

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования

**"Южно-Уральский государственный университет  
(национальный исследовательский университет)"**

**Высшая школа электроники и компьютерных наук  
Кафедра системного программирования**

РАБОТА ПРОВЕРЕНА

Рецензент

IT-архитектор

VyndyuSoft

\_\_\_\_\_ А.В. Бындю

« \_\_\_\_ » \_\_\_\_\_ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,  
профессор

\_\_\_\_\_ Л.Б. Соколинский

« \_\_\_\_ » \_\_\_\_\_ 2019 г.

## **РАЗРАБОТКА ТЕХНОЛОГИИ СОЗДАНИЯ ЦИФРОВЫХ ДВОЙНИКОВ НА ОСНОВЕ РЕСУРСОВ ОБЛАЧНОЙ ВЫ- ЧИСЛИТЕЛЬНОЙ ПЛАТФОРМЫ**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА  
ЮУрГУ – 02.03.02.2019.13-018-1382.ВКР**

Научный руководитель,  
кандидат физ.-мат. наук, доцент

\_\_\_\_\_ Г.И. Радченко

Автор работы,  
студент группы КЭ-401

\_\_\_\_\_ Р.А. Бобин

Ученый секретарь  
(нормоконтролер)

\_\_\_\_\_ О.Н. Иванова

« \_\_\_\_ » \_\_\_\_\_ 2019 г.

Челябинск-2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
**"Южно-Уральский государственный университет  
(национальный исследовательский университет)"**  
Высшая школа электроники и компьютерных наук  
Кафедра системного программирования

УТВЕРЖДАЮ

Зав. кафедрой СП

---

Л.Б. Соколинский

09.02.2019

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы бакалавра**  
студенту группы КЭ-401  
Бобину Ростиславу Алексеевичу,  
обучающемуся по направлению  
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 25.04.2019 № 899)  
Разработка технологии создания цифровых двойников на основе ресурсов облачной вычислительной платформы.
2. **Срок сдачи студентом законченной работы:** 01.06.2019.
3. **Исходные данные к работе**
  - 3.1. K. Borodulin, G. Radchenko, A. Shestakov, L. Sokolinsky, A. Tchernykh, R. Prodan, "Towards Digital Twins Cloud Platform: Microservices and Computational Workflows to Rule a Smart Factory", Proc. the 10th Int. Conf. Util. Cloud Comput. - UCC '17, pp. 209-210, December 2017.
  - 3.2. G. Radchenko, A. Alaasam, A. Tchernykh, "Micro-Workflows: Kafka and Kepler fusion to support Digital Twins of Industrial Processes", IEEE/ACM Int.

Conf. Util. Cloud Comput. – UCC '18, pp. 83-88, December 2018.

#### **4. Перечень подлежащих разработке вопросов**

- 4.1. Обзор научной литературы и существующих решений Интернета вещей для создания цифровых двойников.
- 4.2. Изучение технологий, предоставляемых облачными вычислительными платформами, для создания цифровых двойников.
- 4.3. Разработка и тестирование прототипа цифрового двойника.

#### **5. Дата выдачи задания: 09.02.2019.**

#### **Научный руководитель**

Доцент кафедры СП,  
кандидат физико-математических наук

Г.И. Радченко

#### **Задание принял к исполнению**

Р.А. Бобин

## ОГЛАВЛЕНИЕ

|  |    |
|--|----|
| ВВЕДЕНИЕ.....  | 6  |
| 1. ОБЗОР ЛИТЕРАТУРЫ.....   | 9  |
| 1.1 Концепция Интернета вещей, облачных вычислений и цифровых двойников..... | 9  |
| 1.2 Технологии обработки данных Интернета вещей.....                         | 11 |
| 1.2.1 Microsoft Azure .....  | 12 |
| 1.2.2 Amazon Web Services .....  | 13 |
| 1.2.3 Концепция пространственного интеллектуального графа .....              | 17 |
| 2. МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ .....                                       | 21 |
| 2.1 Данные устройств IoT.....  | 21 |
| 2.2 Требования к системе Цифровой двойник «Энергопотребление» .....          | 22 |
| 2.3 Варианты использования системы.....                                      | 23 |
| 3. АРХИТЕКТУРА СИСТЕМЫ.....  | 26 |
| 3.1 Общее описание архитектуры системы.....                                  | 26 |
| 3.2 Описание компонентов, составляющих систему.....                          | 27 |
| 3.3 Импорт показаний устройств и схема базы данных.....                      | 29 |
| 3.4 Взаимодействие с API Digital Twins .....                                 | 30 |
| 4. РЕАЛИЗАЦИЯ СИСТЕМЫ .....  | 32 |
| 4.1 Средства реализации.....   | 32 |
| 4.2 Иерархия пространственного интеллектуального графа.....                  | 32 |
| 4.3 База данных.....   | 34 |
| 4.4 Реализация отправки телеметрии и ее обработки .....                      | 35 |
| 4.5 Реализация графического интерфейса пользователя .....                    | 36 |
| 5. ТЕСТИРОВАНИЕ .....  | 38 |

|   |    |
|---|----|
| 5.1 Тестирование работоспособности системы .....  | 38 |
| 5.2 Сравнительное тестирование с целью определения правильности<br>обработки телеметрии датчиков..... | 38 |
| 5.3 Тестирование API управления.....  | 39 |
| ЗАКЛЮЧЕНИЕ .....  | 40 |
| ЛИТЕРАТУРА.....   | 41 |
| ПРИЛОЖЕНИЯ.....   | 45 |
| Приложение 1 .....  | 45 |
| Приложение 2 .....  | 49 |
| Приложение 3 .....  | 53 |
| Приложение 4 .....  | 55 |

## ВВЕДЕНИЕ

Концепция Интернета вещей (IoT) сформулирована достаточно давно, однако активное наполнение концепции технологическим содержанием и внедрение практических решений для ее реализации происходит именно в последние годы. Данная концепция представляет собой вычислительную сеть физических предметов, оснащенных встроенными технологиями для взаимодействия друг с другом или с внешней средой [17]. Общий мировой объем капиталовложений в IoT в 2018 году составил 646 миллиардов долларов США. Прогноз в отношении капиталовложений в эти технологии на 2019 год – 745 миллиардов долларов США, на 2022 – более 1 триллиона долларов [16].

Развитие распределенной сетевой инфраструктуры в автоматизированных системах управления технологическим процессом привело к появлению Промышленного Интернета вещей (Industrial Internet of Things – IIoT) – концепции взаимосвязи датчиков, приборов и других устройств, объединенных в сеть с промышленными приложениями компьютеров, включая, помимо прочего, управление производством и энергопотреблением. Такая связь позволяет собирать данные устройств и анализировать их, что потенциально способствует повышению производительности труда и эффективности производства, а также другим экономическим преимуществам [13]. Одним из преимуществ внедрения этого подхода является возможность создания цифрового двойника (Digital Twin) разрабатываемой системы.

Цифровой двойник – это иерархическая система математических моделей, вычислительных методов и программного обеспечения, которая обеспечивает синхронизацию между состоянием реально существующего процесса или системы и сопутствующей виртуальной копией [23, 26].

Благодаря развитию облачных технологий появилась возможность создать инфраструктуру хранения данных, способную поддерживать Интернет вещей. Публичные облачные платформы предоставляют множество ре-

шений для цифрового преобразования бизнеса. Гибкость и автоматизированность полученной среды достигается за счет использования прикладных программных интерфейсов (API). Это позволяет различным устройствам и системам взаимодействовать между собой, даже если они работают на основе разных стандартов и протоколов [30].

Применение облачных технологий может упростить обработку сверхбольших массивов данных, генерируемых устройствами Интернета вещей. Этот факт убеждает в необходимости изучения инструментов облачных платформ для разработки приложений, обрабатывающих данные с устройств IoT. Одними из возможных решений в этой области являются облачные платформы Microsoft Azure [22] и Amazon Web Services [3]. Ключевым преимуществом данных платформ является широкий набор инструментов, в частности, для создания решений Интернета вещей и цифровых двойников.

*Целью данной работы* является разработка технологии создания цифровых двойников, а также реализация и тестирование прототипа цифрового двойника на основе ресурсов облачной вычислительной платформы.

Для достижения цели работы, необходимо решить следующие задачи:

- провести обзор научной литературы и существующих решений Интернета вещей для создания цифровых двойников;
- изучить технологии, предоставляемые облачными вычислительными платформами, для создания цифровых двойников;
- спроектировать, разработать и протестировать прототип цифрового двойника.

### **Структура и объем работы**

Работа состоит из введения, 5 разделов, заключения, библиографии и 4 приложений. Объем работы составляет 55 страниц, объем библиографии – 36 источников, объем приложений – 10 страниц.

Первая глава содержит описание концепции Интернета вещей, облачных вычислений и цифровых двойников, анализ предметной области, обзор существующих работ по теме создания цифровых двойников на основе облачных вычислительных ресурсов.

Во второй главе описана модель прототипа цифрового двойника, определены требования к системе и варианты ее использования.

В третьей главе приведена архитектура системы.

Четвертая глава описывает детали реализации прототипа.

В пятой главе приводятся результаты тестирования разработанного прототипа.

В заключении сделаны выводы о проделанной работе.

Приложение 1 содержит спецификацию вариантов использования разрабатываемой системы.

Приложение 2 содержит реализацию отправки телеметрии.

Приложение 3 содержит исходный код тестирования модуля создания сенсоров.

Приложение 4 содержит протокол тестирования API управления цифровым двойником.



## **1. ОБЗОР ЛИТЕРАТУРЫ**

### **1.1. Концепция Интернета вещей, облачных вычислений и цифровых двойников**

Интернет вещей представляет собой сеть физических объектов, которые содержат встроенные технологии для связи, восприятия или взаимодействия со своими внутренними состояниями или внешней средой [17]. Появление этой концепции представляет новую эру в области вычислительной техники и технологий [28]. Лидерами по объему инвестиций в Интернет вещей в России к 2020 году, по прогнозу IDC, станут производственный сектор и транспортные компании [35].

Благодаря применимости подхода Интернета вещей к различным производственным процессам, появилось множество наполненных новым содержанием и возможностями терминов, описывающих существовавшие ранее подходы [29]. Так, в дополнение к автоматизированным системам управления технологическими процессами (АСУ ТП) была сформирована концепция промышленного Интернета вещей (IIoT). Данная концепция предлагает использование взаимосвязанных датчиков, приборов и других устройств, объединенных в сеть с промышленными приложениями компьютеров, включая, помимо прочего, управление производством и энергопотреблением [18], что обеспечивает сбор, обмен и анализ данных, потенциально способствует повышению производительности и эффективности труда, а также другим экономическим преимуществам [13].

