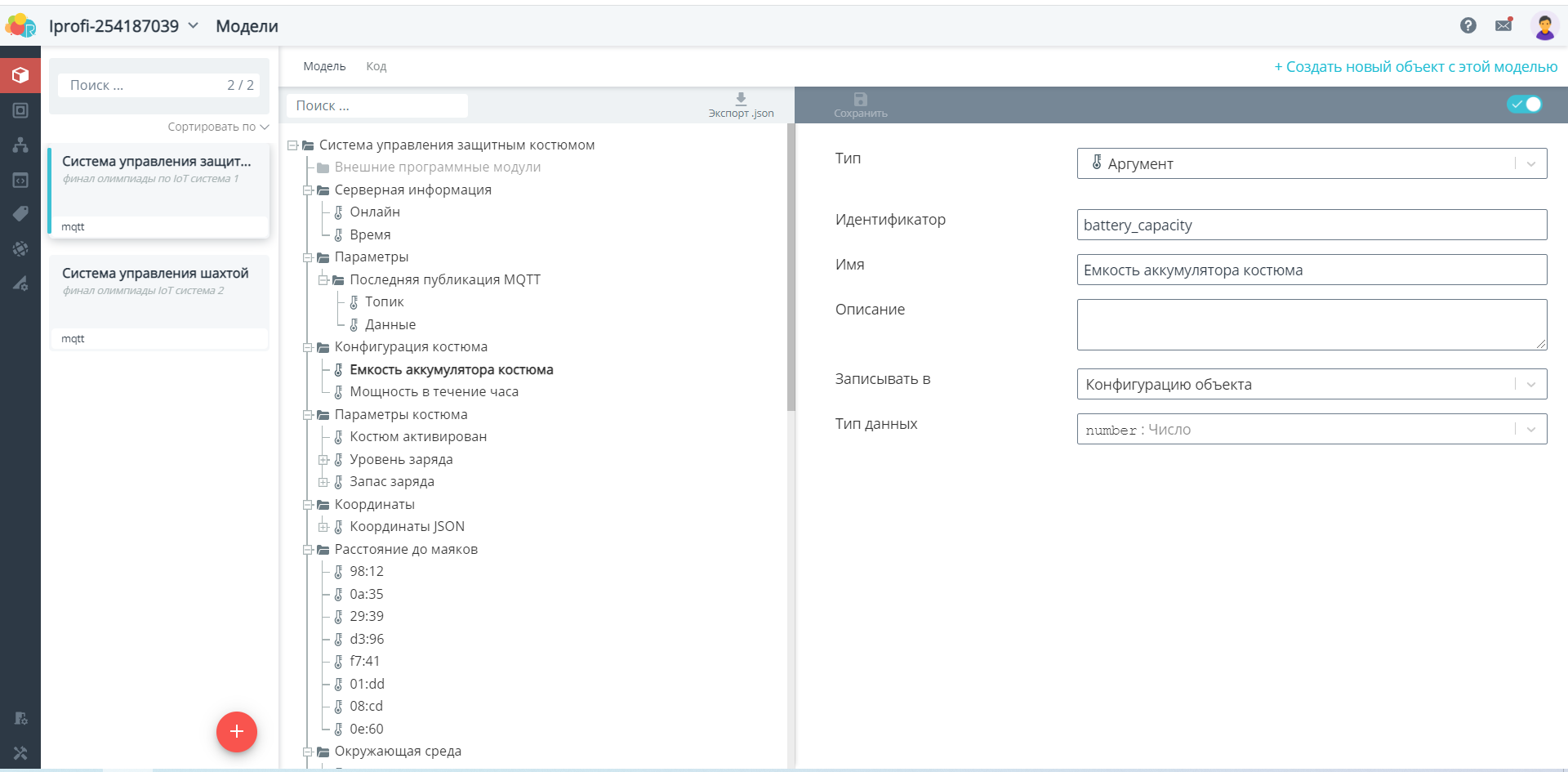


Рисунок 1

Кроме этого, в указанной модели создаются две команды «Активировать костюм», «Деактивировать костюм». Описание команды представлено на рисунке 2.

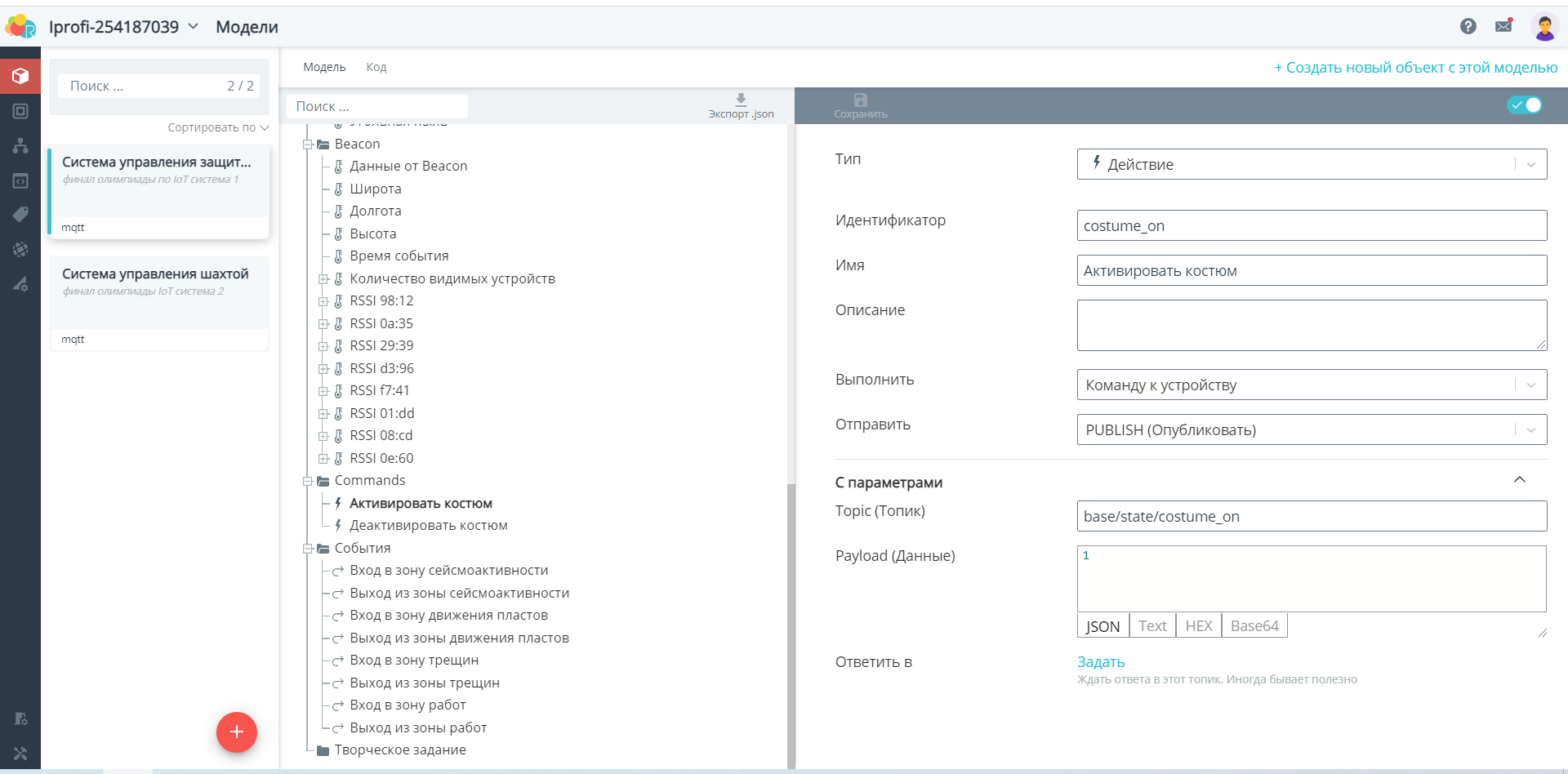


Рисунок 2

Для аргументов, в которых необходимо указать топик, топик выбирается согласно следующему паттерну: base/state/name, где name – идентификатор текущего узла. Результат создания модели «Система управления защитным костюмом» и ее подсистем представлена на рисунке 3. Поскольку модель объемная, более детально рассмотреть ее можно будет на видеофрагменте, ссылка на которой будет приложена далее. Кроме того, я прикреплю .json файл каждой модели.

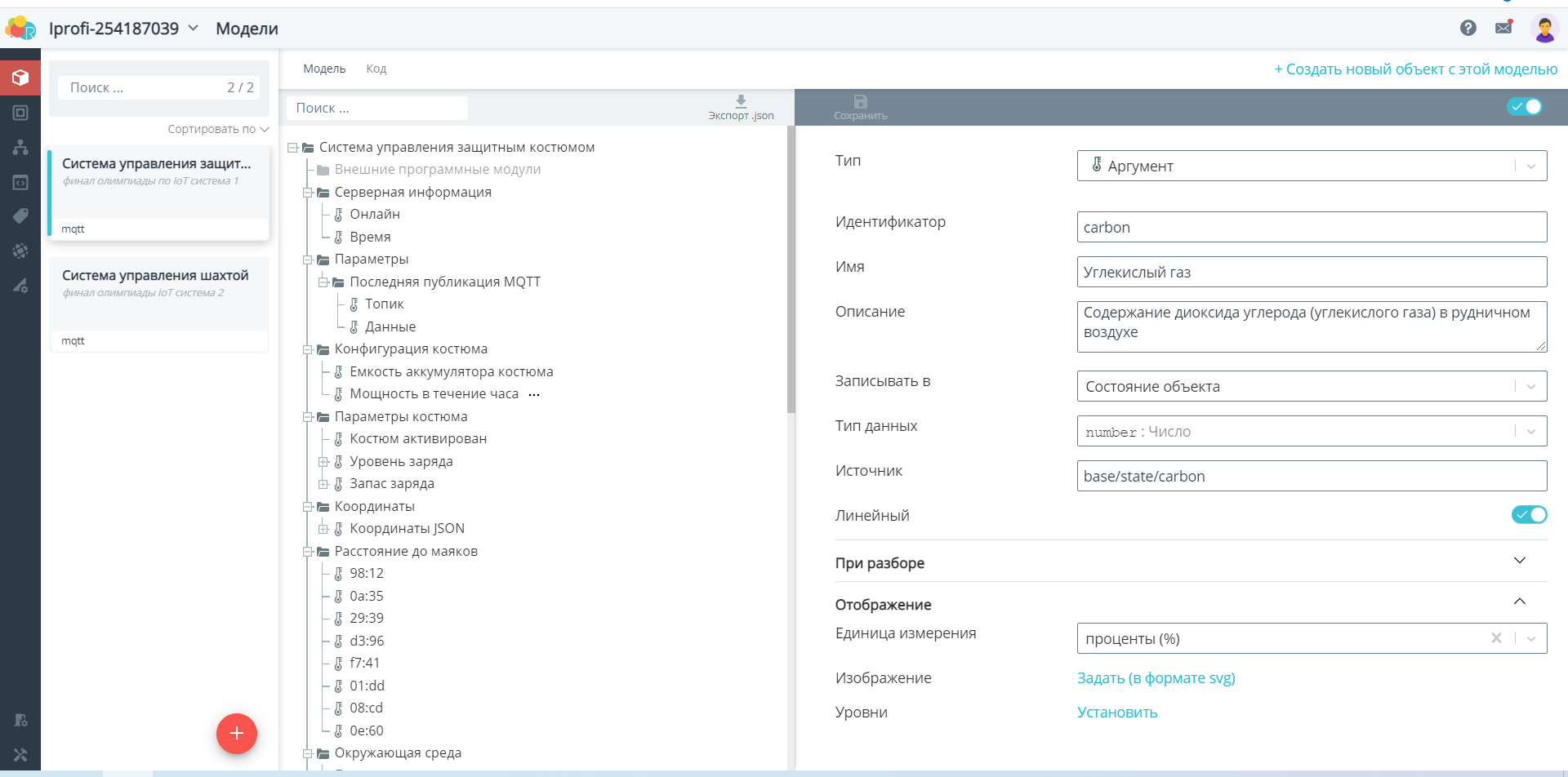


Рисунок 3

Дополнительно создается подсистема «Творческое задание», которая будет использоваться при решении кейса №4 и подсистема «События», которая используется при решении кейса №3.

После этапов, описанных выше, создается модель «Система управления шахтой» на основе исходных данных, указанных в таблице №2 задания олимпиады. Так же, как и в первом случае, необходимо создать подсистемы, узлы, конфигурационные параметры и команды. Поскольку эти этапы аналогичны предыдущим, приводится только один скриншот уже готовой модели (рисунок 4).

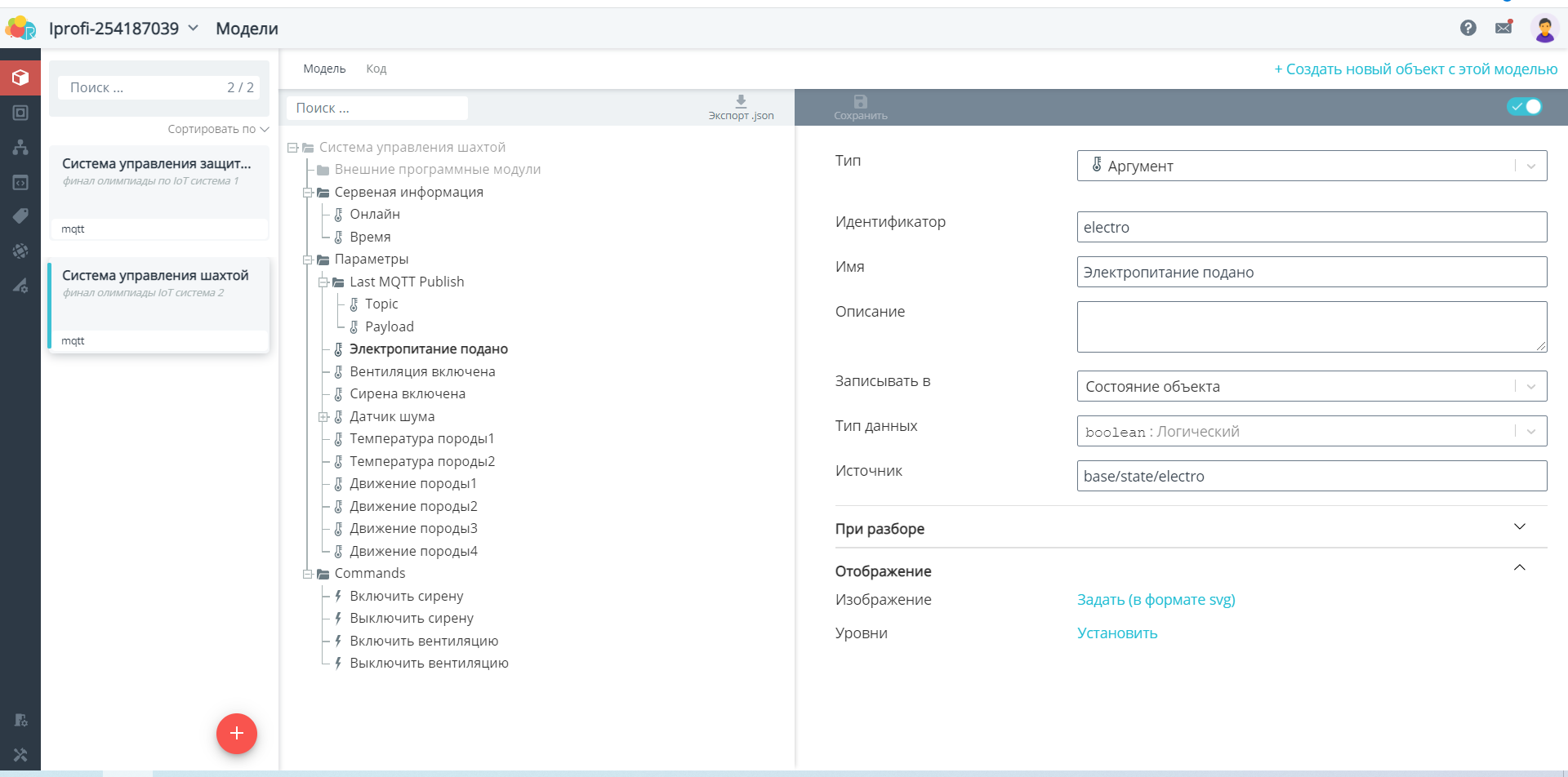


Рисунок 4

После указанных действий создаются объекты для каждой из моделей. Результат приведен на рисунке 5.

