



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РГУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных
технологий

Отчет по практическим работам №5-8

по дисциплине «Технологические основы Интернета вещей»

Выполнили:

Студенты группы ИКБО-20-23

Кузнецов Л.А.
Комисарик М.А.

Проверил:

Жматов Дмитрий Владимирович

2025 г.

ОГЛАВЛЕНИЕ

1. Практическая работа №5	3
2. Практическая работа №6	13
3. Практическая работа №7	14
4. Практическая работа №8	22
5. Дополнительные задания	26
ВЫВОД.....	32

ПРАКТИЧЕСКАЯ ЧАСТЬ

1. Практическая работа №5

Часть 1

Таблица 1 – Описание счетчика электроэнергии Энергомера CE102 R5.1

Параметр	Описание
1. Название датчика/устройства	Счетчик электроэнергии Энергомера CE102 R5.1
2. Тип измерения	Цифровой
3. Измеряемые параметры и диапазон	Действующее напряжение 172.5 - 264.5 В Действующий ток 0.01 - 60 А Активная мощность 0 - 13.8 кВт Активная энергия от 0 до 9999999.99 кВт*ч
4. Точность /Погрешность	±1,0%
5. Напряжение питания	230 В
6. Уникальный идентификатор в веб-интерфейсе	/devices/energy_meter
7. Протокол передачи данных	CE
8. Интерфейс управления (шина)	RS-485
9. Описание входов/выходов, схема подключения	Цифровой сигнал по шине RS-485 https://www.energomera.ru/documentations/product/ce102r5_1_rp.pdf п. 4.5 Схемы подключения

Таблица 2 – Лента светодиодная RGBW

Параметр	Описание
1. Название датчика/устройства	Диммер светодиодных лент на DIN-рейку WB-LED
2. Тип измерения	Исполнительное устройство
3. Измеряемые параметры и диапазон	Яркость светодиодной ленты 0-100% Цвет светодиодной ленты {r:0-255, g:0-255, b:0-255} Напряжение на нагрузке: до 48 В

	Ток на канал: до 5 А Частота ШИМ: 100 Гц - 24 кГц
4. Точность /Погрешность	-
5. Напряжение питания	9 - 28 В
6. Уникальный идентификатор в веб-интерфейсе	/devices/wb-led_39
7. Протокол передачи данных	Modbus RTU
8. Интерфейс управления (шина)	RS-485
9. Описание входов/выходов, схема подключения	4 универсальных дискретных входа для подключения выключателей. Схема подключения: https://wiki.wirenboard.com/wiki/WB-LED_v.1_Modbus_LED_Dimmer

Таблица 3 – Описание датчика протечки

Параметр	Описание
1. Название датчика/устройства	Модуль учета водопотребления и контроля протечек WB-MWAC v.2
2. Тип измерения	Исполнительное устройство
3. Измеряемые параметры и диапазон	Состояние протечки Режим модуля
4. Точность /Погрешность	-
5. Напряжение питания	9 - 28 В
6. Уникальный идентификатор в веб-интерфейсе	/devices/wb-mwac-v2_25
7. Протокол передачи данных	Modbus RTU
8. Интерфейс управления (шина)	RS-485
9. Описание входов/выходов, схема подключения	Дискретные входы: F1-5, S6, P1-2. Схема подключения: https://wiki.wirenboard.com/wiki/WB-MWAC_v.2_Modbus_Water_Consumption_Metering_and_Leak_Monitoring

Часть 2

1. Modbus RTU

Modbus – коммуникационный протокол, основан на архитектуре ведущий-ведомый (master-slave). Использует для передачи данных интерфейсы RS-485, RS-422, RS-232, а также Ethernet сети TCP/IP (протокол Modbus TCP).

Сообщение Modbus RTU состоит из адреса устройства SlaveID, кода функции, специальных данных в зависимости от кода функции и CRC контрольной суммы.

Преимущества:

1. Простота реализации, диагностики и отладки.

Использование стандартных интерфейсов (RS-232/RS-485 и Ethernet) делает Modbus удобным как для разработчиков, так и для пользователей оборудования. При разработке контроллеров и устройств не нужно устанавливать заказные микросхемы для реализации протокола, в отличие от конкурентов-аналогов Profibus и CAN.