Одним из подходов к применению IIoT является концепция «Индустрия 4.0» (Industry 4.0) [1]. Согласно концепции Индустрии 4.0, современная экономика находится на пороге четвертой промышленной революции. Ее ключевые моменты – внедрение киберфизических систем в промышленные процессы и переход к персонализированному производству [1]. Киберфизические системы характеризуются наличием двусторонней связи между физическими процессами и управляющими программами [29]. Эле-

менты такой системы могут находиться как рядом, например, в одной производственной зоне, так и далеко друг от друга, а взаимодействие между ними – осуществляться на всех стадиях «жизненного цикла» (планирование, производство, эксплуатация, ремонт, утилизация).

Сбор и анализ собираемой информации могут служить множеству целей: диагностика состояния, прогнозирование необходимости тех или иных изменений, автоматическая настройка, адаптация и пр. Одним из подходов реализации киберфизических систем является концепция цифровых двойников (Digital Twin – DT), которая поддерживает виртуальные модели реального оборудования, производственных процессов и конечных продуктов. В данной концепции используются методы анализа данных с различных типов датчиков, установленных на объектах, для настройки и актуализации их виртуального состояния. Также в DT используются различные математические модели для моделирования интересующих процессов с использованием статистических методов, интеллектуального анализа данных, метода конечных элементов и т. д. [2].

При переходе экономики к модели Индустрии 4.0 акцент будет смещаться на аналитику. Все шире станут использоваться преимущества облачной модели, средства машинного обучения и когнитивные вычисления. Облачные вычисления (англ. cloud computing) – это модель обеспечения удобного сетевого доступа по требованию к некоторому общему фонду конфигурируемых вычислительных ресурсов (например, сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам – как вместе, так и по отдельности), которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами или обращениями к провайдеру [20]. При этом в платформах IoT активнее будут применяться открытые программные интерфейсы API для обмена данными с другими системами, а также различные приложения с открытым программным кодом. Увеличение обратных связей позволит существенно расширить возможности систем. Облачные вычисления способны справиться с

обработкой большого количества данных, которые поставляют устройства IoT, однако разные модели обработки требуют разных вычислительных ресурсов, а разные данные требуют разных подходов к их обработке и защите [28].

Следовательно, важной тенденцией станет перенос интеллекта ближе к конечным устройствам и наделение их все большими возможностями. Несмотря на эффективность облачной модели, ситуация, когда анализ данных и выработка управляющих воздействий происходят в удаленном крупном центре обработки данных (ЦОД), для решений многих задач является нежелательной [15]. Передача трафика до ЦОДа и обратно может привести к недопустимым задержкам там, где требуется быстрая реакция (производственные процессы, системы безопасности, взаимодействие автомобиля с дорожной инфраструктурой и пр.).

Как следствие, наряду с переносом интеллекта в крупные центры обработки данных формируется обратный процесс, получивший название «туманные вычисления» (Fog computing), когда анализ собираемых данных и выработка управляющих воздействий происходят в узлах, максимально приближенных к конечным устройствам [12]. В будущем все в большей степени анализ собираемых данных будет производиться самими устройствами, что позволит минимизировать задержку при принятии решений и повысит автономность устройств. Только применение технологии гибридных облачных сред могут решить поставленные перед концепцией цифровых двойников задачи.

## **1.2. Технологии обработки данных Интернета вещей**

Применение облачных технологий может упростить обработку данных, генерируемых устройствами Интернета вещей. В этом разделе мы рассмотрим концепцию пространственного интеллектуального графа, как одного из самых распространенных методов проектирования систем обработки данных IoT на сегодняшний день, а также проанализируем возможно-

сти по реализации приложений обработки данных IoT на базе наиболее применяемых на сегодняшний день публичных облачных платформ: Microsoft Azure и Amazon Web Services. Ключевым преимуществом данных платформ является широкий набор инструментов, в частности, для создания решений Интернета вещей и цифровых двойников.

### 1.2.1. Microsoft Azure

Azure Digital Twins от Microsoft представляет собой службу Интернета вещей, с помощью которой можно создать комплексные модели физического окружения в виде пространственных интеллектуальных графов для моделирования связей и взаимодействий между людьми, пространствами и устройствами [9]. Azure Digital Twins имеет следующие ключевые возможности:

- *пространственный интеллектуальный граф* является виртуальным представлением физического окружения. С его помощью можно воспроизводить зависимости из реального мира путем моделирования связей между пользователями, расположениями и устройствами. Кроме того, Digital Twins требует, чтобы каждая часть получаемых данных телеметрии была связана с датчиком в пространственном графе. Это позволяет объединять данные из многих разрозненных источников в некое единое физическое пространство [8];

- *цифровые модели объектов двойников* – это предопределенные протоколы устройств и схема данных. Они представляют потребности определенной предметной области, чтобы ускорить и упростить разработку;

- *расширенные вычислительные ресурсы* – благодаря возможности определять и запускать пользовательские функции в отношении входящих данных устройства для отправки сигналов в заранее определенные конечные точки;

- *встроенное управление доступом* с помощью функций управления

доступом и идентификацией, такие как управление доступом на основе ролей и Azure Active Directory.

Технология Azure Digital Twins использует Azure IoT Hub для хранения данных устройств и сенсоров, а для реализации событий, произошедших в системе – Центр событий Azure Event Hub. Azure IoT Hub – это размещенная в облаке управляемая служба, которая действует в качестве центра сообщений для двусторонней связи между приложением Интернета вещей и устройствами, которыми оно управляет [11]. Инструмент Azure Event Hub служит конечной точкой исходящих событий и сообщений в соответствии с предопределенными настройками маршрутизации [10]. Данная служба отвечает за хранение и обработку событий таких, как изменения в вычисленном значении пространства как результате сообщения о телеметрии устройства, полученных в свою очередь из генератора событий – функции, сравнивающей текущие показатели датчиков с заранее заданными оптимальными значениями. На рисунке 1 представлена диаграмма потока данных Azure Digital Twins.

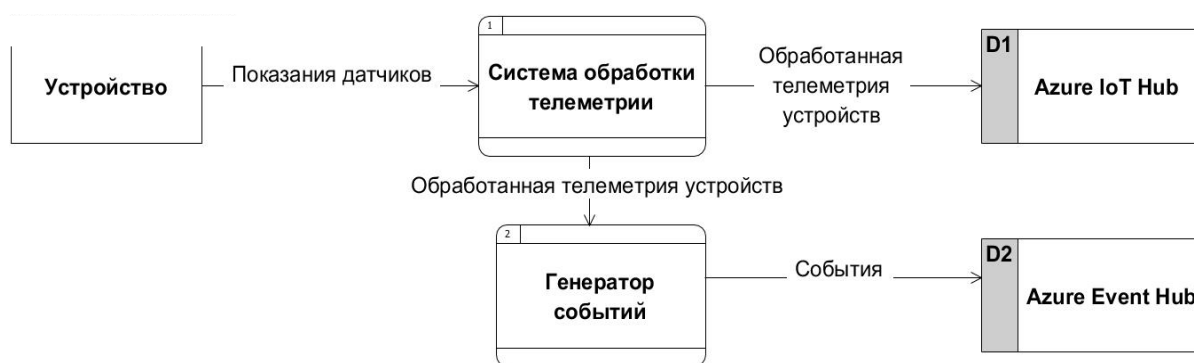


Рис. 1. Диаграмма потока данных Azure Digital Twins

### 1.2.2. Amazon Web Services

AWS IoT Things Graph – продукт в инфраструктуре облака от Amazon [7]. Этот сервис позволяет создавать автоматизированные системы Интернета вещей, которые помогают осуществлять связь между веб-сервисами и физическими устройствами, использующими различные протоколы, фор-

маты данных и синтаксис сообщений [8]. Данное решение также предоставляет мощное средство визуализации построенного графа с возможностью редактирования связей между устройствами и приложениями прямо в редакторе потоков данных. Этот продукт представлен следующими тремя ключевыми концепциями:

- *Модель*. Это абстракция, представляющая устройство как набор действий (входов), событий (выходов) и состояний (атрибутов). Модель отделяет интерфейс устройства от его базовой реализации.

- *Отображение*. Отображение представляет информацию, которая позволяет AWS IoT Things Graph преобразовать вывод сообщения одного объекта в ожидаемый ввод другого. Эта концепция устраняет различия объектов и позволяет создавать приложения IoT, в которых используется множество объектов.

- *Поток*. Поток состоит из последовательности взаимодействий между устройствами и web-сервисами, которые необходимы для выполнения задачи. Потоки позволяют определять связанные действия, необходимые для автоматизации задач.

AWS IoT Things Graph сочетает в себе возможности двух других базовых решений Amazon для Интернета вещей – AWS IoT Core [4] и AWS IoT Greengrass [5].

AWS IoT Core – это платформа, которая обеспечивает подключение устройств к сервисам AWS и другим устройствам, безопасность данных и коммуникации, обработку данных устройств и различные операции с ними. Можно выделить следующие основные возможности платформы:

- *Message Broker*. Это высокопроизводительный брокер сообщений, работающий по стандарту «издатель-подписчик», который передает сообщения всех устройств IoT и приложений в нужном направлении;

- *Шлюз устройств* служит точкой входа для устройств IoT, подклю-

чающихся к AWS. Шлюз устройств управляет всеми активными подключениями устройства и реализует семантику различных протоколов, чтобы обеспечить связь устройств с AWS IoT Core;

- *Пакет SDK AWS IoT* для устройств позволяет подключить аппаратное устройство или мобильное приложение к AWS IoT Core;

- *Метод аутентификации AWS* (называемый SigV4), аутентификация на основе сертификата X.509 и специальная аутентификация на основе токенов (через настраиваемые модули авторизации) обеспечивают взаимную аутентификацию и шифрование во всех точках подключения;

- *Реестр* устанавливает идентификацию для устройств и позволяет отслеживать метаданные, такие как атрибуты или возможности устройства. Реестр позволяет уникальным образом идентифицировать каждое устройство в соответствии с единым форматом, не зависящим от типа устройства или его подключения;

- *Тени устройств*. С помощью AWS IoT Core можно создать постоянную виртуальную версию каждого устройства, так называемую тень устройства, содержащую его последнее состояние и позволяющую приложениям или другим устройствам считывать сообщения и взаимодействовать с данным устройством. Тени устройств хранят последнее зарегистрированное состояние и желаемое будущее состояние каждого устройства;

- *Сервис правил* позволяет создавать приложения IoT для сбора, обработки и анализа данных, генерируемых подключенными устройствами, и выполнения действий с ними. Сервис правил оценивает входящие сообщения, публикуемые в AWS IoT Core, а затем преобразует и доставляет их другому устройству или облачному сервису с учетом заданных бизнес-правил. Правило можно применять к данным от одного или от многих устройств и выполнять на его основе одно действие или же множество параллельных действий.