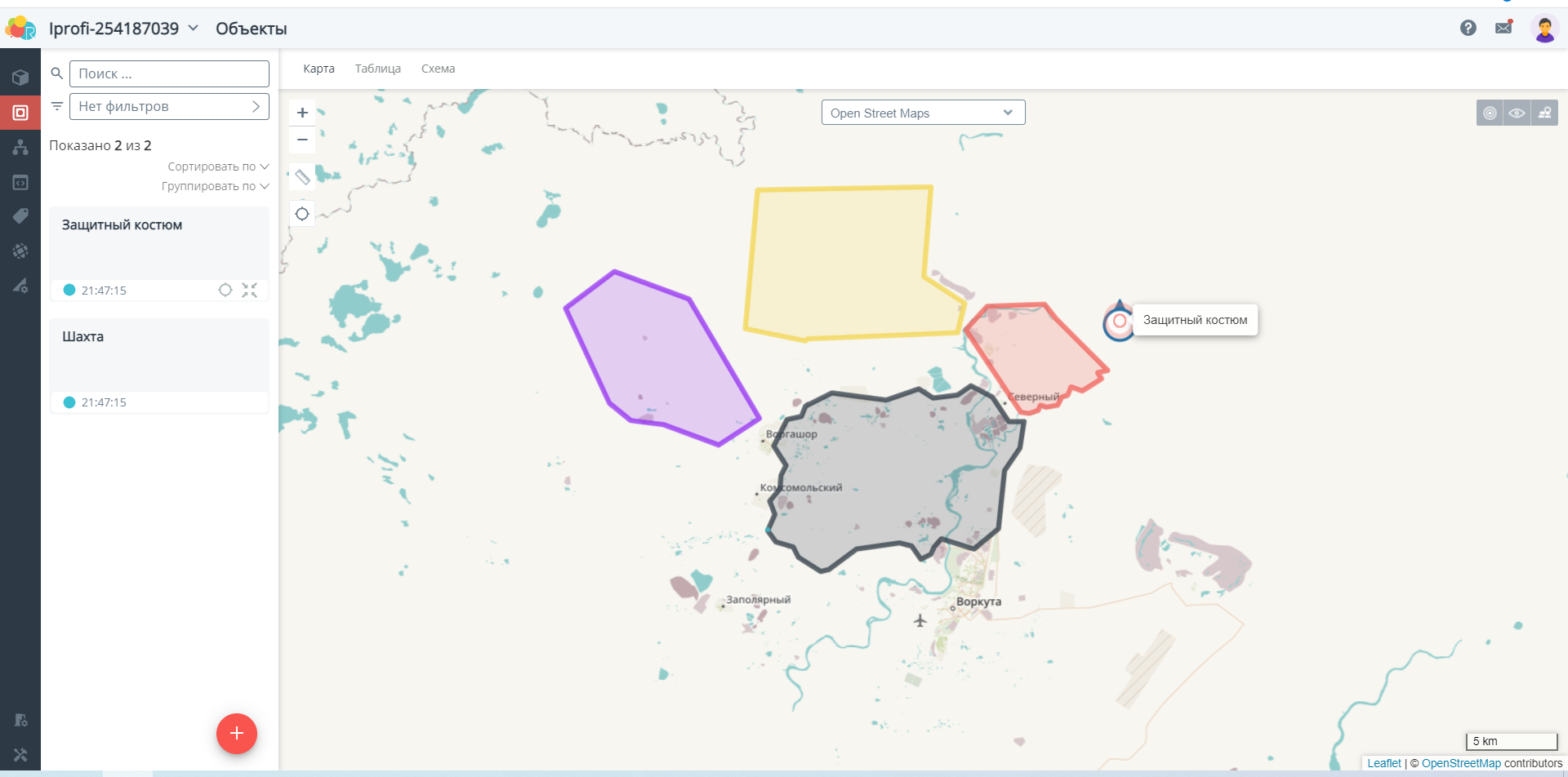


Рисунок 5

Для того, чтобы эмулировать некоторые данные, необходимо настроить объект в качестве бота (рисунок 6).

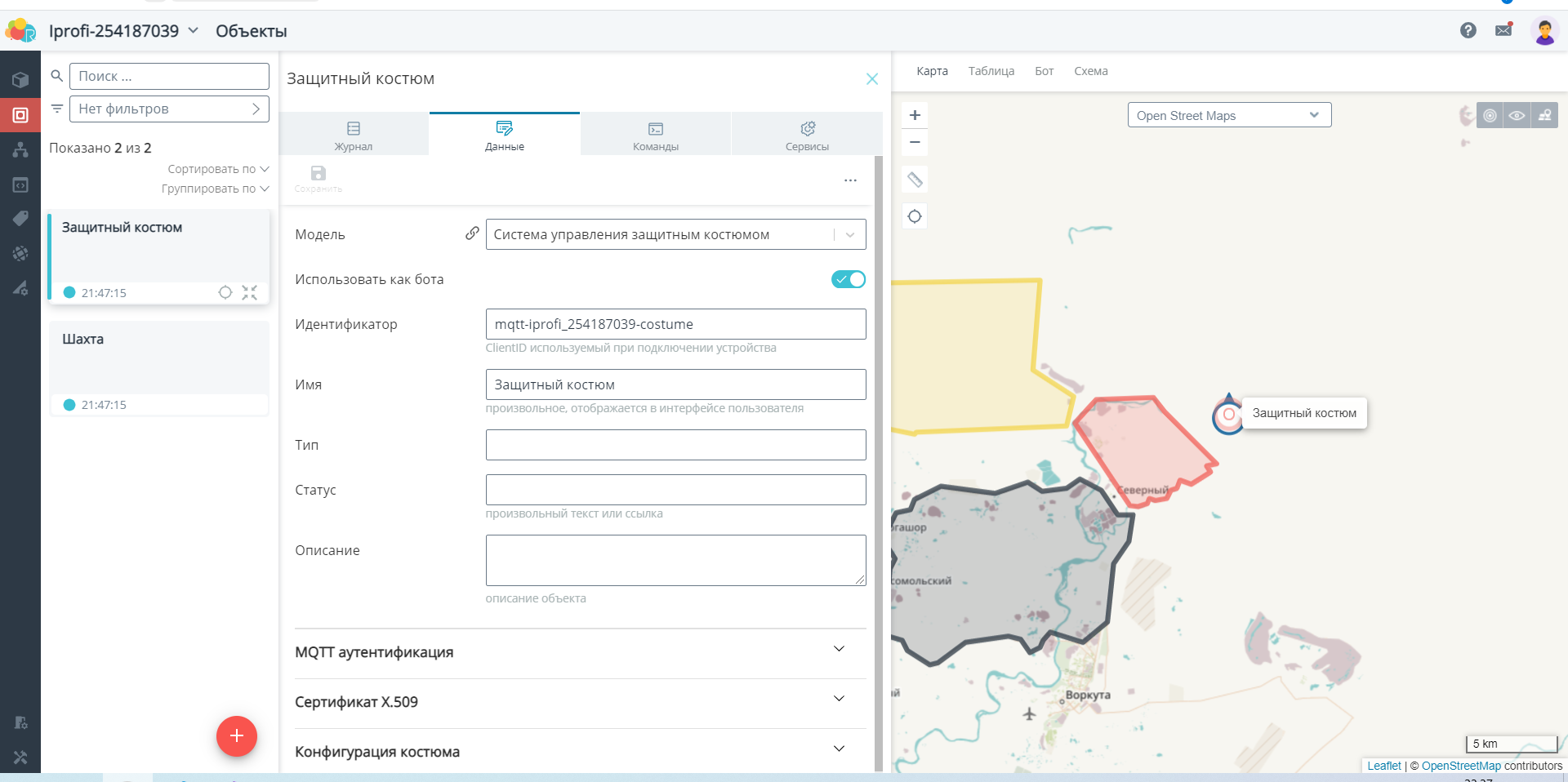


Рисунок 6

Результат создания объекта «Защитный костюм» на основе модели «Система управления защитным костюмом» представлен на рисунке 7.

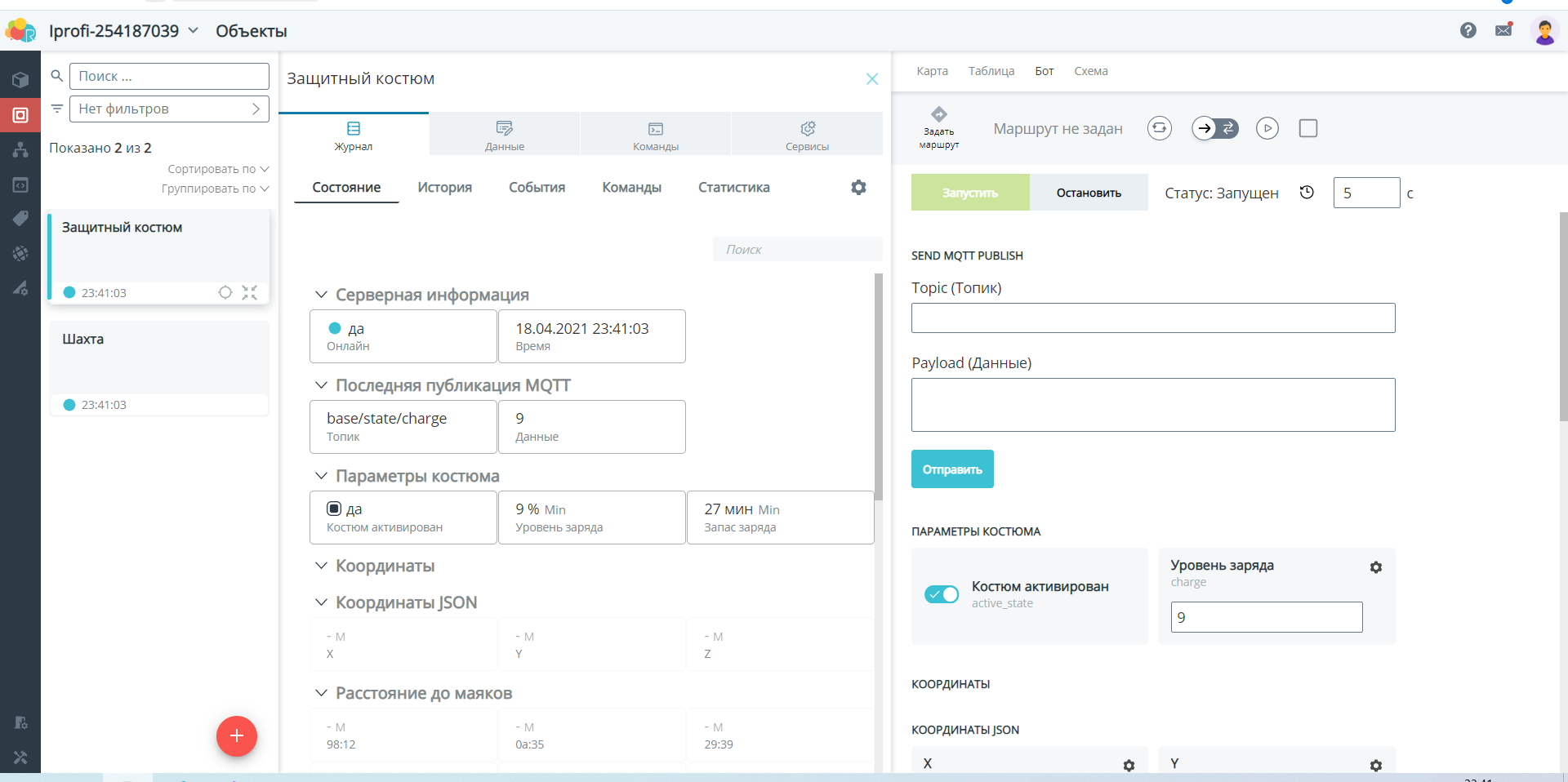


Рисунок 7

Результат создания объекта «Шахта» на основе модели «Система управления шахтой» представлен на рисунке 8.