AWS IoT Greengrass – это инструмент, предназначенный для эффективного распространения возможностей AWS на периферийные устройства, что позволяет им локально работать с данными, которые они создают, используя при этом облако для управления, анализа и надежного хранения данных.

Основным преимуществом использования AWS IoT Greengrass, является возможность локально формировать на устройствах выводы с использованием машинного обучения, применяя модели, созданные и обученные в облаке. Также стоит отметить коннекторы системы AWS IoT Greengrass, позволяющие выполнять поиск, импорт, настройку и развертывание приложений и сервисов на периферии. При этом не требуется разбираться в различных протоколах устройств, управлять данными для доступа или взаимодействовать с внешними API. Кроме того, можно применить общую бизнес-логику одного устройства AWS IoT Greengrass к другому.

В общем виде настройка и развертывание AWS IoT Things Graph состоит из следующих действий: создание потоков, настройка конечных точек и отображений, создание хранилища для конфигурации приложения, а также само развертывание.

В библиотеке моделей AWS IoT Things Graph хранится большое количество моделей, которые можно использовать в редакторе потоков, а также можно создавать собственные модели и добавлять их в эту библиотеку. Пример настройки иерархии графа, содержащего устройства и сервисы, с помощью редактора потоков AWS Things Graph продемонстрирован на рисунке 2. Взаимодействия между моделями определяются с помощью настройки маршрутизации данных в редакторе потоков. После настройки локального развертывания AWS IoT Graph упаковывает потоки вместе с их зависимостями и передает их на устройства из реестра созданной группы AWS IoT Greengrass. Это позволяет графу управлять взаимодействием между локально подключенными устройствами.

После развертывания потока в AWS IoT Things Graph отображаются



такие показатели, как успешность выполнения или неудача, а также время выполнения, сохраняется всю историю выполнения для воспроизведения и отладки.

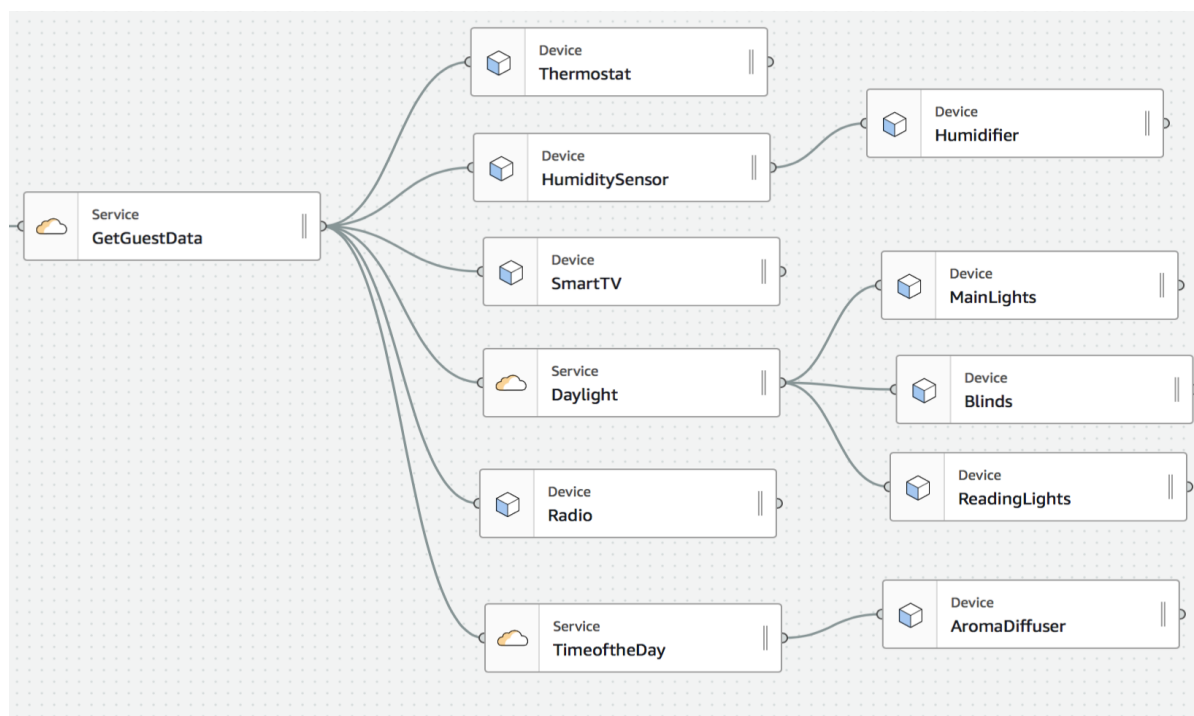


Рис. 2. Пример настройки иерархии графа с помощью редактора потоков AWS Things Graph

### 1.2.3. Концепция пространственного интеллектуального графа

Объектные модели описанных выше систем хранятся в виде интеллектуальных графов. Такое представление позволяет эффективно моделировать взаимосвязи и взаимодействия между людьми, пространствами и устройствами. Рассмотрим отдельно концепцию пространственного интеллектуального графа, используемую в Azure Digital Twins [34]. Данная концепция представляет физические среды и связанные устройства, датчики и пользователей в качестве моделей, хранящихся в виде пространственного интеллектуального графа. Модели создают пользователи, которые хотят адаптировать технологическое решение к своим конкретным потребностям. Вместе эти предварительно определенные объектные модели Azure Digital Twins составляют онтологию. В онтологии могут быть описаны регионы,

площадки, этажи, офисы, зоны, конференц-залы и фокус-комнаты, различные электростанции, подстанции, энергетические ресурсы и потребители. Объектные модели и онтологии Digital Twins позволяют персонализировать разные сценарии и потребности. Можно выделить следующие основные категории объектов Digital Twins:

- *Пространства* являются виртуальными или физическими расположениями;
- *Устройства* являются виртуальными или физическими единицами оборудования, например, Raspberry Pi 3;
- *Датчики* являются объектами, которые позволяют обнаруживать события;
- *Пользователи* представляют собой жильцов и их характеристики;
- *Ресурсы* присоединены к пространству и обычно представляют ресурсы Azure для использования объектами в пространственном графе, например, IoTHub [11];
- *Онтологии* – предварительно определенные объектные модели Digital Twins, представляющие наборы расширенных типов, например, стандартная онтология BASnet определяет типы данных, применяющиеся в создании цифровых двойников различных «умных» зданий;
- *Роли* – это наборы разрешений, которые назначаются пользователям и устройствам в пространственном графе, например, администратор системы;
- *Определяемые пользователем функции* позволяют настраивать обработку телеметрии датчика в пространственном графе, например, задать значение датчика или отправлять уведомления, когда выполняются предварительно определенные условия;
- *Конечные точки* – это расположения, в которые направляются сообщения телеметрии и события Digital Twins, например, Azure Event Hub [10];

- *Сопоставители* являются объектами, которые определяют набор условий, оценивающих, какие действия предпринимаются на основе входящих данных телеметрии, то есть какие именно пользовательские функции выполняются для заданного сообщения телеметрии.

Пространственный граф – это иерархический граф пространств, устройств и людей, определенных в объектной модели Azure Digital Twins. Пространственный граф поддерживает наследование, фильтрацию, обход, масштабируемость и расширяемость. Взаимодействие с пространственным графом и управление им осуществляется с помощью коллекции REST API [14].

На рисунке 3 с помощью инструмента просмотра и редактирования графов Microsoft Azure Digital Twins Graph Viewer [21] представлена схема созданного пространственного интеллектуального графа. Он состоит из следующих компонентов:

- *Пространства*. Представлены виртуальными помещениями с идентификаторами (Focus Rooms);
- *Устройства* являются виртуальными единицами оборудования Raspberry Pi 3 с идентификатором номера комнаты в названии;
- *Датчики* являются виртуальными сенсорами потребления электроэнергии. Установлены на каждом устройстве;

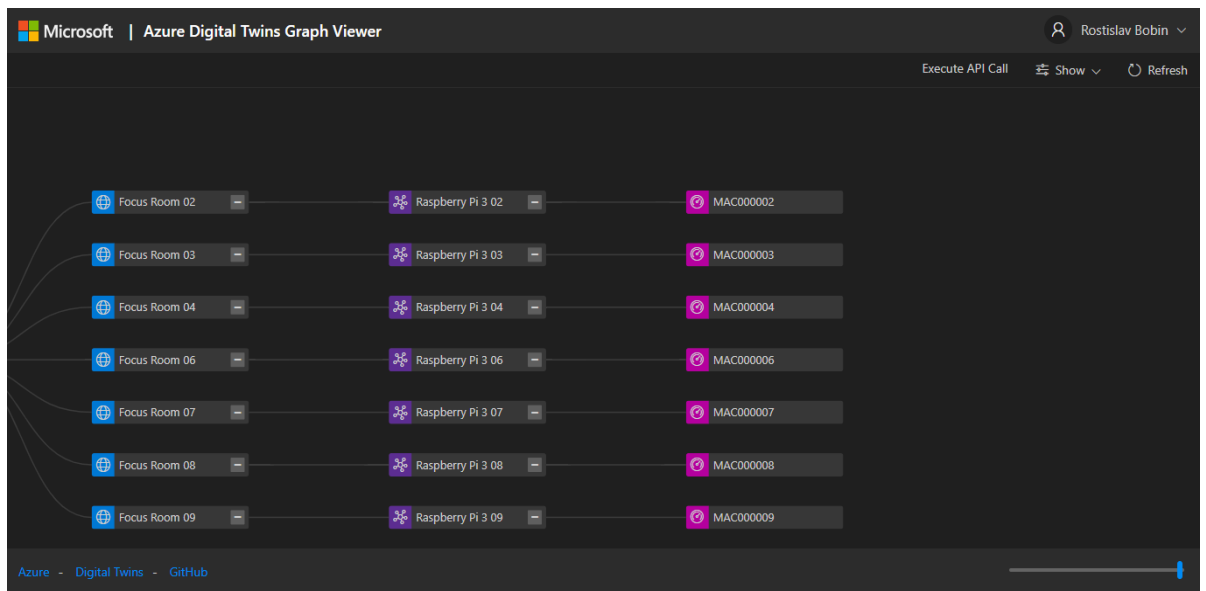


Рис. 3. Схема пространственного интеллектуального графа Azure Digital Twins

## 2. МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

### 2.1. Данные устройств IoT

В качестве источника данных было принято решение использовать проект Low Carbon Project [25], обеспечивающий функционирование системы смс-оповещения клиентов о текущей динамической тарифной стоимости электроэнергии. В рамках реализации проекта, дома-участники были разбиты на две тарифные группы. Первая группа пользовалась стандартными тарифами потребления электроэнергии. Потребителям второй группы отправлялись сигналы о высокой цене энергопотребления для снижения нагрузки на локальные распределительные сети в течение периодов высокой загруженности. Проект был реализован Британскими электросетями (UK Power Networks) в Лондоне в период с ноября 2011 года по февраль 2014 года. Выбранный набор данных представляет собой показатели энергопотребления для выборки из 5567 лондонских домов, оснащенных датчиками потребления энергии. Чтение данных проводилось с получасовыми интервалами. Набор данных содержит потребление энергии, в кВт/ч (за полчаса), уникальный идентификатор домохозяйства, дату и время, а также тарифную группу. Можно выделить несколько задач, образующихся при анализе этих данных:

- сегментация дневного потребления энергии;
- разбивка кривой электрической нагрузки;
- прогнозирование потребления электроэнергии домами;
- исследование различий последствий использования; электрической и аккумуляторной систем отопления;
- прогнозирование потребления электроэнергии в масштабе города.

Для разработки цифрового двойника была использована часть вышеописанного набора данных [19], представляющая собой файл формата CSV, схема которого описана в таблице 1.

Табл. 1. Схема исходного набора данных

| LCLid     | stdor<br>ToU | DateTime                       | KWH/h<br>h | Acorn       | Acorn_grouped |
|-----------|--------------|--------------------------------|------------|-------------|---------------|
| MAC000002 | Std          | 2012-10-12<br>00:30:00.0000000 | 0.143      | ACORN<br>-A | Affluent      |
| MAC000006 | ToU          | 2012-10-28<br>16:30:00.0000000 | 0.013      | ACORN<br>-Q | Adversity     |
| MAC000027 | Std          | 2013-08-13<br>15:00:00.0000000 | 0.41       | ACORN<br>-H | Comfortable   |
| MAC000032 | Std          | 2013-09-11<br>12:00:00.0000000 | 0.112      | ACORN<br>-J | Comfortable   |

Рассмотрим подробнее поля представленной таблицы:

- *LCLid* – уникальный идентификатор квартиры;
- *stdorToU* – тарифный план квартиры;
- *DateTime* – время очередного сбора данных с IoT-устройства;
- *KWH/hh* – показания IoT-устройства, отображающие количество потребленных киловатт электроэнергии за 30 минут;
- *Acorn* – текущая категория квартиры по шкале Acorn;
- *Acorn\_grouped* – статус текущего энергопотребления по шкале Acorn.

## 2.2. Требования к системе Цифровой двойник «Энергопотребление»

На основании анализа предметной области и описания модели системы были определены следующие функциональные требования к проектируемой системе:

- система должна предоставлять информацию по запросу пользователей о количестве потребляемой ими электроэнергии в текущий момент и в заданный промежуток времени;

- система должна предоставлять интерфейс добавления новых пользователей и устройств;
- система должна оповещать пользователей о неэффективном энергопотреблении при выполнении заранее определенных условий.

На основании анализа существующих технологий создания цифровых двойников на базе облачных вычислительных систем был сделан вывод, что обе рассмотренные системы находятся в предварительной стадии, поэтому выбор используемой системы основан на объеме предоставляемых возможностей, количестве сопутствующих облачных инструментов и решений, наличии ограничений использования. При сравнении данных характеристик лучшими результатами обладает Azure Digital Twins от Microsoft. Поэтому в работе используется именно это решение.

На основании анализа Azure Digital Twins были определены следующие нефункциональные требования к проектируемой системе:

- функции первичной обработки телеметрии должны быть написаны на языке программирования JavaScript;
- доставка сообщений, содержащих показания устройств, а также клиентское приложение должны быть реализованы с использованием программной платформы .Net Framework;

### **2.3. Варианты использования системы**

В ходе анализа проектируемой системы были выявлены основные варианты использования, которые представлены на рисунке 4.

Были выделены следующие основные актеры, взаимодействующие с системой:

- *Администратор системы* – это пользователь системы, отвечающий за добавление и удаление пользователей и устройств. Также имеет возможность просматривать статистику потребления энергии другими пользователями системы;

– *Пользователь* – это владелец дома-участника проекта, имеющий доступ к результату обработки данных, полученных с его устройства.



Рис. 4. Диаграмма вариантов использования системы Цифровой двойник «Энергопотребление»

Для данных актеров были определены следующие основные варианты использования системы:

- *Добавить источник данных* – добавить в систему новое устройство или сенсор;
- *Добавить пользователя* – добавить в систему нового пользователя;
- *Посмотреть текущее энергопотребление* – предоставить пользователю информацию о текущей потребляемой им энергии.
- *Посмотреть среднее энергопотребление за промежуток времени* – предоставить пользователю информацию о потребленной им энергии за определенный промежуток времени;
- *Оповестить пользователя о неэффективном энергопотреблении*



– отправить уведомление пользователю по электронной почте при выполнении заранее определенного условия – превышения потребления заданного количества энергии;

Спецификация вариантов использования приведена в приложении 1.

### 3. АРХИТЕКТУРА СИСТЕМЫ

#### 3.1. Общее описание архитектуры системы

Проектируемая система в самом общем виде состоит из базы данных, хранящей общую статистику энергопотребления, службы Интернета вещей Digital Twins облачной платформы Microsoft Azure, работающей по принципу Platform as a Service (PaaS), обработчика телеметрии устройств – приложения, отвечающего за сбор, отправку и обработку телеметрии, а также клиентского Web-приложения для управления цифровым двойником, осуществляющего HTTP REST запросы к API созданного экземпляра Digital Twins. Диаграмма потоков данных проектируемой системы представлена на рисунке 5.



Рис. 5. Диаграмма потоков данных проектируемого цифрового двойника

Приложение, реализующее отправку и обработку телеметрии, выполняет обработку данных, полученных из генератора, а также их передачу на сервер. Оно состоит из нескольких компонентов: модуль создания пространственного интеллектуального графа, функция обработки телеметрии, выполняющая пользовательскую логику для данных, поступающих из устройств и обработчика данных с сервера для определения статуса помещения с целью проверки превышения рекомендуемых показателей энергопотребления для дальнейшей отправки уведомлений пользователям. Клиентское приложение состоит из модуля регистрации и аутентификации, а

также генератора HTTP REST запросов к API Digital Twins. Генератор телеметрии включает в себя базу данных «Общее потребление энергии». Диаграмму компонентов всей системы можно видеть на рисунке 6.

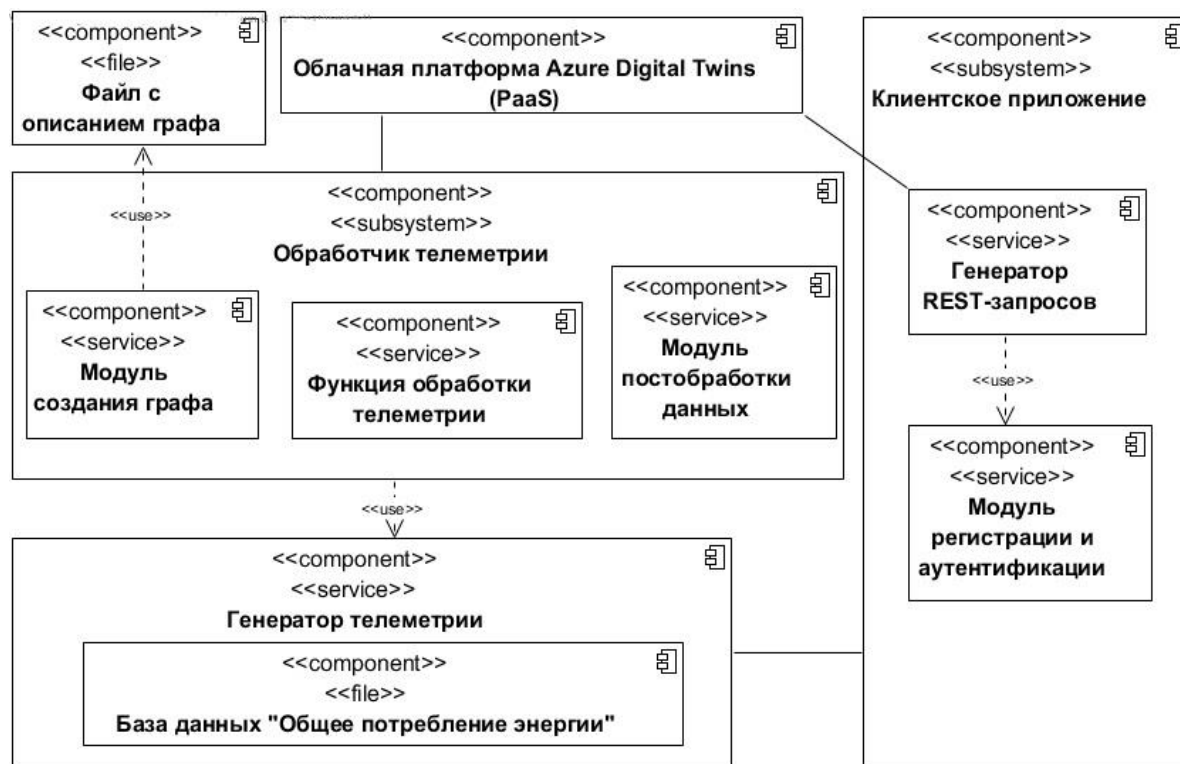


Рис. 6. Диаграмма компонентов цифрового двойника

### 3.2. Описание компонентов, составляющих систему

Разберем подробно отдельные компоненты, составляющие программную систему. Цифровой двойник состоит из следующих компонентов.

1. *Облачная платформа Azure Digital Twins (PaaS)* – это компонент системы, представляющий собой развернутый и настроенный экземпляр Azure Digital Twins.

2. *Обработчик телеметрии* – это компонент, отвечающий за обработку данных телеметрии и их отправку в облачную платформу Azure Digital Twins. Состоит из следующих элементов:

- *функция обработки телеметрии* – осуществляет первичную обработку данных с устройств до их отправки в центр Интернета вещей, когда выполняются условия, указанные сопоставителями, в частности, выявление

помещений, текущие показатели энергопотребления которых превышают заранее определенные рекомендуемые нормы энергопотребления;

- *модуль создания графа* – получает на вход *файл с описанием графа* сериализации данных формата YAML, в котором описана иерархия физических помещений и датчиков, установленных в них. В соответствии с объектной моделью производит серию POST-запросов к API Digital Twins для создания пространственного интеллектуального графа в соответствии с определенной схемой, а также возвращает строку подключения для дальнейшей работы с экземпляром цифрового двойника. Также возвращает пользователю код для регистрации устройства в Azure Active Directory [31], т.е. устройство, с которого планируется получить файл с иерархией графа и осуществлять сбор телеметрии, должно иметь на это право;

- *модуль постобработки данных* – производит обработку данных с устройств, определяя статус помещения на основании рекомендуемых данных для последующей передачи этих условий в Центры Событий [36] для отправки email-сообщений пользователям системы и администратору.