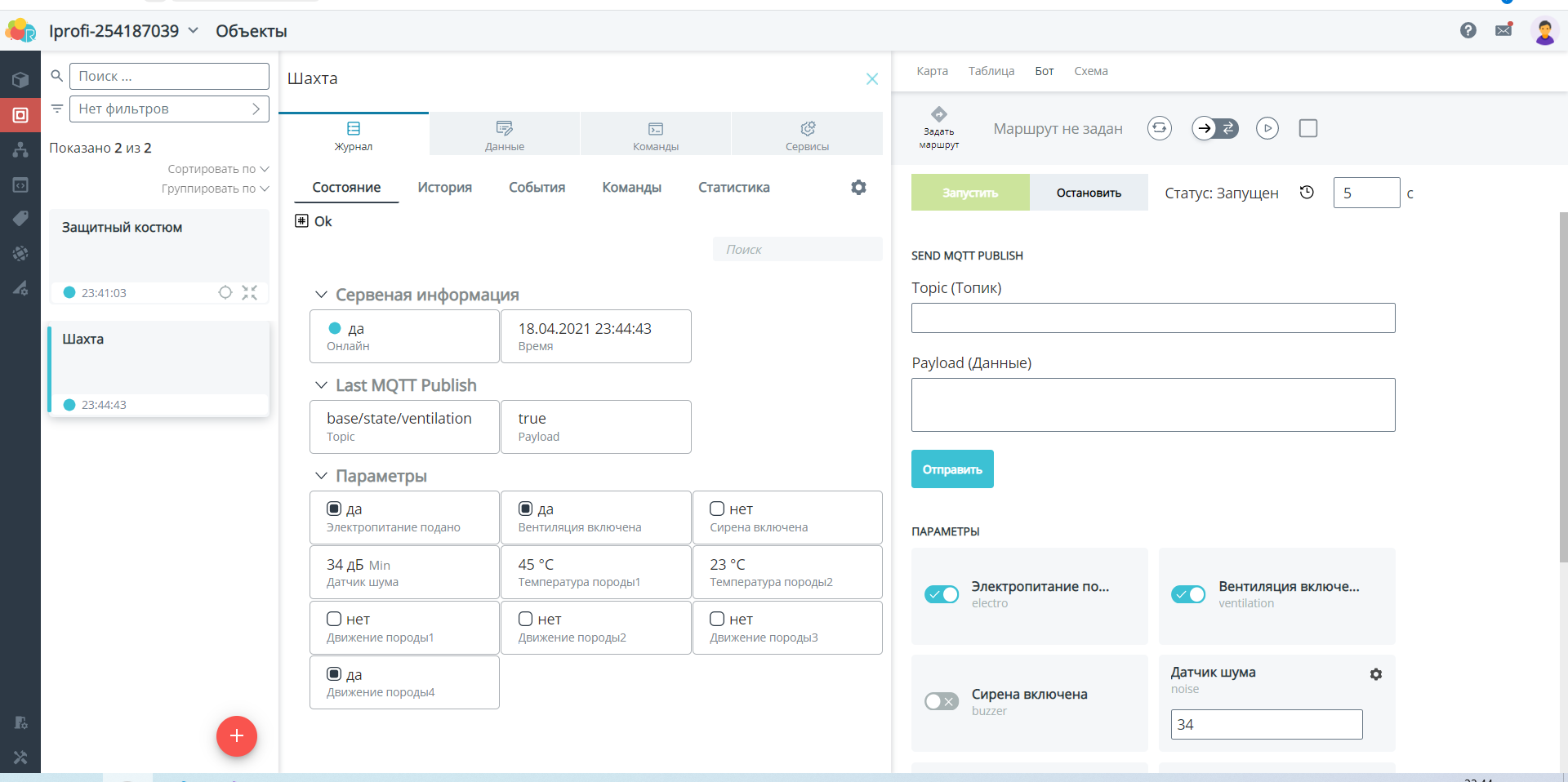


Рисунок 8

Более детальный разбор решения этапа №1 представлен в видеоматериале, расположенном по ссылке: <https://www.youtube.com/watch?v=t892GFQQj4c>

**Этап реализации кейса №2**

**Постобработка полученных данных**

В данном этапе финального задания олимпиады необходимо создать три обработчика. Для пояснения работы каждого из них приводятся блок-схемы алгоритмов работы программы, скриншоты исходного кода и результата выполнения каждого из обработчиков, а также видеоматериал с комментариями.

**Обработчик №1**

**Расчет запаса заряда аккумулятора защитного костюма**

Для написания исходного кода обработчика необходимо перейти во вкладку «Обработчики» на платформе RIC. Алгоритм работы обработчика №1 представлен на рисунке 9.

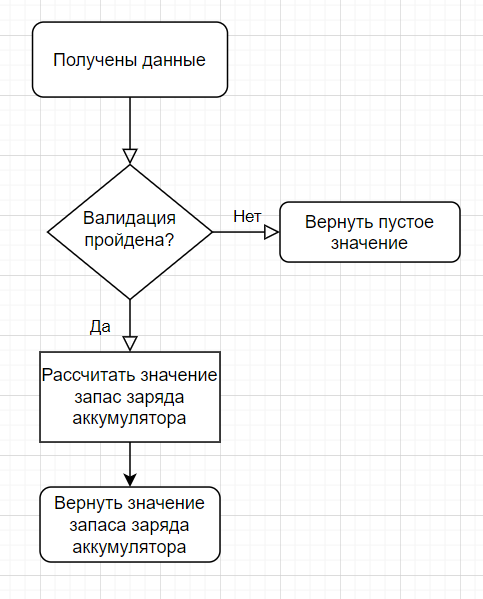


Рисунок 9

Исходных код программы обработчика представлен на рисунке 10.

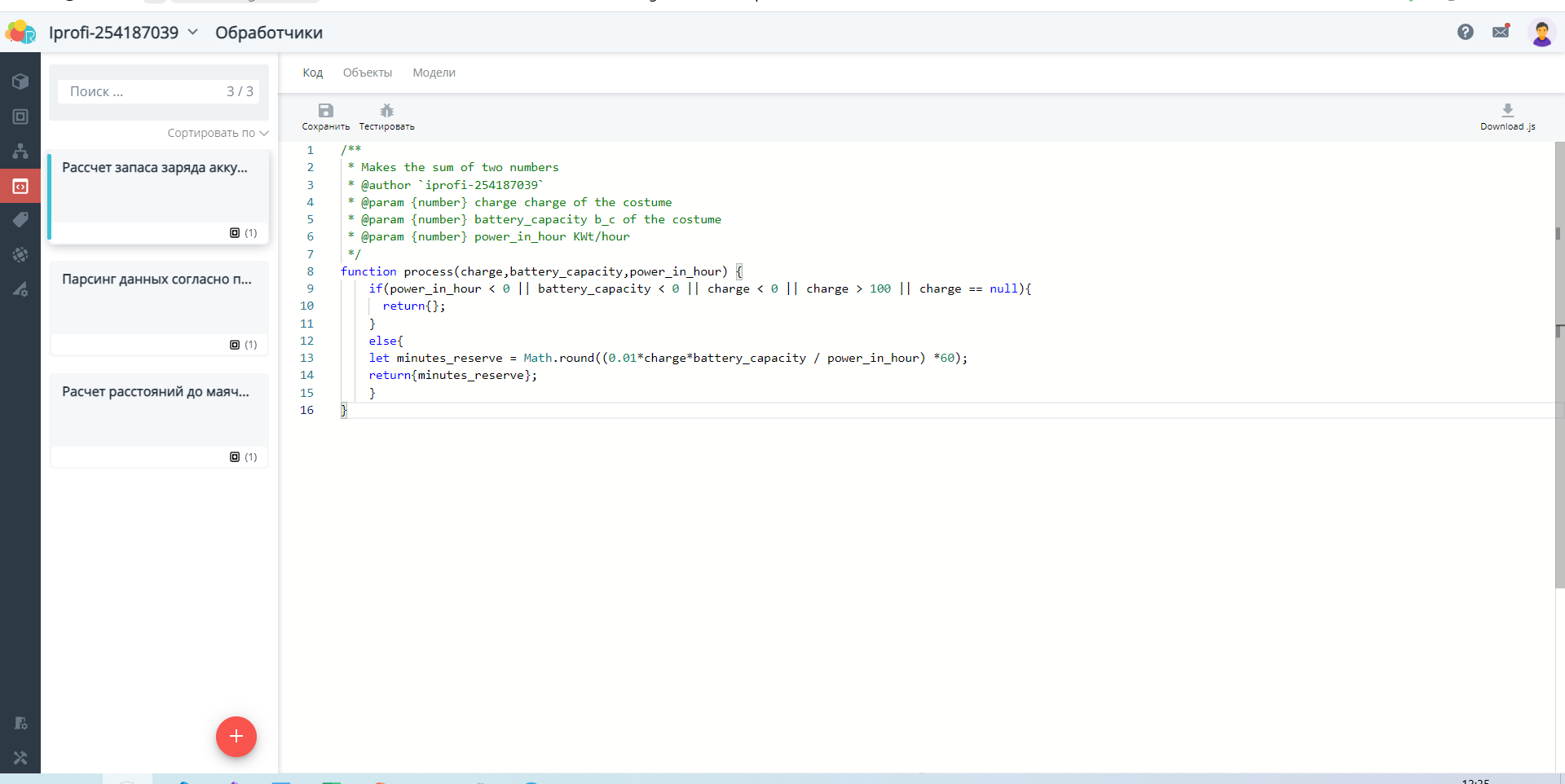


Рисунок 10

На вход обработчика подаются параметры: заряд аккумулятора (charge) (эмулируется ботом), емкость аккумулятора костюма (battery\_capacity) (конфигурационный параметр), мощность в течение часа (power\_in\_hour) (конфигурационный параметр). Далее в блоке If выполняется валидация полученных данных (если мощность в течение часа меньше нуля или емкость аккумулятора костюма меньше нуля или заряд аккумулятора меньше нуля или заряд аккумулятора больше 100 или заряд аккумулятора равен null). Если одно из этих условий выполнено, обработчик вернет null. Если ни одно из условий не выполняется, то при выполнении программы данный блок кода игнорируется, и выполняется блок кода else. В данном блоке создается новая переменная типа Number, значение которой равно результату выполнения функции round() модуля Math). Данная функция принимает в качестве аргумента результат выполнения операции: . Множитель 0.01 необходим для того, чтобы перевести значение заряда аккумулятора из процентов в доли от единицы, а множитель 60 нужен для того, чтобы перевести полученный результат из часов в минуты. Далее применяя функцию Math.round() полученное значение округляется до целых.

Далее необходимо связать существующий объект с обработчиком (рисунок 11), после чего нужно связать параметры объекта, с аргументами функции обработчика (рисунок 12).

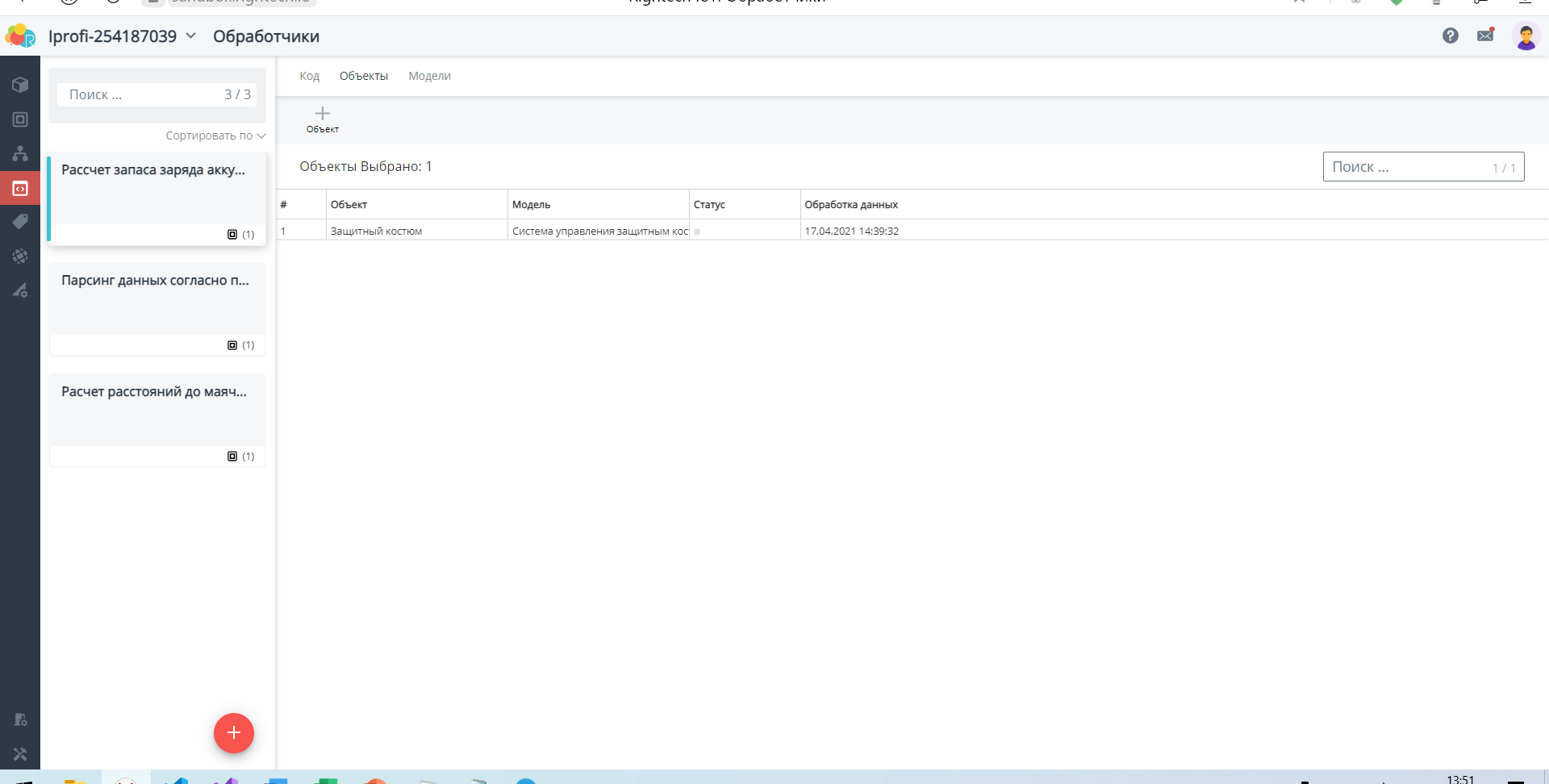


Рисунок 11

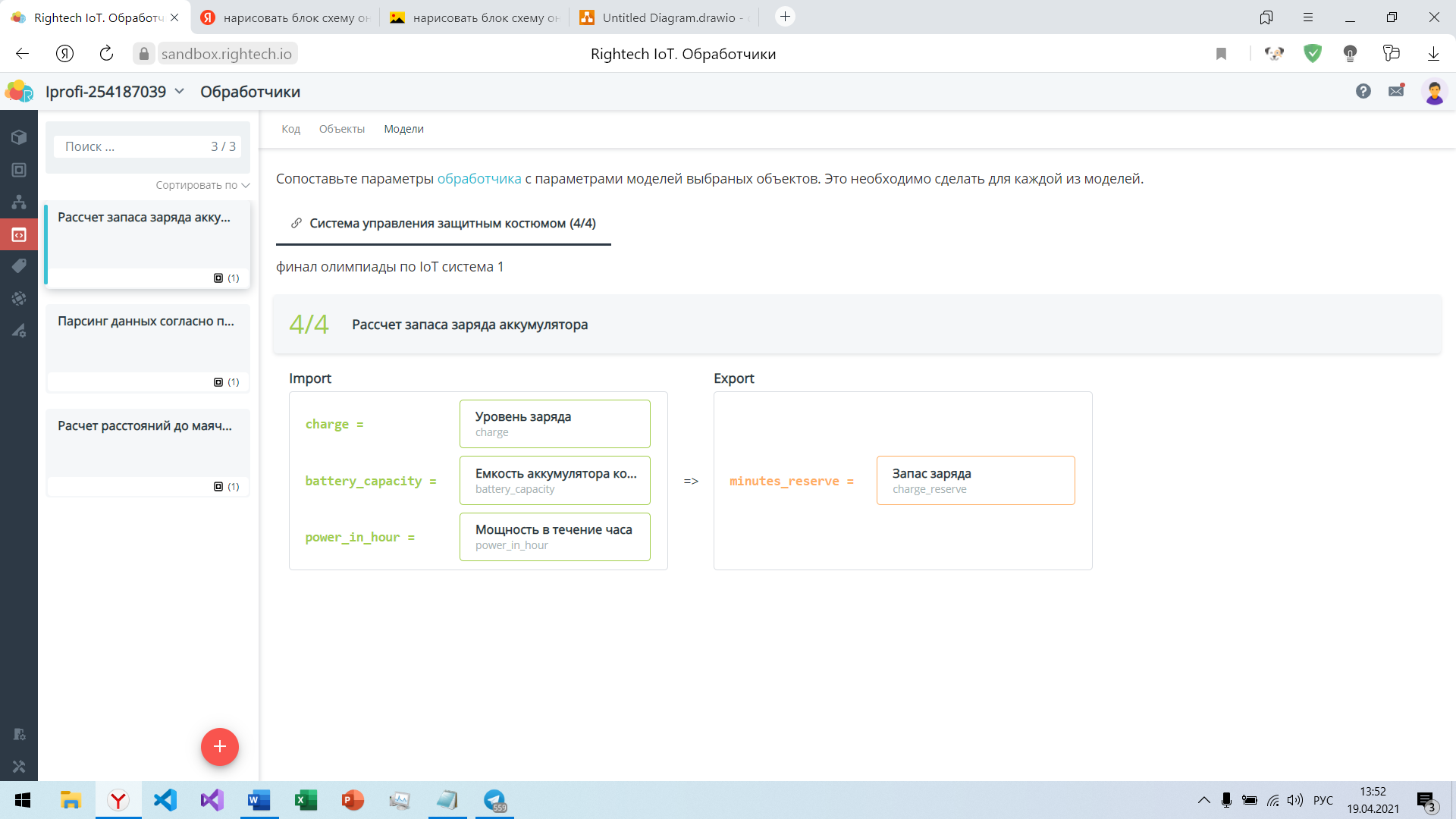


Рисунок 12

Результат работы обработчика №1 представлен на рисунке 13.