3. *Клиентское приложение* – реализует основную логику управления пространственным интеллектуальным графом, аутентификацию пользователей в системе, а также получение значений телеметрии из центра Интернета вещей. Клиентское приложение включает в себя следующие элементы:

- *модуль регистрации и аутентификации* – отвечает за вход пользователей в систему, а также использование OAuth-токена авторизации пользователя, от имени которого выполняется запрос [27]. Модуль принимает на вход токен авторизации (полученный с помощью стороннего ПО – Postman API [24]) и снабжает каждый запрос требуемыми заголовками авторизации;

- *генератор REST-запросов* – получает на вход набор идентификаторов устройств и помещений от пользователей. Путем отправки HTTP

REST-запросов к API облачной платформы получает и возвращает пользователю информацию о выбранных устройствах и помещениях, а также создает новые компоненты графа, то есть осуществляет управление пространственным интеллектуальным графом.

4. *Генератор телеметрии* – производит поиск показаний электропотребления в базе для каждого устройства, отправляет полученные данные в центр Интернета вещей [33], автоматический генерируемый облачной платформой Microsoft Azure при создании Azure Digital Twins. Включает в себя базу данных «Общее потребление электроэнергии». Это база данных, которая хранит статистику о потребляемой электроэнергии, используемые тарифные планы, личные данные пользователей для входа в систему, уникальные идентификаторы устройств, сенсоров и помещений.

### **3.3. Импорт показаний устройств и схема базы данных**

В качестве показателей сенсоров было принято решение использовать описанные выше данные проекта Low Carbon Project, представляющие собой файл формата CSV, содержащий 1 миллион строк. В результате импортирования CSV-файла и создания дополнительных таблиц был получен SQL-файл со всей базой.

В качестве системы управления базами данных был выбран Microsoft SQL Server 2017. Для работы с базой данных было использовано пространство имен System.Data.SqlClient для C#. Данное пространство имен является поставщиком данных платформы .Net для SQL Server.

Для хранения паролей, идентификаторов помещений, сенсоров и устройств потребовалось две дополнительных таблицы:

- registration – содержащая логины и пароли пользователей;
- spaces – хранит идентификаторы помещений, устройств и датчиков.

В итоге получаем базу данных, диаграмма «сущность – связь» которой изображена на рисунке 7.

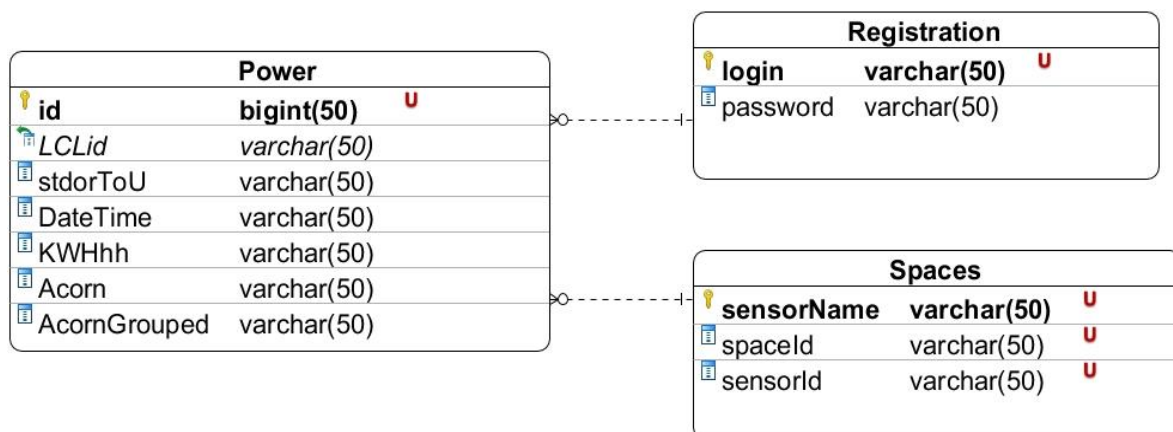


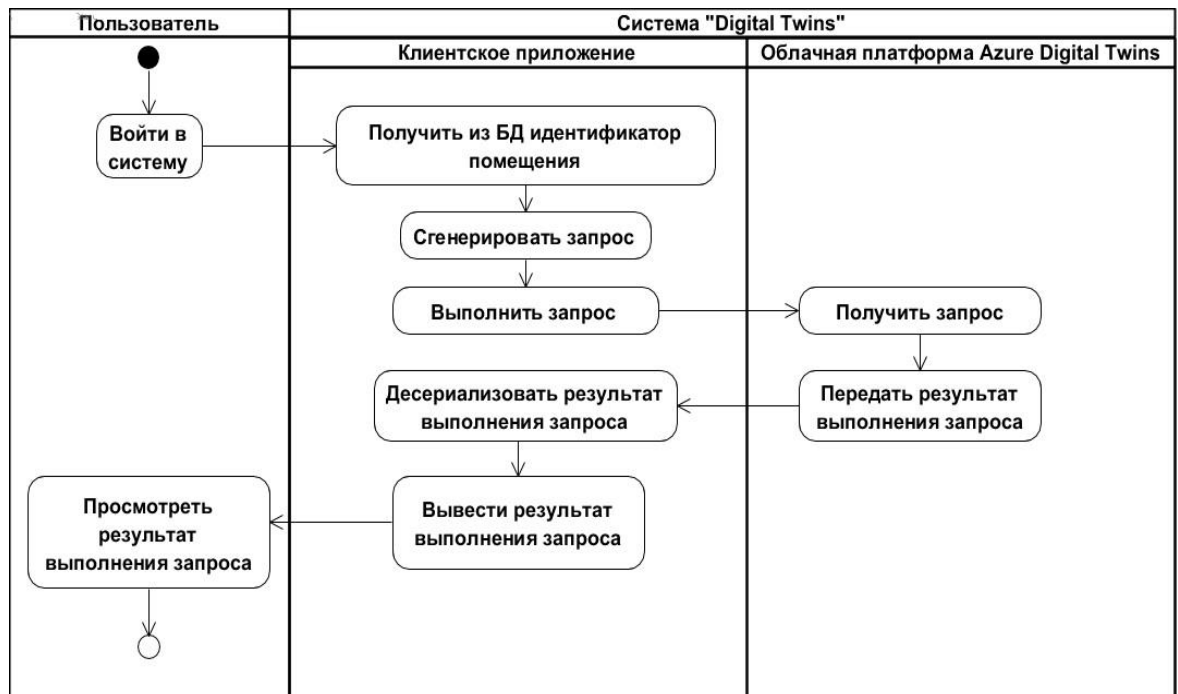
Рис. 7. Диаграмма «сущность – связь» базы данных цифрового двойника

### 3.4. Взаимодействие с API Digital Twins

На рисунке 8 представлена диаграмма деятельности, пошагово отражающая выполнение взаимодействия с API Digital Twins для получения информации о текущем энергопотреблении.

На первом шаге пользователь производит вход в систему с помощью модуля регистрации и аутентификации, предварительно получив токен авторизации с помощью настроенного для работы с используемым экземпляром Digital Twins клиента Postman API. На этом шаге модуль регистрации и аутентификации обращается к таблице Registration для проверки правильности данных, введенных пользователем. После этого пользователь с помощью графического интерфейса клиентского приложения инициирует генерацию REST-запроса к API Azure Digital Twins путем использования соответствующего элемента управления. Генератор REST-запросов обращается к таблице Spaces для получения идентификатора помещения, в котором требуется определить текущее значение энергопотребления. Далее происходит выполнение сгенерированного запроса. По-

сле этого API Digital Twins возвращает клиентскому приложению результат выполнения запроса в виде JSON-ответа, из которого извлекается нужная информация путем десериализации и выводится пользователю.



**Рис. 8.** Диаграмма деятельности системы для предоставления пользователю информации о текущем энергопотреблении

## 4. РЕАЛИЗАЦИЯ СИСТЕМЫ

### 4.1. Средства реализации

В качестве основного языка программирования для реализации системы был выбран язык C#, т.к. Azure Digital Twins в настоящий момент не поддерживает другие языки (не считая функции обработки телеметрии – они могут быть написаны на единственном поддерживаемом в данный момент для этих целей языке JavaScript, собственно, он и выбран для этих целей в качестве вспомогательного языка системы). В качестве среды разработки программного обеспечения была выбрана Microsoft Visual Studio Enterprise 2017.

Для отправки REST-запросов был использован HTTP REST клиент RestSharp для .Net. Он содержит все необходимые инструменты для работы с запросами. В качестве фреймворка для сериализации/десериализации был выбран фреймворк Json.NET от Newtonsoft.

В качестве регистратора событий был применен Центр событий Microsoft Azure. Для отправки email-сообщений использовался встроенный Gmail-клиент Azure.

### 4.2. Иерархия пространственного интеллектуального графа

Подготовка пространственного интеллектуального графа осуществляется с помощью файла конфигурации формата YAML, содержащего элементную структуру графа согласно выбранной онтологии. Для разработки системы была выбрана стандартная онтология VASnet, содержащая все основные типы данных, требующиеся в системе. Фрагмент YAML-файла конфигурации иерархии графа представлен в листинге 1.

Листинг 1. Фрагмент YAML-файла конфигурации иерархии графа

```
- name: Quickstart Building
  type: Venue
  resources:
  - type: IoTHub
  spaces:
  - name: Floor 3
    type: Floor
    spaces:
    - name: Focus Room 30
      type: Room
```



```

subType: FocusRoom
devices:
- name: Raspberry Pi 3 30
  hardwareId: 123456789030
  sensors:
  - dataType: Light
    hardwareId: MAC000030
matchers:
- name: Matcher Light
  dataTypeValue: Light
userdefinedfunctions:
- name: Motion Processor
  matcherNames:
  - Matcher Light
  script: actions/userDefinedFunctions/availability.js
roleassignments:
- roleId: 98e44ad7-28d4-4007-853b-b9968ad132d1 # System Role:
SpaceAdministrator
  objectName: Motion Processor
  objectIdType: UserDefinedFunctionId

```

Структура YAML-файла включает несколько следующих обязательных разделов:

- *Resources* – данный узел создает ресурс Центра Интернета вещей для взаимодействия с устройствами. Центр Интернета вещей в корневом узле графа может взаимодействовать со всеми устройствами и датчиками в графе.
- *Spaces* – в объектной модели Digital Twins данный узел представляет физические расположения. Каждое пространство имеет тип (Type) и имя (Name).
- *Devices* – это раздел, представляющий устройства, управляющие рядом датчиков.
- *Sensors* – данный раздел описывает используемые датчики.
- *Matchers* – это раздел, описывающий сопоставители – наборы конкретных условий, которые нужно отслеживать в данных устройств или датчиков. Сопоставитель будет отслеживать датчик типа `dataTypeValue`. Для подготовки сопоставителя, который будет отслеживать один из этих датчиков, его значение `dataTypeValue` должно соответствовать `dataType` этого датчика.