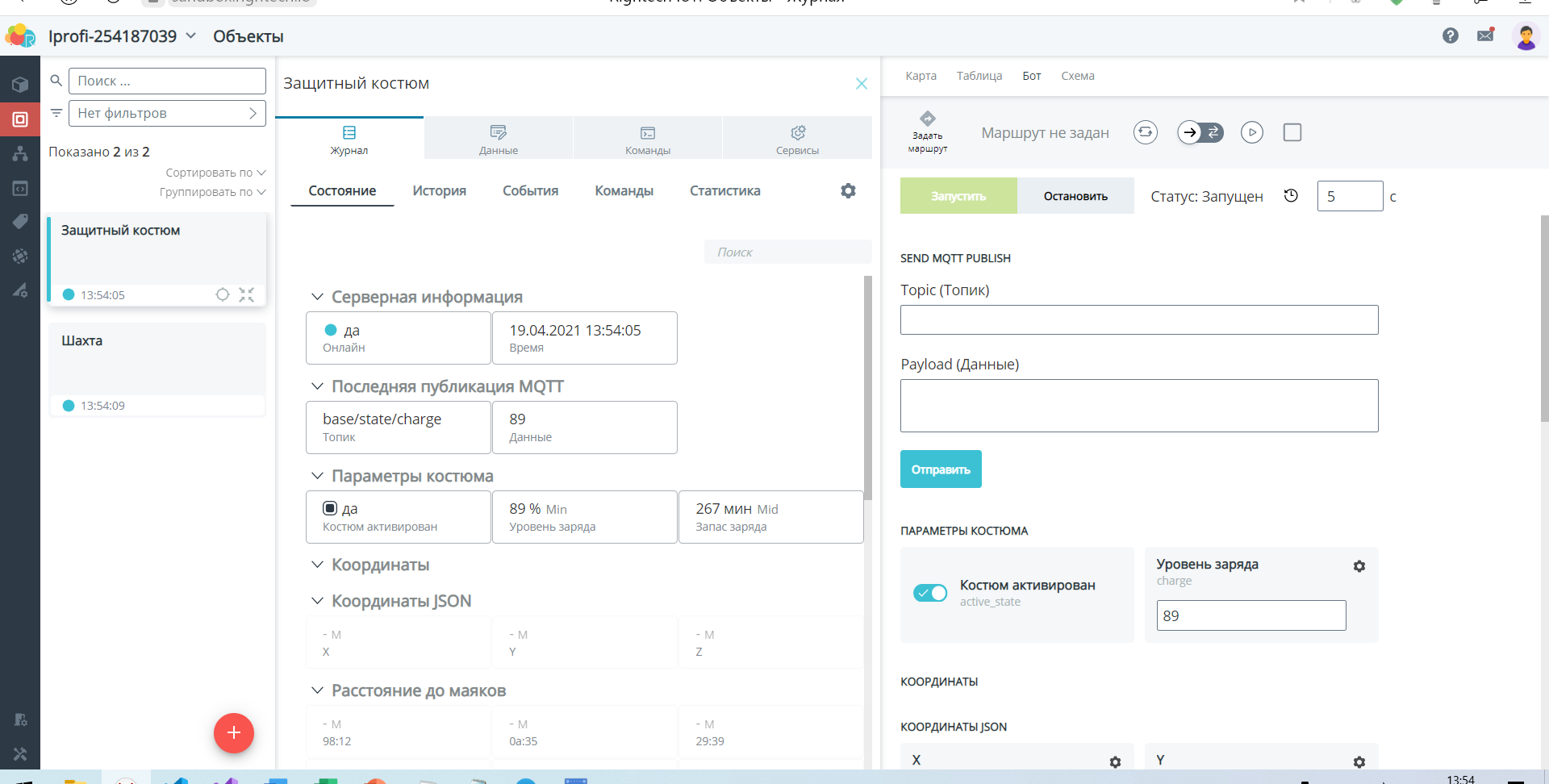


Рисунок 13

**Обработчик №2**

**Парсинг данных согласно протоколу**

Для выполнения данного задания необходимо так же перейти во вкладку обработчики и создать новый обработчик. Упрощенный алгоритм работы обработчика №2 представлен на рисунке 14.

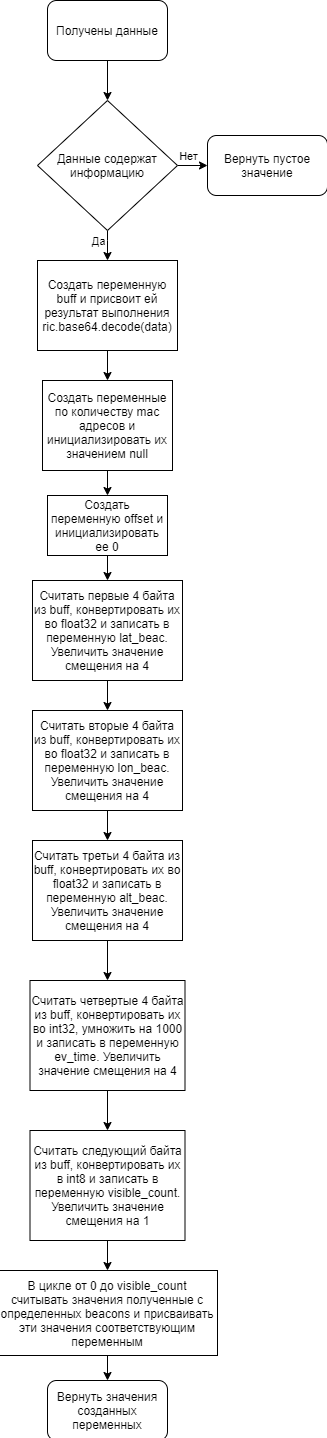


Рисунок 14

Исходный код обработчика №2 представлен на рисунке 16, рисунке 16 и рисунке 17.

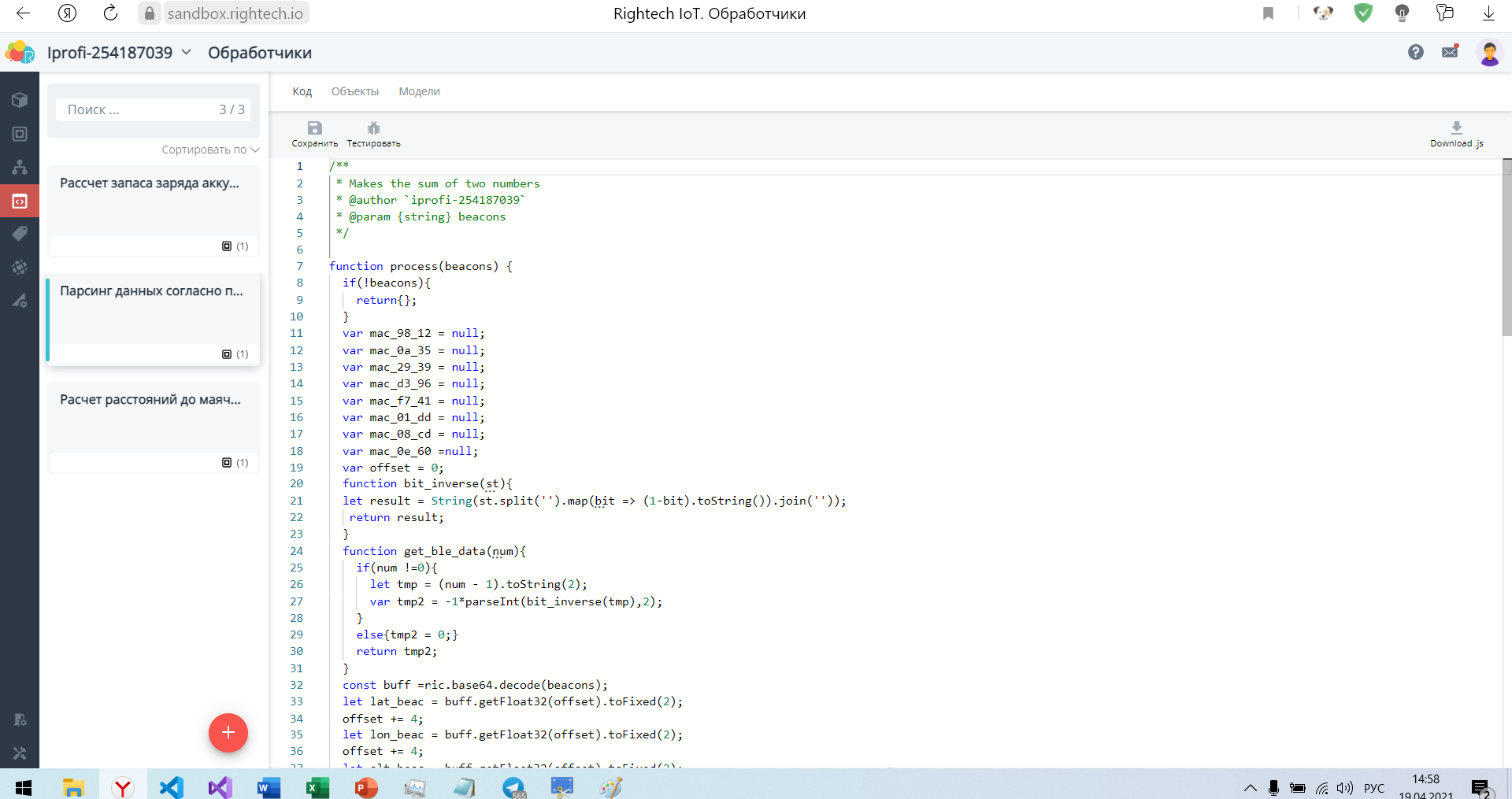


Рисунок 15

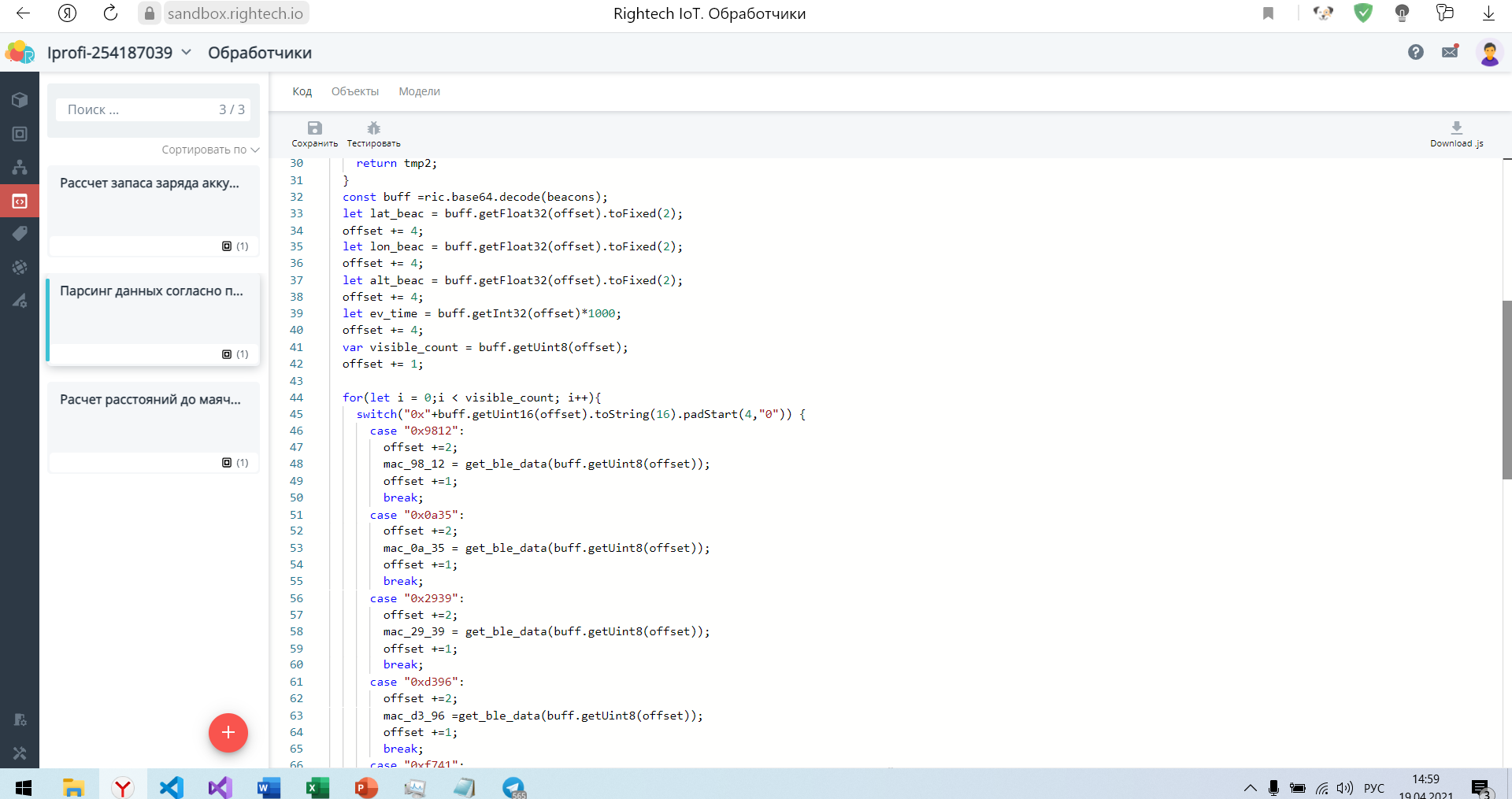


Рисунок 16



Рисунок 17

Сначала полученные данные проходят валидацию: если получена пустая строка, то обработчик вернет null. Если строка содержит данные, то создаются переменные по количеству mac адресов beacon маяков и инициализируются null. Это сделано для того, чтобы обработчик не сохранял прошлые данные для каких либо маяков (более подробно поясню в видеоматериале).Далее создается переменная offset (смещение) и инициализируется 0. После этого создается переменная buff в которую записывается результат выполнения функции ric.base64.decode(data). После считывается 4 байта из переменной buff, конвертируется во float32, округляется до 2ух знаков после запятой и записывается в переменную. Далее увеличивается смещение на 4. Таким образом, записываем значения широты, долготы, высоты, полученные из строки base64 в переменные lat\_beac, lon\_beac, alt\_beac. Аналогично получаем значение для ev\_time, за исключением того, что данные конвертируются в int32 и умножаются на 1000. Умножение на 1000 необходимо для того, чтобы значение ev\_time было записано в микросекундах и на платформе RIC отображалось корректно. Далее смещение увеличивается на 1 и считывается байт отвечающий за число видимых маяков. Считанное значение конвертируется в int8 и записывается в переменную visible\_count. Далее данные попадают в цикл, число итераций которого зависит от значения visible\_count. Далее данные попадают в блок switch case в зависимости от значения полученного в результате конвертации данных из переменной buff ("0x"+buff.getUint16(offset).toString(16).padStart(4,"0")) (в этом блоке кода сначала данные из buff преобразовываются в uint16, далее преобразовываются в hex (после этой операции тип данных – строка), после чего к ним конкатинируется 0 слева,таким образом, чтобы в строке было 4 символа, после этого к строке конкатинируется 0х слева (более подробное объяснение в видеоматериале)). Далее попадая в один из блоков case происходит увеличение смещения на 2, после чего происходит вызов функции get\_ble\_data(), которая возвращает тип int – значение сигнала с определенного маяка, после чего происходит увеличение смещения на 1.

Функция get\_ble\_data() не является библиотечной и разрабатывается отдельно. Она необходима для того, чтобы корректно раскодировать отрицательное значение сигнала, переданное с маяка (по условию значения уровня сигнала от -127 до 0). Функция на вход принимает число, полученное в результате выполнения кода buff.getUint8(offset). Если полученное число не 0, то из него вычитается 1 и оно переводится в двоичный вид. Далее блоке кода  var tmp2 = -1\*parseInt(bit\_inverse(tmp),2) вызывается функция bit\_inverse, которая возвращает строку с инвертированными битам. После чего эта строка преобразуется в тип int из двоичного вида и умножается на -1. В результате чего получаем корректное отображение отрицательного числа.

Функция bit\_inverse() работает следующим образом: на вход она принимает строку битов, далее создается переменная result в которую записывается результат String(st.split('').map(bit => (1-bit).toString()).join('')). Здесь сначала переданная строка разбивается на подстроки длиной в символ, далее создается массив, в который записывается результат применения функции инвертирования для каждой подстроки. Инвертирование происходит тривиально – из 1 вычитается бит, который инвертируется. Далее полученный массив элементов конкатинируется и в явном виде полученный результат приводится к типу string. Более подробно работа обработчика описана в видеоматериале.

Далее обработчик связывается с объектом (рисунок 18) и параметры объекта связываются с аргументами обработчика (рисунок 19).

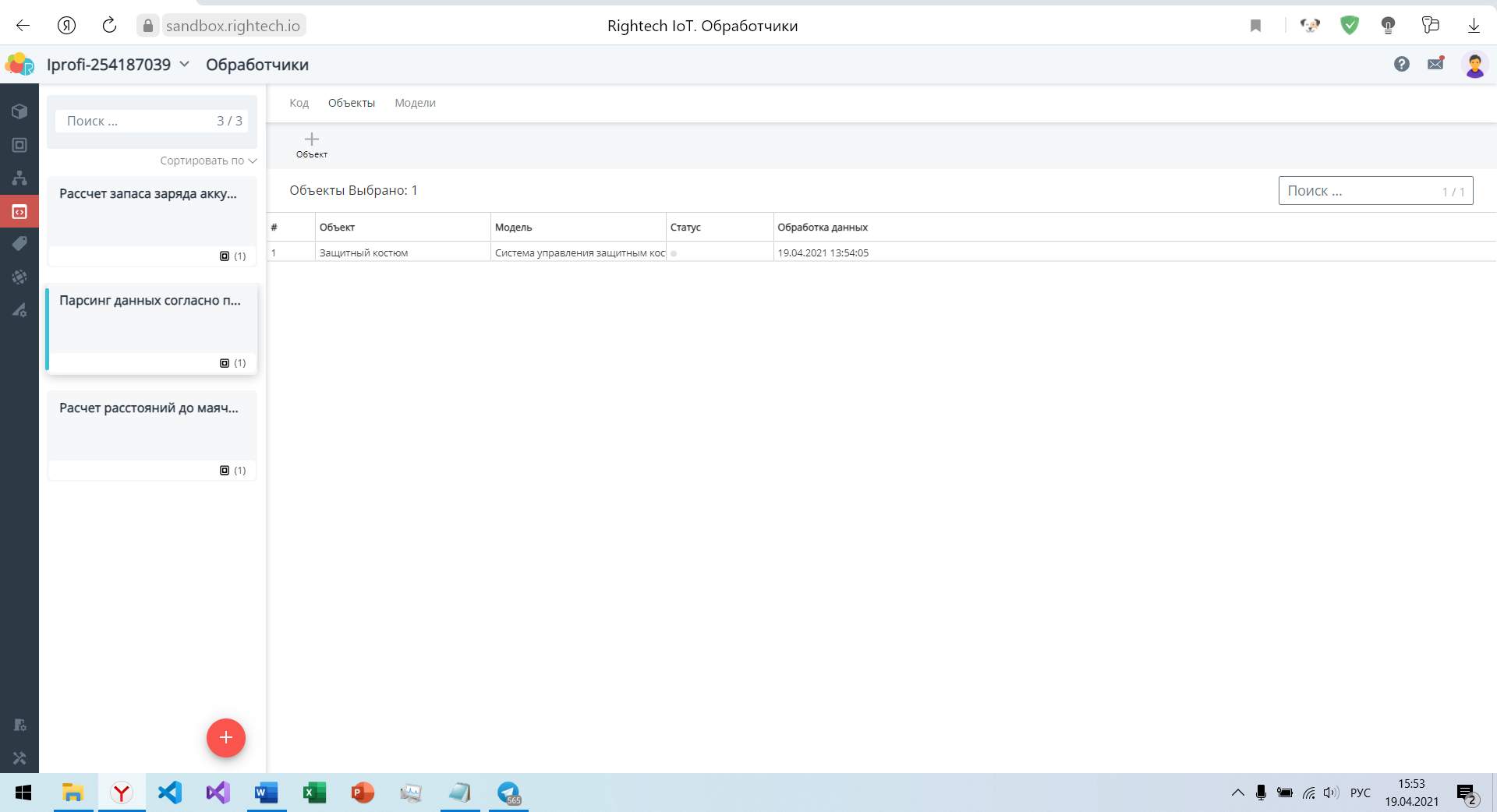


Рисунок 18

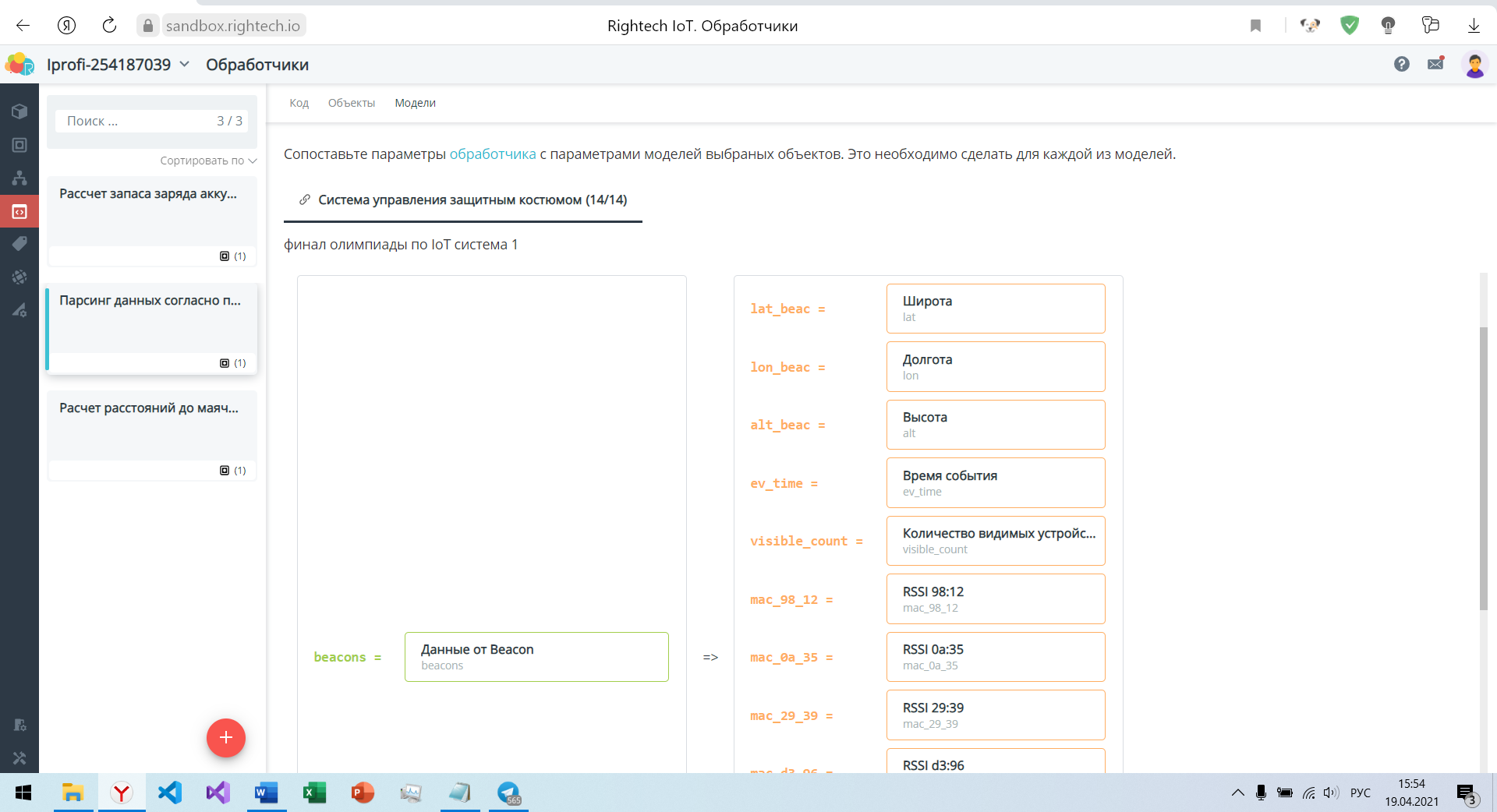


Рисунок 19

Стоит отметить алгоритм формирования строки base64. Сначала все данные преобразуются из типов, указанный в таблице №3 задания олимпиады в hex, и только потом преобразуются в base64.

Пример данных для тестирования обработчика №2 представлен в таблице №1

Таблица 1

|  |  |  |
| --- | --- | --- |
| Название данных | Данные в исходном представлении | Данные в hex |
| Широта | 23,54 | 41 bc 51 ec |
| Долгота | 34,89 | 42 0b 8f 5c |
| Высота | 12,4 | 41 46 66 66 |
| Время фиксации данных | 1618843138 | 60 7D 96 02 |
| Количество видимых BLE | 3 | 03 |
| Mac адрес 1 видимого маяка | 98 12 | 98 12 |
| Уровень сигнала первого видимого маяка | -125 | 83 |
| Мак адрес 2 видимого маяка | 29 39 | 29 39 |
| Уровень сигнала второго видимого маяка | -100 | 9C |
| Мак адрес 3 видимого маяка | F7 41 | F7 41 |
| Уровень сигнала 3 видимого маяка | 0 | 00 |

Для конвертации данных в hex и base64 используются онлайн конвертеры.

Строка данных в hex: 41 bc 51 ec 42 0b 8f 5c 41 46 66 66 60 7D 96 02 03 98 12 83 29 39 9C F7 41 00

Строка данных в base64: QbxR7EILj1xBRmZmYH2WAgOYEoMpOZz3QQA=

Результат работы обработчика №2 представлен на рисунке 20.

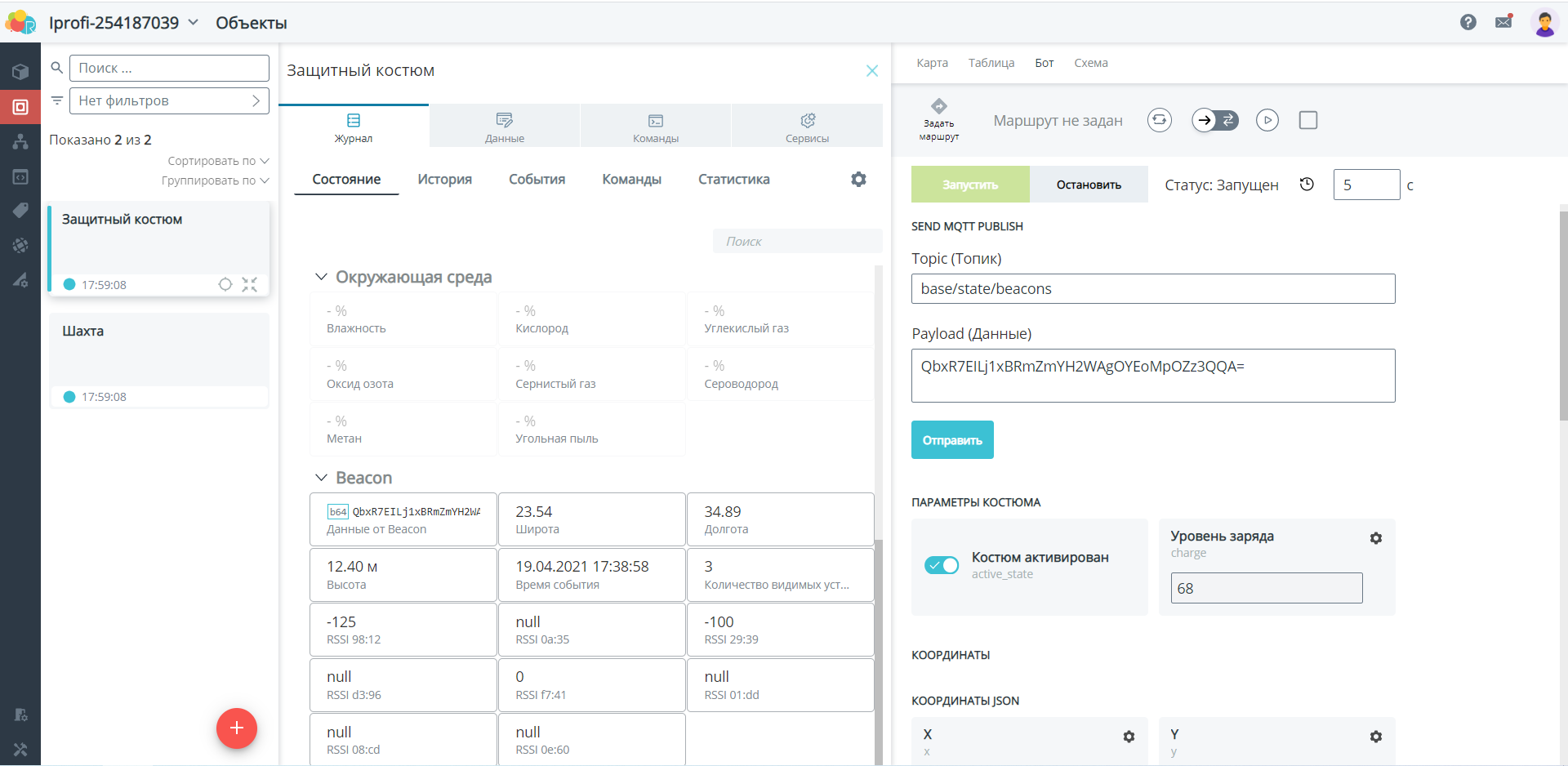


Рисунок 20

**Обработчик №3**

**Расчет расстояний до маячков**

Для выполнения данного задания необходимо так же перейти во вкладку обработчики и создать новый обработчик. Алгоритм работы обработчика №3 представлен на рисунке 21.