– *UserDefinedFunctions* – это узел, описывающий функции первичной обработки телеметрии. Они выполняют пользовательскую логику для данных, поступающих из пространств и устройств, когда выполняются условия, указанные сопоставителями.

### 4.3. База данных

В соответствии с разработанной схемой, была реализована база данных для хранения данных счетчиков, а также информации о пользователях. Для реализации была использована СУБД Microsoft SQL Server 2017. SQL-скрипт создания таблиц представлен в листинге 2.

#### Листинг 2. SQL-скрипт создания таблиц в базе данных

```
CREATE TABLE [Power] (
    id bigint(50) NOT NULL,
    LCLid varchar(50) NOT NULL,
    stdorToU varchar(50) NOT NULL,
    DateTime varchar(50) NOT NULL,
    KWH hh varchar(50) NOT NULL,
    Acorn varchar(50) NOT NULL,
    Acorn Grouped varchar(50) NOT NULL,
    CONSTRAINT [PK_POWER] PRIMARY KEY CLUSTERED
    (
        [id] ASC
    ) WITH (IGNORE_DUP_KEY = OFF)
)
GO
CREATE TABLE [registration] (
    login varchar(50) NOT NULL,
    password varchar(50) NOT NULL,
    CONSTRAINT [PK_REGISTRATION] PRIMARY KEY CLUSTERED
    (
        [login] ASC
    ) WITH (IGNORE_DUP_KEY = OFF)
)
GO
CREATE TABLE [spaces] (
    spaceId varchar(50) NOT NULL,
    sensorId varchar(50) NOT NULL,
    sensorName varchar(50) NOT NULL,
    CONSTRAINT [PK_SPACES] PRIMARY KEY CLUSTERED
    (
        [spaceId] ASC
    ) WITH (IGNORE_DUP_KEY = OFF)
)
GO
ALTER TABLE [registration] WITH CHECK ADD CONSTRAINT [registration_fk0] FOREIGN KEY ([login]) REFERENCES [Power]([LCLid])
ON UPDATE CASCADE
```

```

GO
ALTER TABLE [registration] CHECK CONSTRAINT [registration_fk0]
GO

ALTER TABLE [spaces] WITH CHECK ADD CONSTRAINT [spaces_fk0] FOREIGN KEY
([sensorName]) REFERENCES [Power] ([LCLid])
ON UPDATE CASCADE
GO
ALTER TABLE [spaces] CHECK CONSTRAINT [spaces_fk0]
GO

```

#### 4.4. Реализация отправки телеметрии и ее обработки

В приложении 2 представлена реализация функции отправки телеметрии в центр Интернета вещей, написанная на языке C#.

В листинге 3 приведена реализация первичной обработки телеметрии до отправки в центр Интернета вещей, написанная на языке JavaScript.

##### Листинг 3. Реализация обработки телеметрии

```

var Type = "Light";
var StatusOk = "OK";
var ElectricityThreshold = 2.0;

function process(telemetry, executionContext) {

    try {
        log(`Sensor ID: ${telemetry.SensorId}. `);
        log(`Sensor value: ${JSON.stringify(telemetry.Message)}.`);
        var sensor = getSensorMetadata(telemetry.SensorId);
        var parseReading = JSON.parse(telemetry.Message);
        setSensorValue(telemetry.SensorId, sensor.DataType, parseReading.SensorValue);
        var parentSpace = sensor.Space();
        var otherSensors = parentSpace.ChildSensors();
        var ElectricitySensor = otherSensors.find(function(element) {
            return element.DataType === motionType;
        });
        var ElectricityValue = getFloatValue(ElectricitySensor.Value().Value);
        var roomIsOk = "Room is OK";
        var roomIsNotOk = "Room is not OK";
        if(ElectricityValue < ElectricityThreshold) {
            log(`${roomIsOk}. Electricity: ${ElectricityValue}.`);
            setSpaceValue(parentSpace.Id, StatusOk, roomIsOk);
        }
        else {
            log(`${roomIsNotOk}. Electricity: ${ElectricityValue}.`);
            setSpaceValue(parentSpace.Id, StatusOk, roomIsNotOk);
            parentSpace.Notify(JSON.stringify(roomIsNotOk));
        }
    }
    catch (error)
    {
        log(`error: ${error.name} Message ${error.message}.`);
    }
}

function getFloatValue(str) {
    if(!str) {

```

}

#### 4.5. Реализация графического интерфейса пользователя

Согласно функциональным требованиям клиентское приложение системы управления цифровым двойником предоставляет пользователю интерфейс, состоящий из двух окон: окна регистрации и окна получения результатов запросов. Окно регистрации предоставляет пользователю интерфейс входа в систему. Окно получения результатов запросов предоставляет пользователю интерфейс выбора интересующего его помещения, а также интерфейс для получения информации о текущем энергопотреблении и энергопотреблении за определенный промежуток времени в данном помещении.

Интерфейс клиентского приложения системы представлен на рисунках 9–10.

|           |   |
|-----------|---|
| Bearer    | eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImhCeGw5bUFINmd4YXZDa2NvT1UyVEhzRE5hMCIsImtpZCI6ImhCeGw5bUFINmQxLWUiOjEudDk1InRlc2VybnQiOiJkaWEyZW5ja2F1c2lvbiJ9. |
| Login:    |   |
|           | MAC000006   |
| Password: |   |
|           | *****   |
|           | <button>Submit</button>   |
| Status    |   |

Рис. 9. Окно регистрации

API caller

GetCurrentValue

0.06

GetHistoricalValues

0,05760416

GetRoomStatus

OK

Рис. 10. Окно получения результатов запросов

## **5. ТЕСТИРОВАНИЕ**

Тестирование предложенной системы было проведено в несколько этапов: тестирование работоспособности системы на основе модульных тестов, проверяющих статусы ответов к осуществляемым к пространственному графу API-запросам, сравнительное тестирование с целью определения правильности обработки телеметрии датчиков, а также тестирование API управления системы, то есть правильности результатов выполнения API-запросов, выполняемых системой.

### **5.1. Тестирование работоспособности системы**

Для проверки работоспособности системы было написан модульный тест, проверяющий правильность работы компонента, отвечающего за создание пространственного интеллектуального графа на основе YAML-файла, содержащего иерархическую структуру моделируемой физической системы. Тест проверяет статусы ответов сервера к осуществляемым системой API-запросам, а также обработку исключительных ситуаций. Исходный код тестирования правильности создания сенсоров системы показан в приложении 3.

### **5.2. Сравнительное тестирование с целью определения правильности обработки телеметрии датчиков**

Для проверки правильности предоставленных системой текущих и суточных показателей было проведено сравнение наборов данных, полученных напрямую из таблицы, содержащей все показания датчиков и данных, предоставленных пользователю системы по запросу, то есть полученных после обработки системой. Результат сравнения приведен в таблице 2.

Табл. 2. Результат сравнения данных

| Тестовый случай   | Входные данные | Ожидаемый результат | Полученный результат | Тест пройден? |
|---|----------------|---------------------|----------------------|---------------|
| Получить текущее значение сенсора “MAC000003”                                   | -              | 0.452               | 0.452                | Да            |
| Получить показатель среднего энергопотребления в помещении “MAC000012” за сутки | -              | 0.335               | 0.335                | Да            |

### 5.3. Тестирование API управления

Для проверки правильности результатов выполнения API-запросов был использован упомянутый выше REST клиент Postman API [24], предварительно настроенный для работы с приложением Azure Active Directory, предоставляющим неявный поток разрешений OAuth 2.0. POST-клиент использовался для выполнения HTTP-запросов, содержащих токены к API управления экземпляра Digital Twins, а также для составления составных запросов POST в API управления. Результат тестирования API управления с помощью Postman API приведен в приложении 4.

## **ЗАКЛЮЧЕНИЕ**

В рамках дипломного проекта была разработана технология создания цифровых двойников, а также создан прототип цифрового двойника на основе ресурсов облачной вычислительной платформы Microsoft Azure. Также было разработано клиентское приложения для работы с цифровым двойником. В ходе разработки были решены следующие задачи:

- произведен обзор научной литературы и существующих решений Интернета вещей для создания цифровых двойников;
- изучены технологий, предоставляемые облачными вычислительными платформами, для создания цифровых двойников;
- спроектирован и разработан прототип цифрового двойника на основе облачных вычислительных ресурсов вычислительной платформы Microsoft Azure;
- произведено тестирование разработанного прототипа.



## ЛИТЕРАТУРА

1. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. Lee J., Bagheri B., Kao H. // Manufacturing Letters, 2015. – Vol. 3. – P. 18–23.
2. A smart manufacturing use case: Furnace temperature balancing in steam methane reforming process via kepler workflows. Korambath P., Wang J., Kumar A., Davis J., Graybill R., Schott B., Baldea M. // Procedia Comput. Sci., 2016 – Vol. 80. – P. 680–689.
3. Amazon Web Services Management Console. [Электронный ресурс] URL: <https://aws.amazon.com/console/> (дата обращения: 10.02.2019).
4. AWS IoT Core Features. [Электронный ресурс] URL: <https://aws.amazon.com/ru/iot-core/features/> (дата обращения: 18.04.2019).
5. AWS IoT Greengrass Features. [Электронный ресурс] URL: <https://aws.amazon.com/ru/greengrass/features/> (дата обращения: 18.04.2019).
6. AWS IoT Things Graph Documentation. [Электронный ресурс] URL: [https://docs.aws.amazon.com/en\\_us/thingsgraph/latest/ug/iot-tg-what-is.html/](https://docs.aws.amazon.com/en_us/thingsgraph/latest/ug/iot-tg-what-is.html/) (дата обращения: 13.02.2019).
7. AWS IoT Things Graph Overview. [Электронный ресурс] URL: <https://aws.amazon.com/ru/iot-things-graph/> (дата обращения: 10.02.2019).
8. Azure Digital Twins Documentation. [Электронный ресурс] URL: <https://docs.microsoft.com/en-us/azure/digital-twins/> (дата обращения: 11.02.2019).
9. Azure Digital Twins Overview. [Электронный ресурс] URL: <https://azure.microsoft.com/en-us/services/digital-twins/> (дата обращения: 10.02.2019).
10. Azure Event Hubs. [Электронный ресурс] URL: <https://azure.microsoft.com/ru-ru/services/event-hubs/> (дата обращения: 16.04.2019).
11. Azure IoT Hub Documentation. [Электронный ресурс] URL: <https://docs.microsoft.com/ru-ru/azure/iot-hub/> (дата обращения: 16.04.2019).