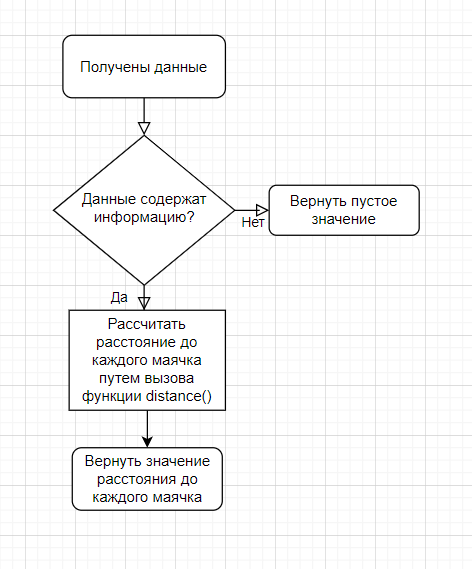


Рисунок 21

Исходный код обработчика №3 представлен на рисунке 22, рисунке 23 и рисунке 24.



Рисунок 22

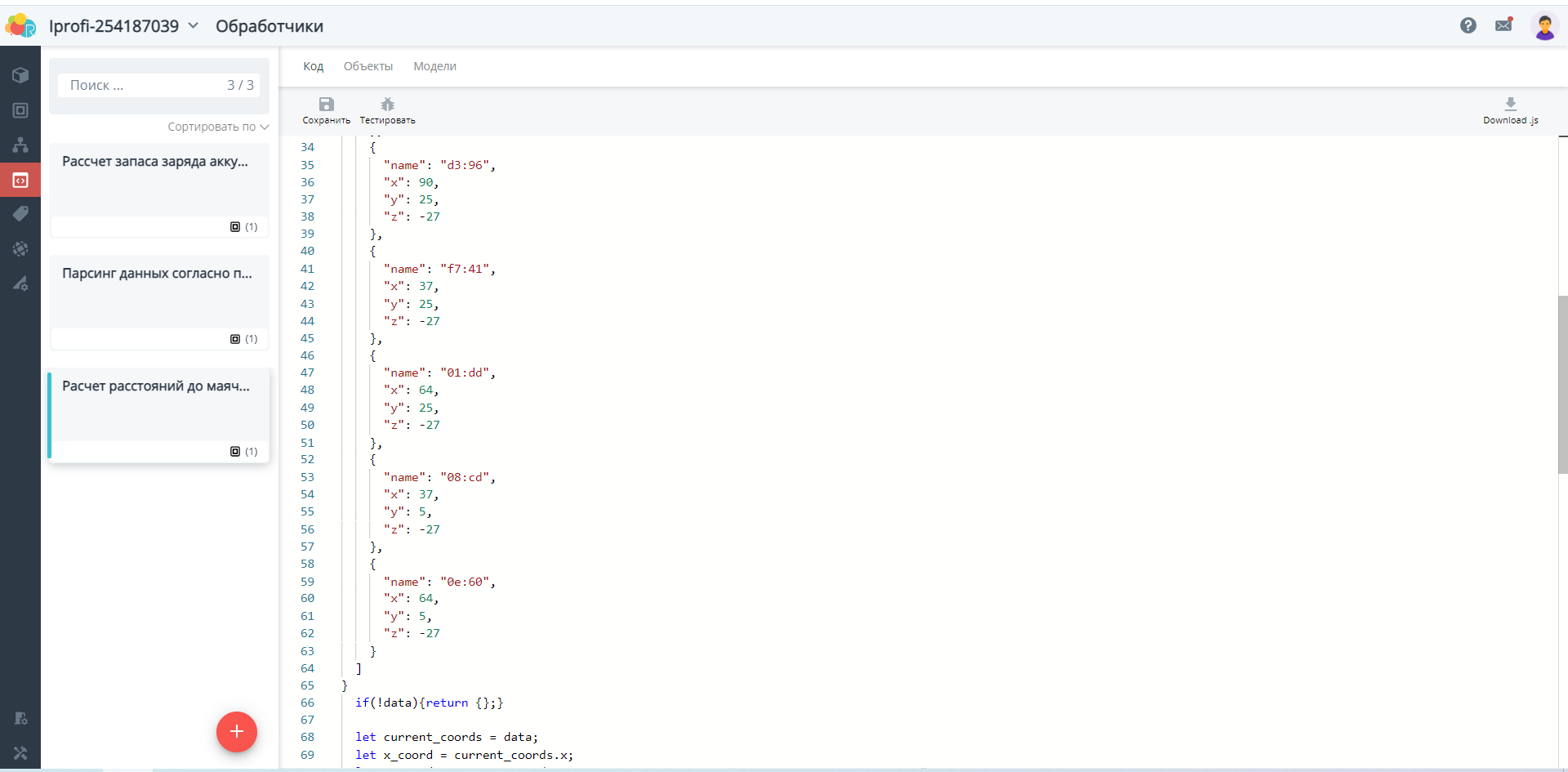


Рисунок 23

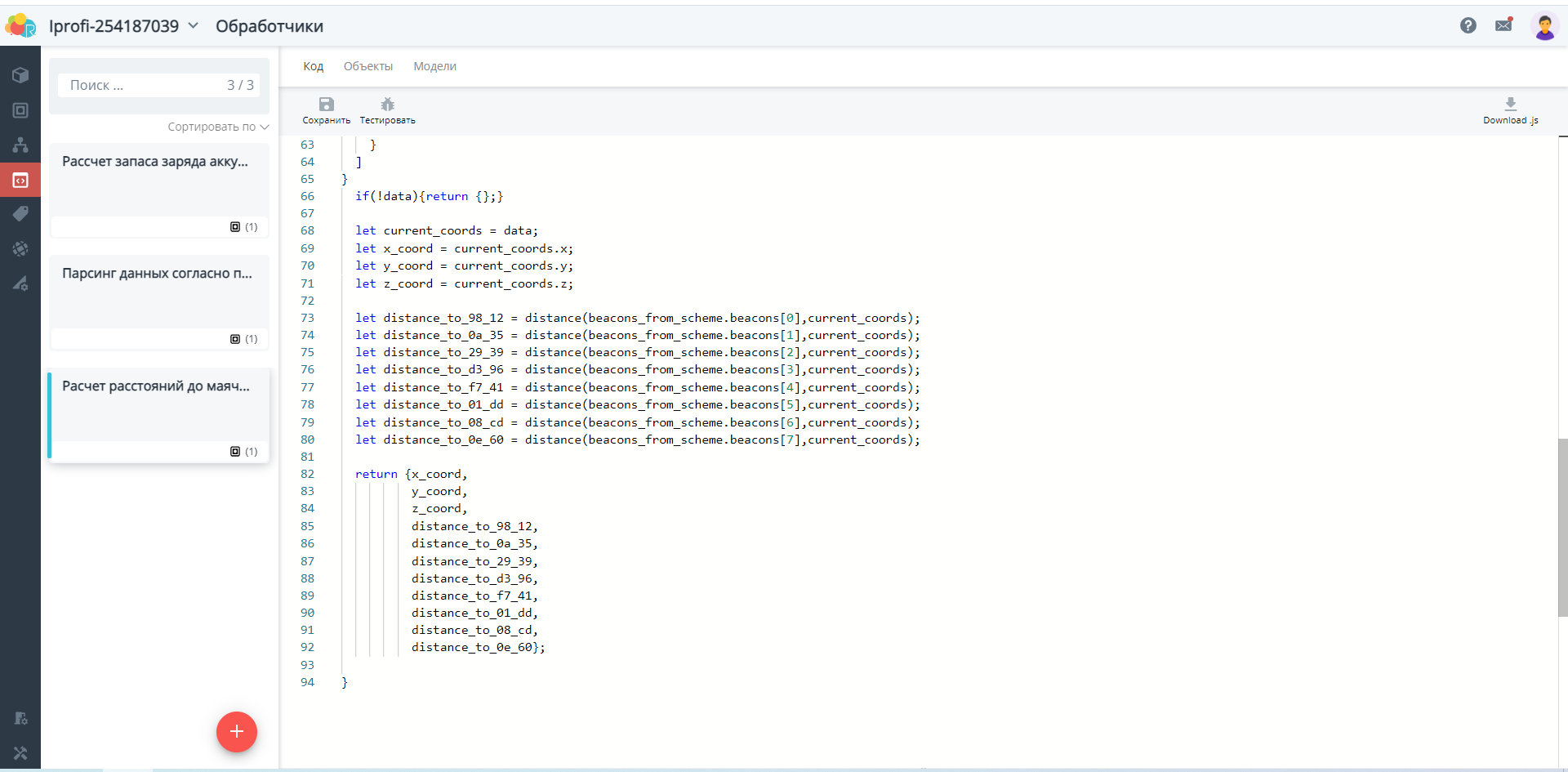


Рисунок 24

Сначала проверяются полученные данные, если была получена пустая строка, то обработчик возвращает null. Далее создается переменная current\_coords из полученного JSON объекта. На основе этой переменной создаются x\_coords, y\_coords, z\_coords. Далее вызывается функция distance() для каждой переменной, описывающей расстояние до какого-либо маяка.

Функция distance() работает следующим образом: на вход она принимает два объекта JSON. Первый объект создается при запуске функции process – данные для его создания выгружаются из вкладки маяка геозоны «Зона отслеживания». Второй объект current\_coords создается на основе входных данных. Далее расчет производится согласно теореме Пифагора в пространстве. Для этого используются функции Math.pow() и Math.sqrt().

Результат работы обработчика №3 представлен на рисунке 25. Видеоматериал по данной части задания расположен по ссылке: <https://www.youtube.com/watch?v=icyJFjX6EwA>

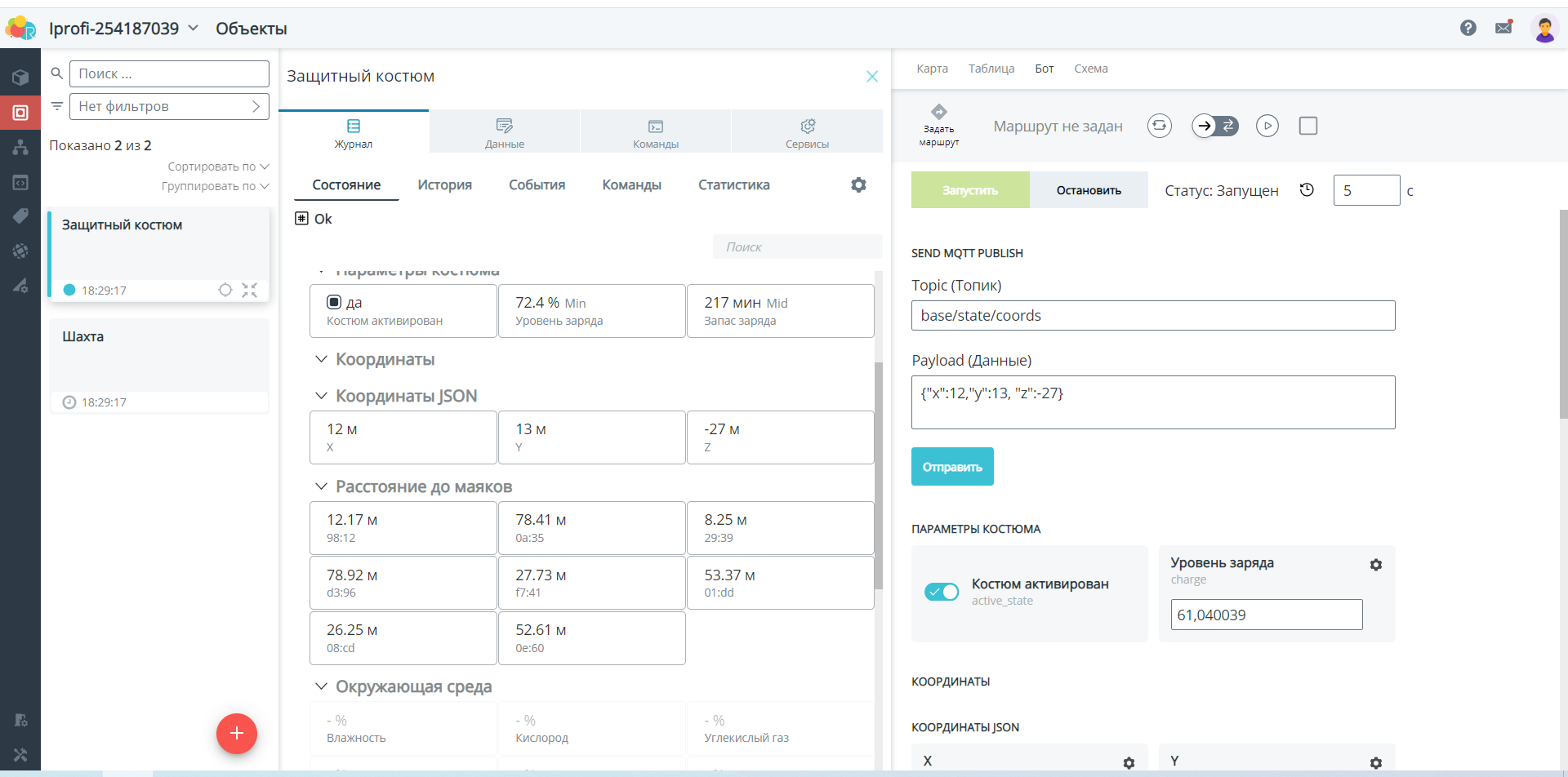


Рисунок 25

**Этап реализации кейса №3**

**Реализация сценариев автоматизации взаимодействия IoT-устройств**

В данном этапе необходимо создать 3 автомата, которые будут реагировать на изменение определенных параметров и выполнять определенные действия. Для создания автоматов используется вкладка «Логика» системы RIC.

**Сценарий №1**

**Вход в зону проведения работ**

Для реализации сценария создается конечный автомат на платформе RIC. Алгоритм работы автомата представлен на рисунке 26.