12. Bar-Magen Numhauser, J. Fog Computing introduction to a New Cloud Evolution. // Proceedings from the Cies III Congress, January 2012. University of Alcala. – P. 111–126.
13. Boyes H., Hallaq B., Cunningham J., Watson T. The industrial internet of things (IIoT): An analysis framework. // Computers in Industry, October 2018. – Vol. 101. – P. 1–12.
14. Digital Twins Service Management APIs. [Электронный ресурс] URL: <https://azuredigitaltwinsexample.northeurope.azuremartspaces.net/management/swagger/ui/index/> (дата обращения: 12.03.2019).
15. Fog Computing and Its Role in the Internet of Things. Bonomi F., Milito R., Zhu J., Addepalli S. // Proceedings of ACM MCC, 2012. – P. 13–16.
16. IDC Forecasts Worldwide Spending on the Internet of Things in 2019. [Электронный ресурс] URL: <https://www.idc.com/getdoc.jsp?containerId=prUS44596319/> (дата обращения: 04.02.2019).
17. Internet of Things. Gartner IT glossary. Gartner (5 May 2012). [Электронный ресурс] URL: <https://www.gartner.com/it-glossary/internet-of-things/> (дата обращения: 04.02.2019).
18. K. Rose, S. Eldridge, L. Chapin The internet of things: an overview Internet Soc. (2015). – P. 7.
19. Low Carbon London webpage. [Электронный ресурс] URL: [http://innovation.ukpowernetworks.co.uk/innovation/en/Projects/tier-2-projects/Low-Carbon-London-\(LCL\)/](http://innovation.ukpowernetworks.co.uk/innovation/en/Projects/tier-2-projects/Low-Carbon-London-(LCL)/) (дата обращения: 16.04.2019).
20. Mell, P., Grance, T. The NIST Definition of Cloud Computing. // Recommendations of the National Institute of Standards and Technology. NIST (20 October 2011).
21. Microsoft Azure Digital Twins Graph Viewer Github page. [Электронный ресурс] URL: <https://github.com/Azure/azure-digital-twins-graph-viewer/> (дата обращения: 08.03.2019).

22. Microsoft Azure Portal. [Электронный ресурс] URL: <https://portal.azure.com/> (дата обращения: 10.02.2019).
23. Micro-Workflows: Kafka and Kepler fusion to support Digital Twins of Industrial Processes. Radchenko G., Alaasam A., Tchernykh A. // IEEE/ACM Int. Conf. Util. Cloud Comput. – UCC '18, pp. 83-88, December 2018.
24. Postman API Requests. [Электронный ресурс] URL: [https://learning.getpostman.com/docs/postman/sending\\_api\\_requests/requests/](https://learning.getpostman.com/docs/postman/sending_api_requests/requests/) (дата обращения: 07.04.2019).
25. SmartMeter Energy Consumption Data in London Households. [Электронный ресурс] URL: <https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households/> (дата обращения: 27.03.2019).
26. Towards Digital Twins Cloud Platform: Microservices and Computational Workflows to Rule a Smart Factory. Borodulin K., Radchenko G., Shestakov A., Sokolinsky L., Tchernykh A., Prodan R. // Proc. The 10th Int. Conf. Util. Cloud Comput. – UCC '17, pp. 209–210, December 2017.
27. Авторизация доступа к веб-приложениям Azure Active Directory с помощью потока предоставления кода OAuth 2.0. [Электронный ресурс] URL: <https://docs.microsoft.com/ru-ru/azure/active-directory/develop/v1-protocols-oauth-code/> (дата обращения: 19.04.2019).
28. Барсков А. IoT как инструмент цифровой экономики. Журнал сетевых решений/LAN. [Электронный ресурс] URL: <https://www.osp.ru/lan/2017/05/13052169/> (дата обращения: 26.02.2019).
29. Барсков А. Промышленный Интернет вещей. Готовы ли сети? Журнал сетевых решений/LAN. [Электронный ресурс] URL: <https://www.osp.ru/lan/2016/09/13050308/> (дата обращения: 26.02.2019).
30. Грингард С. Интернет вещей: Будущее уже здесь. – М.: Альпина Паблишер, 2016. – 188 с.
31. Документация Azure Active Directory. [Электронный ресурс] URL: <https://docs.microsoft.com/ru-ru/azure/active-directory/> (дата обращения: 16.03.2019).

32. Документация по Microsoft Azure. Microsoft Docs. [Электронный ресурс]  
URL: <https://docs.microsoft.com/ru-ru/azure/> (дата обращения: 04.02.2019).
33. Документация по центру Интернета вещей. [Электронный ресурс] URL:  
<https://docs.microsoft.com/ru-ru/azure/iot-hub/> (дата обращения: 18.03.2019).
34. Основные сведения об объектных моделях и пространственном интеллектуальном графе в Digital Twins. [Электронный ресурс] URL:  
<https://docs.microsoft.com/ru-ru/azure/digital-twins/concepts-objectmodel-spatialgraph/> (дата обращения: 28.03.2019).
35. Семеновская Е. Индустриальный Интернет вещей. Перспективы российского рынка. [Электронный ресурс] URL:  
[https://www.company.rt.ru/projects/IIoT/study\\_IDC.pdf/](https://www.company.rt.ru/projects/IIoT/study_IDC.pdf/) (дата обращения: 27.02.2019).
36. Центры событий Microsoft Azure. Документация. [Электронный ресурс]  
URL: <https://docs.microsoft.com/ru-ru/azure/event-hubs/> (дата обращения: 19.04.2019).

## ПРИЛОЖЕНИЯ

### Приложение 1

#### Спецификация вариантов использования

|  |
|--|
| <i>UseCase:</i> Добавить источник данных   |
| <i>ID:</i> 1   |
| <i>Аннотация:</i> выполнить запрос HTTP POST, инициирующий создание нового устройства с одним подключенным сенсором в выбранном помещении  |
| <i>Главные актеры:</i> 1   |
| <i>Второстепенные актеры:</i> -  |
| <i>Предусловия:</i> администратор предоставил идентификатор помещения, в котором требуется создать источник данных   |
| <i>Основной поток:</i> <ol style="list-style-type: none"><li>1. Выполнить POST-запрос к Web API Digital Twins, инициирующий создание устройства в помещении с предоставленным идентификатором</li><li>2. Получить от сервера ответ формата JSON, десериализовать его</li><li>3. Внести идентификатор созданного устройства в базу данных</li><li>4. Вернуть администратору идентификаторы сенсора и устройства</li></ol> |
| <i>Постусловия:</i> -  |
| <i>Альтернативные потоки:</i> -  |
|  |
| <i>UseCase:</i> Добавить пользователя  |
| <i>ID:</i> 2   |
| <i>Аннотация:</i> выполнить запрос HTTP POST, инициирующий создание нового пользователя  |

|  |
|--|
| <i>Главные актеры:</i> 1   |
| <i>Второстепенные актеры:</i> -  |
| <i>Предусловия:</i> -  |
| <i>Основной поток:</i> <ol style="list-style-type: none"> <li>1. Выполнить POST-запрос к Web API Digital Twins, инициирующий создание нового помещения</li> <li>2. Получить от сервера ответ формата JSON, десериализовать его</li> <li>3. Сгенерировать пароль для входа нового пользователя в систему</li> <li>4. Внести в базу данных идентификатор созданного помещения и пароль для входа</li> <li>5. Вернуть администратору идентификатор созданного помещения и пароль для входа нового пользователя</li> </ol> |
| <i>Постусловия:</i> -  |
| <i>Альтернативные потоки:</i> -  |
|  |
| <i>UseCase:</i> Оповестить пользователя о неэффективном энергопотреблении  |
| <i>ID:</i> 3   |
| <i>Аннотация:</i> при достижении заранее определенных условий, уведомить пользователя системы и администратора о неэффективном энергопотреблении в одном из помещений  |
| <i>Главные актеры:</i> 2   |
| <i>Второстепенные актеры:</i> -  |
| <i>Предусловия:</i> на этапе обработки данных, полученных с устройств, было выявлено, что текущие значения расхода электроэнергии в одном из помещений превышают рекомендуемые   |
| <i>Основной поток:</i>   |

|   |
|---|
| 1. Активировать событие, отправляющее email-сообщения администратору системы и владельцу выявленного помещения  |
| <i>Постусловия:</i> -   |
| <i>Альтернативные потоки:</i> -   |
|   |
| <i>UseCase:</i> Посмотреть среднее энергопотребление за промежуток  |
| <i>ID:</i> 4  |
| <i>Аннотация:</i> предоставить пользователю информацию о среднем энергопотреблении в выбранном помещении  |
| <i>Главные актеры:</i> 2  |
| <i>Второстепенные актеры:</i> -   |
| <i>Предусловия:</i> пользователь системы выбрал помещение и интересующий его промежуток времени для получения информации о среднем энергопотреблении  |
| <i>Основной поток:</i> <ol style="list-style-type: none"> <li>1. Выполнить запрос HTTP GET к Web API Digital Twins для получения выбранного в зависимости от промежутка времени количества значений HistoricalValues компонента Sensor, принадлежащего выбранному помещению</li> <li>2. Вернуть пользователю среднее значение HistoricalValue компонента Sensor на заданном промежутке времени</li> </ol> |
| <i>Постусловия:</i> -   |
| <i>Альтернативные потоки:</i> -   |
|   |
| <i>UseCase:</i> Посмотреть текущее энергопотребление  |
| <i>ID:</i> 5  |
| <i>Аннотация:</i> предоставить пользователю информацию о текущем энергопотреблении в выбранном помещении  |

|  |
|--|
| <i>Главные актеры: 2</i>   |
| <i>Второстепенные актеры: -</i>  |
| <i>Предусловия:</i> пользователь системы выбрал помещение для получения информации о текущем энергопотреблении   |
| <p><i>Основной поток:</i></p> <ol style="list-style-type: none"> <li>1. Выполнить запрос HTTP GET к Web API Digital Twins для получения значения Value компонента Sensor, принадлежащего выбранному помещению</li> <li>2. Вернуть пользователю текущее значение Value компонента Sensor</li> </ol> |
| <i>Постусловия: -</i>  |
| <i>Альтернативные потоки: -</i>  |