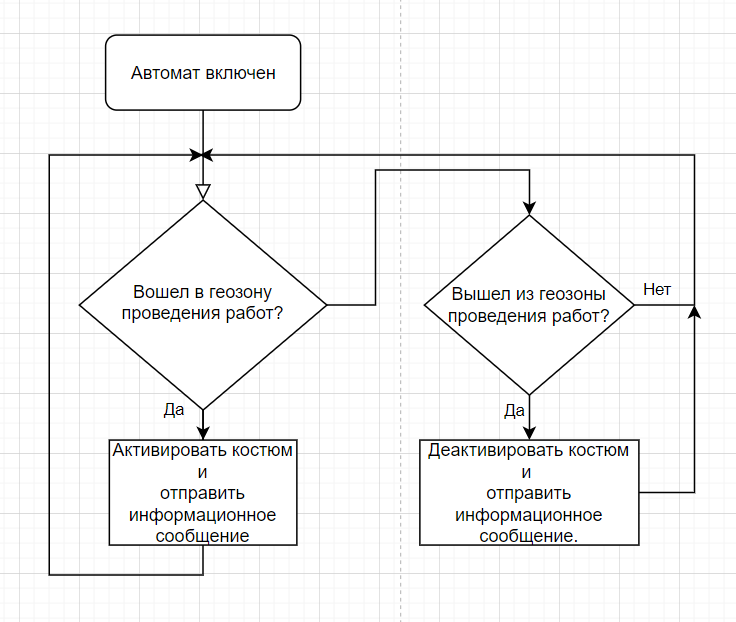


Рисунок 26

Готовый автомат представлен на рисунке 27.

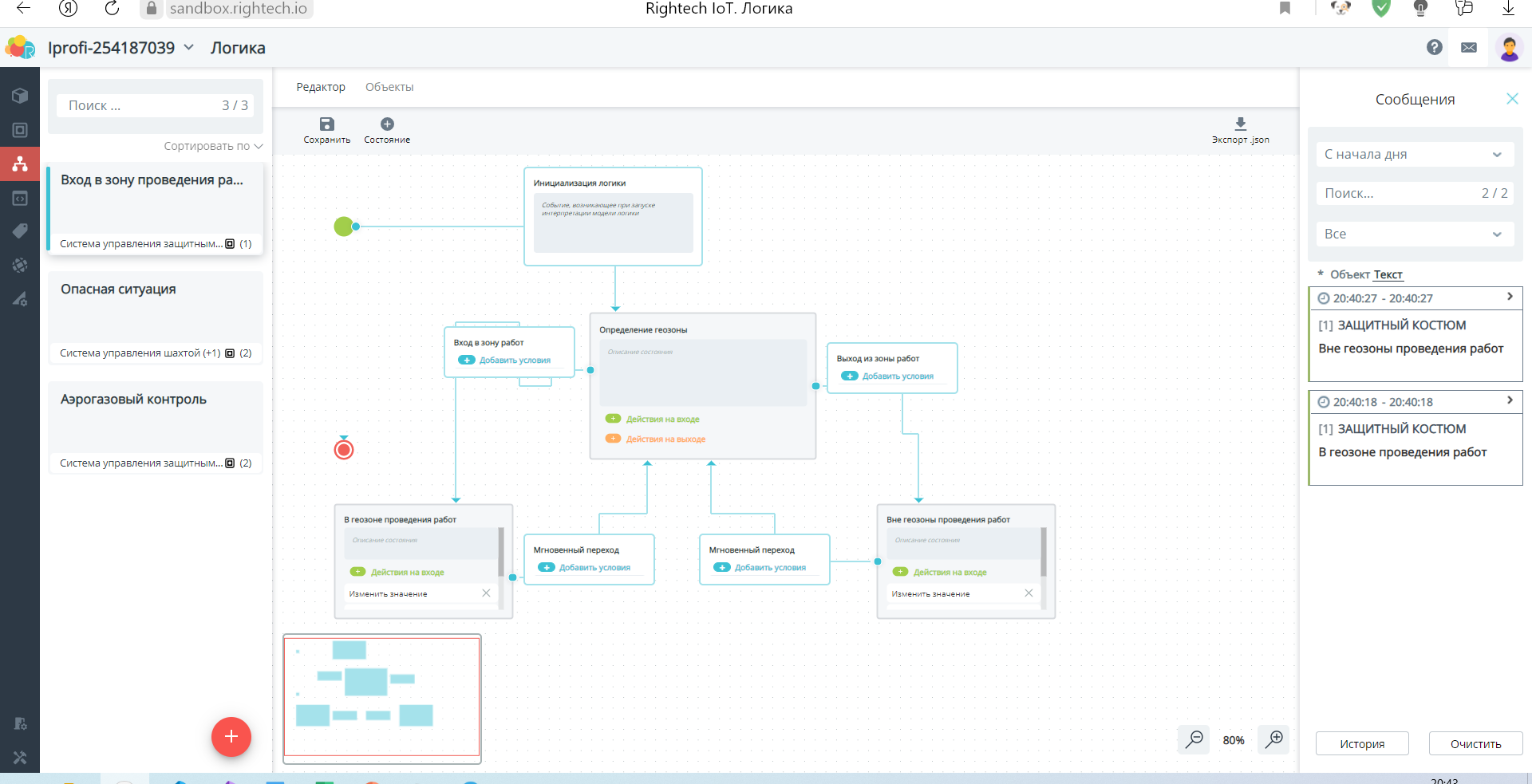


Рисунок 27

Стоит отметить, что это «бесконечный автомат», то есть он не имеет конечных состояний. Он работает следующим образом: постоянно проверяется вошел ли рабочий в зону проведения работ. Если работник вошел в зону, то отправляется команда активировать костюм, активируется костюм, отправляется информационное сообщение «В геозоне проведения работ». Если работник покидает данную геозону, то отправляется команда деактивировать костюм, деактивируется костюм, отправляется информационное сообщение «Вне геозоны проведения работ». Кроме этого, были созданы события «Вход в зону работ» и «Выход из зоны работ», для корректной работы автомата.

Результат работы автомата представлен на рисунке 28.

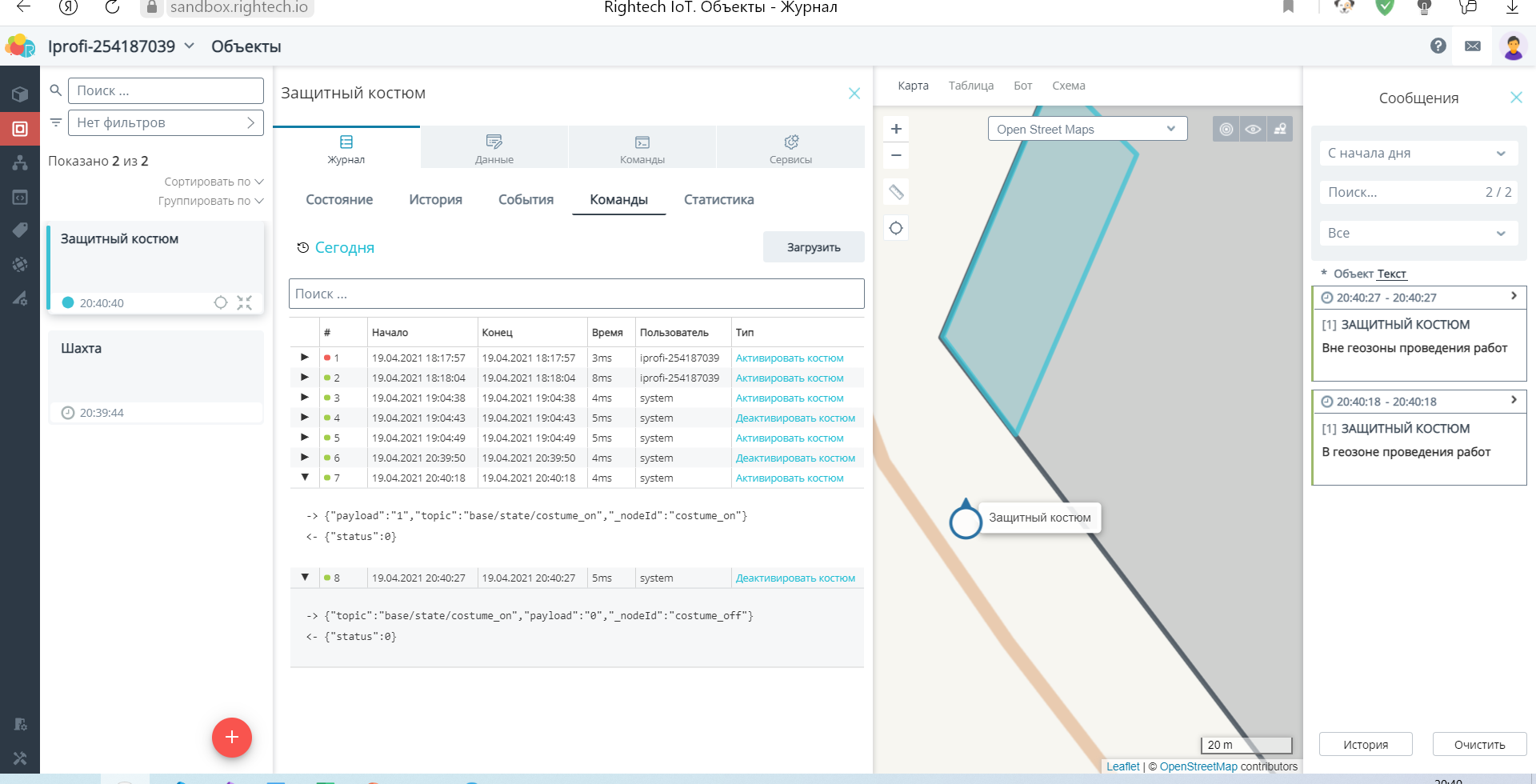


Рисунок 28

**Сценарий №2**

**Вход в зону проведения работ**

Для реализации сценария создается конечный автомат на платформе RIC. Алгоритм работы автомата представлен на рисунке 29.

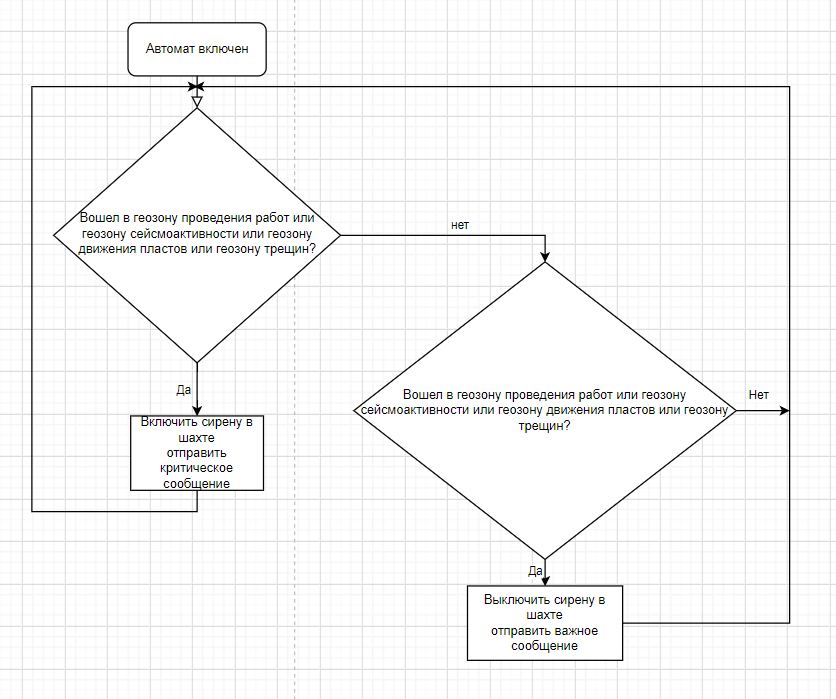


Рисунок 29

Стоит отметить, что это «бесконечный автомат», то есть он не имеет конечных состояний. Он работает следующим образом: постоянно проверяется вошел ли рабочий в зону сейсмоактивности или зону движения пластов или зону трещин. Если работник вошел какую-либо из этих зон, то отправляется команда включить сирену, включается сирена, отправляется критическое сообщение «В опасной зоне». Если работник покидает какую-либо из указанных геозон, то отправляется команда выключить сирену, сирена выключается, отправляется важное сообщение «В безопасности». Кроме этого, были созданы события «Вход в зону сейсмоактивности» ,«Выход из зоны сейсмоактивности», «Вход в зону движения пластов», «Выход из зоны движения пластов», «Вход в зону трещин», «Выход из зоны трещин» для корректной работы автомата. Готовый автомат представлен на рисунке 30.

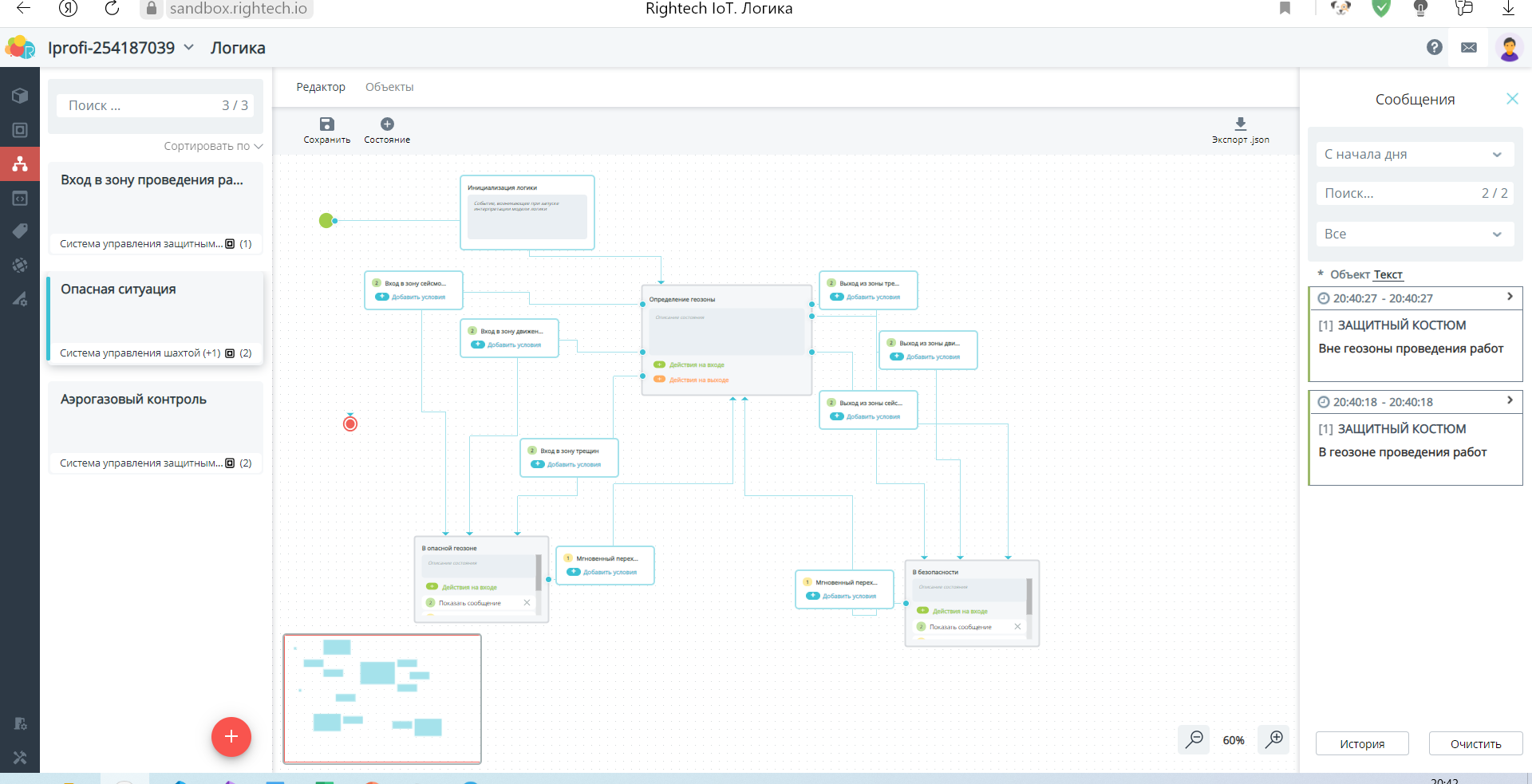


Рисунок 30

Результат работы автомата представлен на рисунке 31.