## Приложение 2

### Реализация отправки телеметрии

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net.NetworkInformation;
using System.Runtime.Serialization.Json;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Azure.Devices.Client;
using Microsoft.Azure.DigitalTwins.Samples.Models;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Configuration.Binder;
using System.Data.SqlClient;
using System.Data;

namespace Microsoft.Azure.DigitalTwins.Samples
{
    public static class DBConnector
    {
        public static int GetShift(Dictionary<string, int> dict, string id)
        {
            int index = 0;
            foreach (var el in dict)
            {
                if (el.Key == id)
                {
                    index = el.Value;
                    return index;
                }
            }
            return index;
        }
        public static int GetIndex(DataTable dt, string id)
        {
            for (int i = 0; i < dt.Rows.Count; i++)
            {
                if (dt.Rows[i][0].ToString() == id)
                {
                    return i;
                }
            }
            return -1;
        }
        public static DataTable Connect(string sql, string connection-
String)
        {
            SqlConnection connection = new SqlConnection(connection-
String);
            DataTable dt = new DataTable();
            SqlDataAdapter adapter = new SqlDataAdapter(sql, connection);

            using (connection)
            {
                connection.Open();
                DataTable ds = new DataTable();
                new SqlDataAdapter(sql, connection).Fill(ds);
                dt = ds;
            }
        }
    }
}
```

```

        }
        return dt;
    }

}

class Program
{
    static Dictionary<string, int> dict = new Dictionary<string,
int>();
    static string sql_dt = "select * from [Power-Networks-LCL-
June2015(withAcornGps)v2_1] as tbl order by tbl.LCLid asc, tbl.DateTime
asc";
    static string connectionString_dt = @"Data Source=.\SQLEXPRESS;In-
itial Catalog=Electricity;Integrated Security=True";
    private static DataTable dt = DBConnector.Connect(sql_dt, connec-
tionString_dt);

    static string sql_reg = "select * from [registration] as reg order
by reg.login asc";
    static string connectionString_reg = @"Data Source=.\SQLEX-
PRESS;Initial Catalog=Electricity;Integrated Security=True";
    private static DataTable reg = DBConnector.Connect(sql_reg, con-
nectionString_reg);

    private static IConfigurationSection settings;
    static void Main(string[] args)
    {
        foreach (var el in dict)
            Console.WriteLine(el);

        for (int i = 0; i < reg.Rows.Count; i++)
            dict.Add(reg.Rows[i][1].ToString(), DBConnector.Get-
Index(dt, reg.Rows[i][1].ToString()));

        if (args.Length != 0)
        {
            Console.WriteLine("Usage: dotnet run\nNo arguments are
supported");
            return;
        }

        settings = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("appsettings.json")
            .Build()
            .GetSection("Settings");

        try
        {
            DeviceClient deviceClient = DeviceClient.CreateFromConnec-
tionString(settings["DeviceConnectionString"]);

            if (deviceClient == null)
            {
                Console.WriteLine("ERROR: Failed to create DeviceCli-
ent!");
                return;
            }

            SendEvent(deviceClient).Wait();
        }
        catch (Exception ex)
        {

```

```

        Console.WriteLine("EXIT: Unexpected error: {0}", ex.Mes-
sage);
    }

    }

    static Func<string> CreateGetRandomSensorReading(int iteration,
int shift)
    {
        string temp = dt.Rows[iteration + shift][3].ToString();
        return () => temp.ToString(CultureInfo.InvariantCulture);
    }

    static async Task SendEvent(DeviceClient deviceClient)
    {
        var serializer = new DataContractJsonSerializer(typeof(Cus-
tomTelemetryMessage));

        var sensors = settings.GetSection("Sensors").Get<Sensor[]>();

        var delayPerMessageSend = int.Parse(settings["MessageInterva-
lInSeconds"]);
        var countOfSendsPerIteration = sensors.Length;
        var maxSecondsToRun = 15 * 60;
        var maxIterations = maxSecondsToRun / countOfSendsPerIteration
/ delayPerMessageSend;
        var curIteration = 0;

        do {
            foreach (var sensor in sensors)
            {
                var getRandomSensorReading = CreateGetRandomSen-
sorReading(curIteration, DBConnector.GetShift(dict, sensor.Hard-
wareId.ToString()));
                var telemetryMessage = new CustomTelemetryMessage()
                {
                    SensorValue = getRandomSensorReading(),
                };

                using (var stream = new MemoryStream())
                {
                    serializer.WriteObject(stream, telemetryMessage);
                    var byteArray = stream.ToArray();
                    Message eventMessage = new Message(byteArray);
                    eventMessage.Properties.Add("DigitalTwins-Teleme-
try", "1.0");
                    eventMessage.Properties.Add("DigitalTwins-Senso-
rHardwareId", $"{sensor.HardwareId}");
                    eventMessage.Properties.Add("CreationTimeUtc",
DateTime.UtcNow.ToString("o"));
                    eventMessage.Properties.Add("x-ms-client-request-
id", Guid.NewGuid().ToString());

                    Console.WriteLine($"{DateTime.UtcNow.ToLocal-
Time()}> Sending message: {Encoding.UTF8.GetString(eventMessage.Get-
Bytes())} Properties: {{ {eventMessage.Properties.Aggregate(new String-
Builder(), (sb, x) => sb.Append($"{x.Key}': '{x.Value}',"), sb =>
sb.ToString()) } }}");

                    await deviceClient.SendEventAsync(eventMessage);
                }
            }
        }
    }
}

```

```

        await Task.Delay(TimeSpan.FromSeconds(delayPerMessageSend));

        } while (++curIteration < maxIterations);

        Console.WriteLine($"Finished sending {curIteration} events
(per sensor type)");
    }

    public class Sensor
    {
        public string DataType { get; set; }
        public string HardwareId { get; set; }
    }
}

```

## Приложение 3

### Исходный код тестирования модуля создания сенсоров

```
using System;
using System.Linq;
using Xunit;
using Microsoft.Azure.DigitalTwins.Samples;
using System.Net.Http;
using System.Net;
using System.Threading.Tasks;
using Newtonsoft.Json;
using System.Collections.Generic;
using Moq;
using System.IO;
using System.Collections;
using YamlDotNet.Serialization;
using Microsoft.Extensions.Logging;

namespace Microsoft.Azure.DigitalTwins.Samples.Tests
{
    public class ProvisionSampleSensorsTests
    {
        private static Serializer yamlSerializer = new Serializer();
        private static Guid sensor1Guid = new Guid("00000000-0000-0000-0000-000000000001");
        private static Guid sensor2Guid = new Guid("00000000-0000-0000-0000-000000000002");

        [Fact]
        public async Task GetProvisionSampleCreatesDescriptions()
        {
            var yaml = @"
- name: Test1
  devices:
    - name: Device1
      hardwareId: DeviceHardwareId1
      sensors:
        - dataType: SensorType1
          hardwareId: SensorHardwareId1
";
            var expectedDescriptions = new [] { new SpaceDescription()
            {
                name = "Test1",
                devices = new [] {
                    new DeviceDescription()
                    {
                        name = "Device1",
                        hardwareId = "DeviceHardwareId1",
                        sensors = new [] {
                            new SensorDescription()
                            {
                                dataType = "SensorType1",
                                hardwareId = "SensorHardwareId1",
                            }
                        }
                    }
                },
            }
        };
            var actualDescriptions = await Actions.GetProvisionSampleTopology(new StringReader(yaml));
            Assert.Equal(yamlSerializer.Serialize(expectedDescriptions),
                yamlSerializer.Serialize(actualDescriptions));
        }
    }
}
```

```

    }

    [Fact]
    public async Task CreateTwoSensors()
    {
        (var httpClient, var httpHandler) = FakeDigitalTwinsHttpClient.CreateWithDevice(
            postResponseGuids: new [] { sensor1Guid, sensor2Guid },
            getResponses: Enumerable.Repeat(Responses.NotFound, 2)
        );

        var descriptions = new [] { new SpaceDescription()
        {
            name = FakeDigitalTwinsHttpClient.Space.Name,
            devices = new [] {
                new DeviceDescription()
                {
                    name = FakeDigitalTwinsHttpClient.Device.Name,
                    hardwareId = FakeDigitalTwinsHttpClient.Device.HardwareId,
                    sensors = new [] {
                        new SensorDescription()
                        {
                            dataType = "SensorType1",
                            hardwareId = "SensorHardwareId1",
                        },
                        new SensorDescription()
                        {
                            dataType = "SensorType2",
                            hardwareId = "SensorHardwareId2",
                        }
                    }
                }
            }
        },
        };

        await Actions.CreateSpaces(httpClient, Loggers.SilentLogger,
            descriptions, Guid.Empty);
        Assert.Equal(2, httpHandler.PostRequests["sensors"].Count);
    }
}

```

## Приложение 4

### Протокол тестирования API управления цифровым двойником

| Тестовый случай  | Выполненный запрос  | Полученный ответ   | Тест пройден? |
|--|---|--|---------------|
| Получить характеристики помещения с названием "Focus Room 02"                                | <a href="https://electricitydigitaltwin.northeurope.azuremartspaces.net/management/api/v1.0/spaces?name=Focus Room 02&amp;includes=values">https://electricitydigitaltwin.northeurope.azuremartspaces.net/management/api/v1.0/spaces?name=Focus Room 02&amp;includes=values</a>               | [<br>{<br>"id": "6553c267-6519-4561-9b70-1d14e139ec07",<br>"name": "Focus Room 02",<br>"typeId": 14,<br>"parentSpaceId": "2dfb2418-a2a6-4f59-b7ca-b4ec394e18e8",<br>"subTypeId": 13,<br>"statusId": 12<br>}<br>]                                       | Да            |
| Получить список устройств, принадлежащих помещению с ID 8b73d652-4fa3-46cb-8ac2-b27bfe3c10f6 | <a href="https://electricitydigitaltwin.northeurope.azuremartspaces.net/management/api/v1.0/devices?spaceId=8b73d652-4fa3-46cb-8ac2-b27bfe3c10f6">https://electricitydigitaltwin.northeurope.azuremartspaces.net/management/api/v1.0/devices?spaceId=8b73d652-4fa3-46cb-8ac2-b27bfe3c10f6</a> | [<br>{<br>"name": "Raspberry Pi 3 19",<br>"typeId": 2,<br>"subTypeId": 1,<br>"hardwareId": "123456789019",<br>"spaceId": "8b73d652-4fa3-46cb-8ac2-b27bfe3c10f6",<br>"status": "Provisioned",<br>"id": "ad6921b8-813f-45d8-ac5c-8892efc45e52"<br>}<br>] | Да            |
| Получить информацию об определяемой пользователем функции                                    | <a href="https://electricitydigitaltwin.northeurope.azuremartspaces.net/management/api/v1.0/userdefinedfunctions">https://electricitydigitaltwin.northeurope.azuremartspaces.net/management/api/v1.0/userdefinedfunctions</a>   | [<br>{<br>"id": "1f1dd40e-5fc1-465f-af05-76ee0ce232f6",<br>"spaceId": "439ee6ea-d477-4f50-8c02-f92babea3fde",<br>"name": "Motion Processor",<br>"disabled": false<br>}<br>]  | Да            |