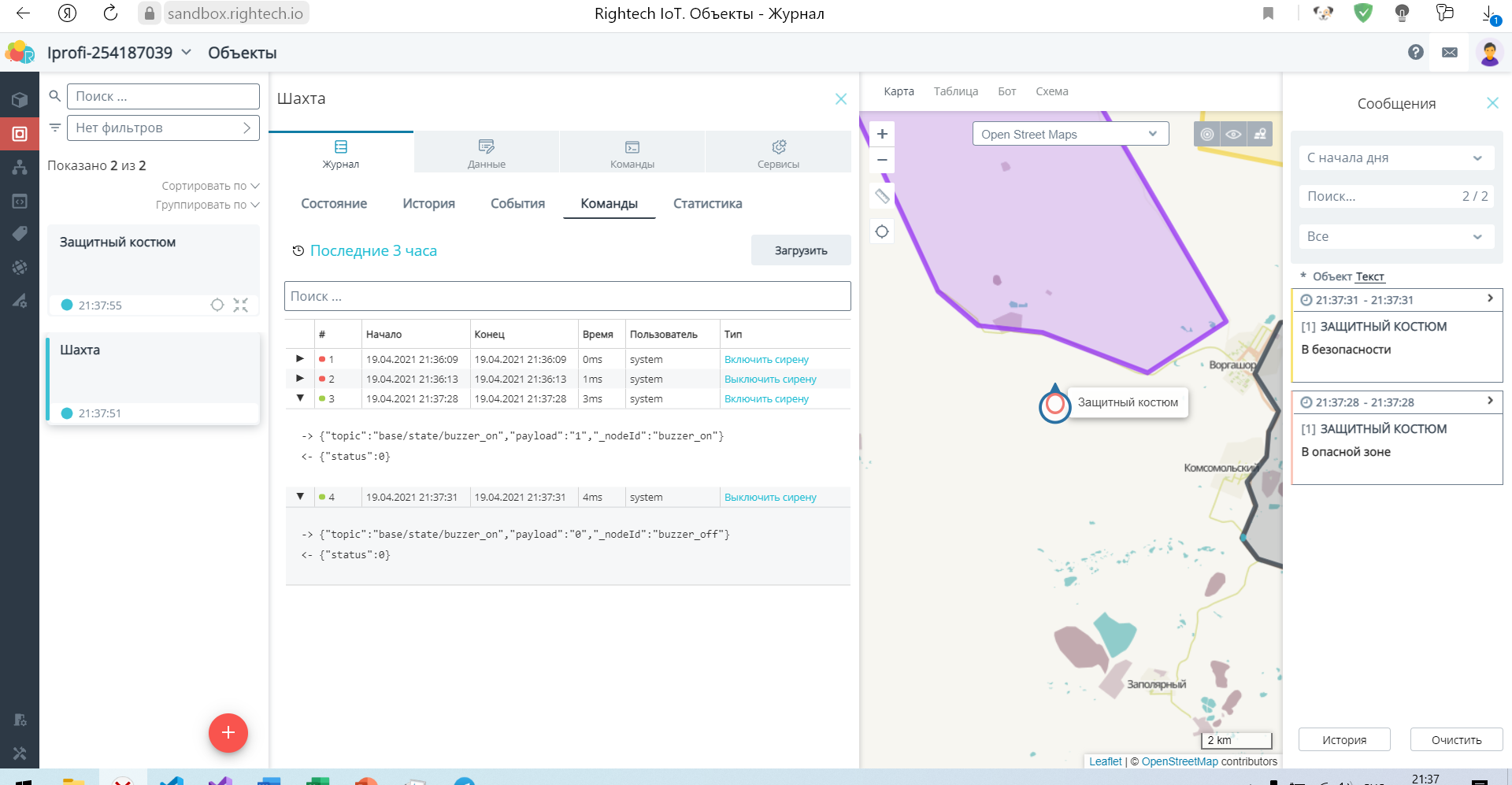


Рисунок 31

**Сценарий №3**

**Аэрогазовый контроль**

Для реализации сценария создается конечный автомат на платформе RIC. Алгоритм работы автомата представлен на рисунке 32.

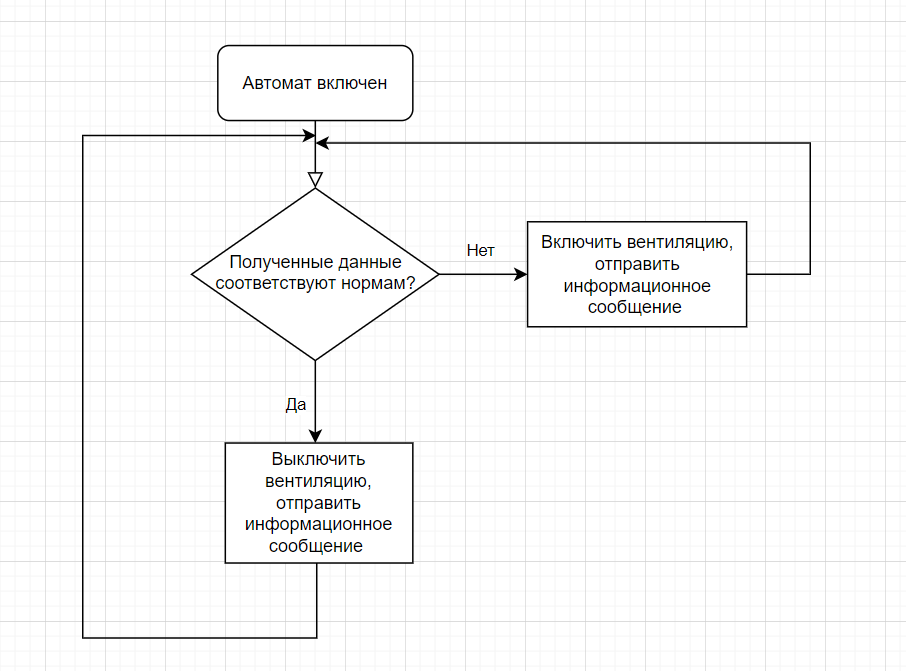


Рисунок 32

Стоит отметить, что это «бесконечный автомат», то есть он не имеет конечных состояний. Он работает следующим образом: постоянно проверяются параметры окружающей среды, описанные в таблице №4 файла задания. Если данные не соответствуют нормам, то отправляется команда включить вентиляцию, включается вентиляция, отправляется информационное сообщение «Вентиляция включена». Если данные соответствуют нормам, отправляется команда выключить вентиляцию, выключается вентиляция, отправляется информационное сообщение «Вентиляция выключена». Готовый автомат представлен на рисунке 33.

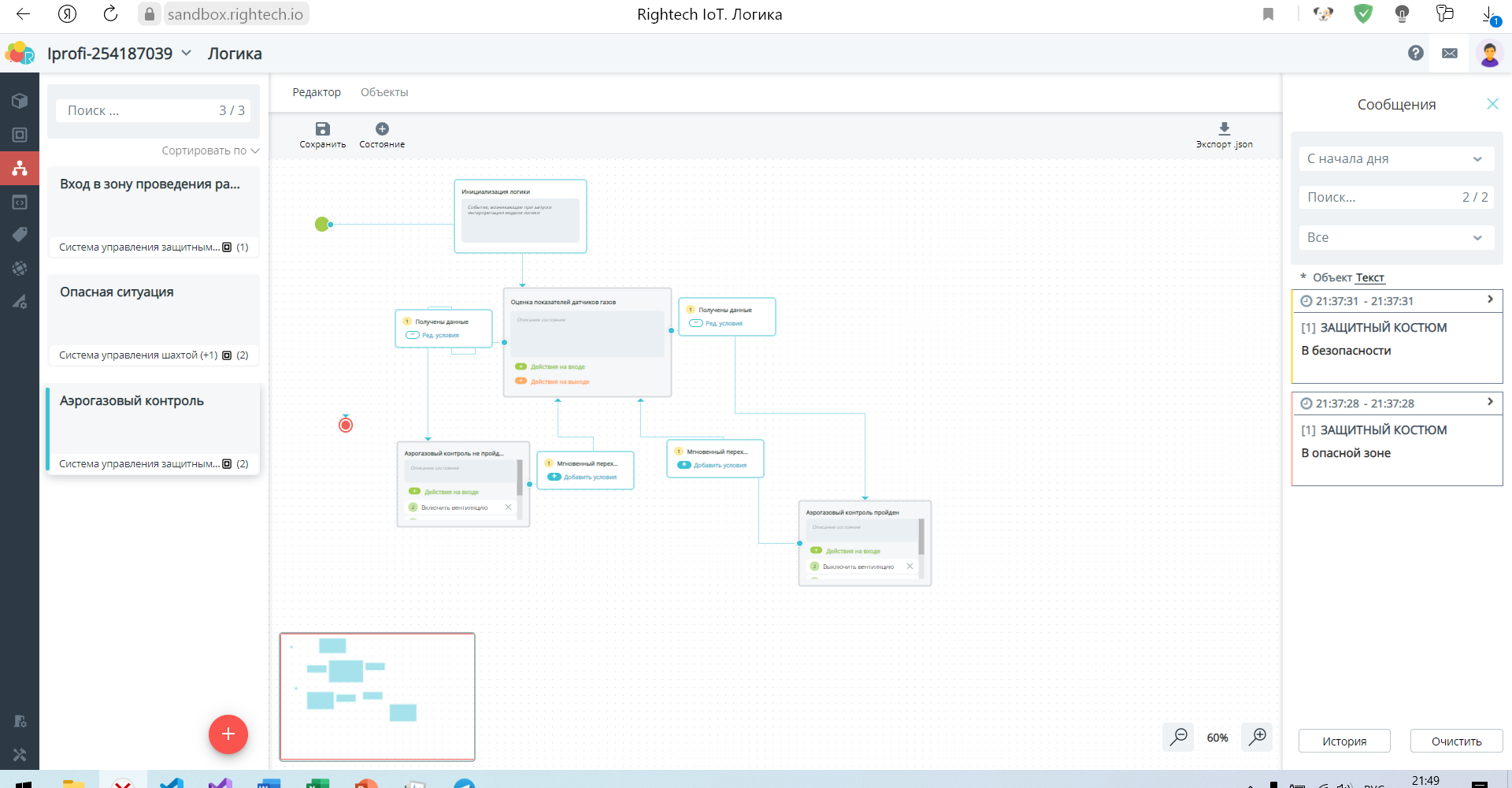


Рисунок 33

Результат работы автомата представлен на рисунке 34.

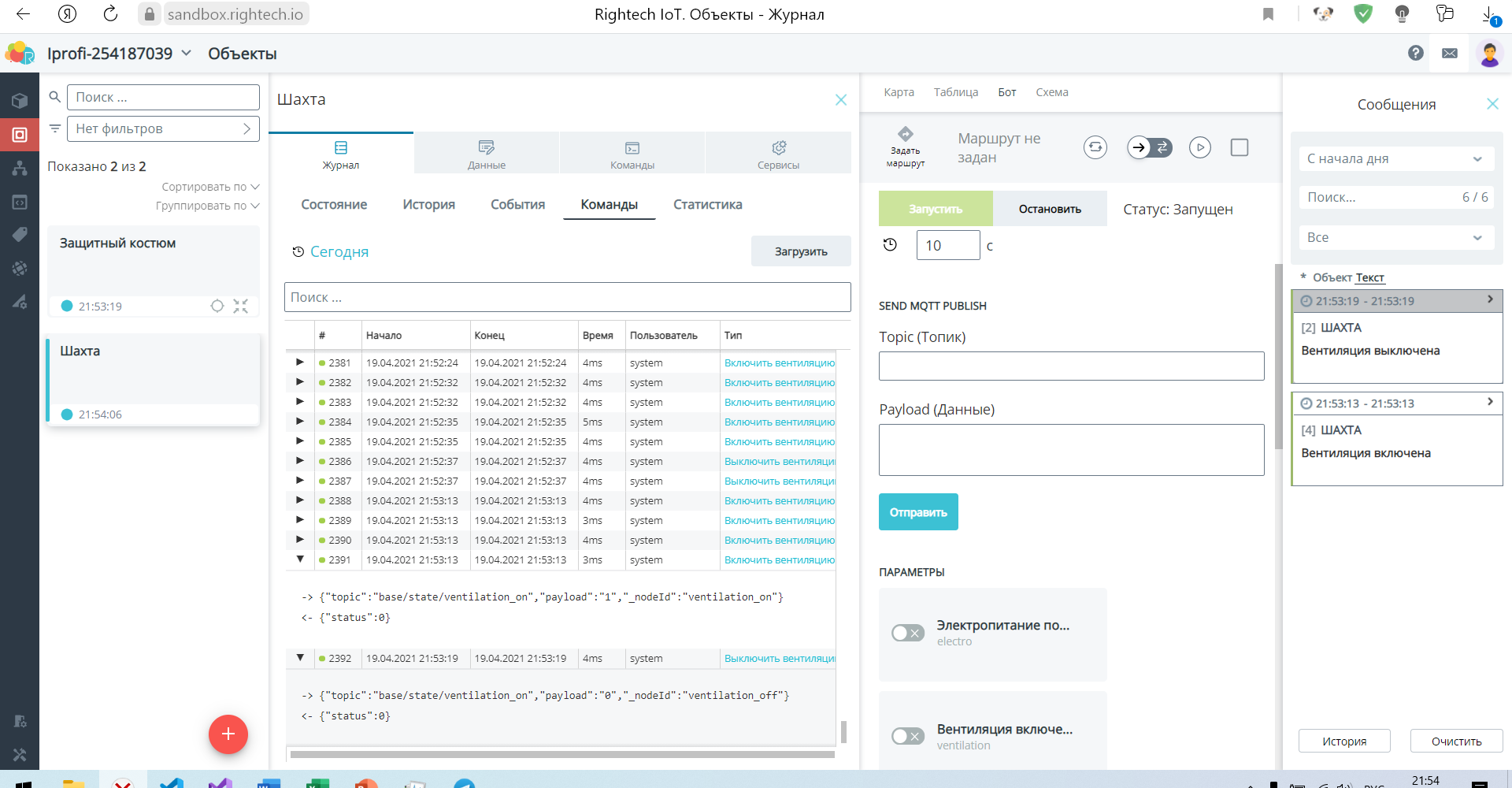


Рисунок 35

Более детально данный этап разобран в видеоматериале, доступном по ссылке:

<https://www.youtube.com/watch?v=0sXPK7rG8kk>