2. Высокая скорость внедрения.

Для развертывания первого Modbus-решения нужно всего пару дней, тогда как некоторые протоколы требуют месяцев на подготовку.

3. Нетребовательность к ресурсам.

Разработанный в эпоху 8-битных процессоров, Modbus не предъявляет высоких требований к CPU и RAM. Для начала работы требуется минимум оборудования, и разработка проста в любой операционной системе.

4. Высокая надежность и достоверность при передаче данных.

Поддержка алгоритмов CRC и LRC позволяет определять ошибки в передаче данных с высокой точностью.

5. Универсальность и открытость.

Практически все промышленные АСУ ТП имеют программные драйверы для работы с Modbus-сетями. Благодаря этому Modbus принято считать стандартом де-факто в интеграции мультивендорного оборудования.

Протокол реализован сотнями поставщиков на множество различных датчиков и исполнительных устройств для передачи дискретной и аналоговой информации. Благодаря единому протоколу устройства от различных производителей могут без труда общаться друг с другом.

Недостатки:

1. Отсутствие встроенной аутентификации и шифрования передаваемых данных.

Поэтому при использовании протокола Modbus TCP необходимо настраивать дополнительные VPN-トンнели. Относительно недавно для Modbus TCP было разработано расширение Modbus Security (с поддержкой TLS), но оно пока не получило широкого распространения.

2. Отсутствие начальной инициализации системы.

Назначать сетевые адреса и настраивать параметры каждого конкретного устройства требуется вручную. Некоторые производители разрабатывают шаблоны для своих Modbus-устройств, но это не упрощает их взаимодействие с контроллерами и ПО других производителей.

3. Спецификации для ограниченного набора типов данных.

В протоколе определен метод передачи только для битов и 16-битных регистров. С другими типами данных (строки, числа с плавающей запятой и так далее) различные производители Modbus-решений поступали по собственному усмотрению. По этой причине впоследствии невозможно было внести дополнения в протокол, так как это могло привести к проблемам из-за уже существующего несовпадения форматов.

4. Недостатки Master-Slave-взаимодействия.

Модель «ведущий — ведомый», изначально положенная в основу протокола, предполагает обмен данными только по инициативе клиентского (ведущего) устройства, которое по очереди опрашивает все серверные (ведомые). Из-за этого возникают следующие ограничения:

- У серверных устройств нет возможности отправить оперативную

информацию клиенту (например, сигнал прерывания): нужно ждать своей очереди в опросе.

- Серверные устройства не способны обнаружить потерю связи с клиентом.
- Серверные устройства не могут обмениваться данными друг с другом без участия клиента.

5. Отсутствие поддержки режима Multi-Master для интерфейсов RS-232/RS-485.

Другие протоколы, основанные на этих же интерфейсах, поддерживают работу с несколькими ведущими устройствами (например, CAN и Profibus).

Сфера применения:

Чаще всего Modbus применяется для передачи сигналов от контрольно-измерительных приборов к главному контроллеру или системе сбора данных.

Основные сценарии использования Modbus:

- Клиент-серверные приложения для мониторинга и программирования устройств (в том числе дистанционного) в промышленности, строительстве, инфраструктуре, транспорте, энергетике. Примеры: мониторинг энергопотребления, контроль производственных процессов, надзор за ходом строительства и так далее.
- Передача данных от датчиков и приборов интеллектуальным устройствам в интернете вещей (Internet of Things, IoT).
- Связь диспетчерских компьютеров с удаленными терминалами в SCADA-системах.
- Приложения, где требуется беспроводная связь, например в газовой и нефтяной промышленности.

Несмотря на свой возраст, Modbus активно используется и с современными технологиями — например, он отлично чувствует себя в облаке. Многие провайдеры предлагают возможность создания облачных IoT-платформ — для снижения затрат на разработку IoT-сервисов, обеспечения сбора данных и управления устройствами в real-time-режиме. И поддержка

Modbus — обязательный пункт для подобных решений, так как невозможно построить межмашинное взаимодействие без протокола, реализованного множеством поставщиков на тысячах различных устройств.

2. 1-Wire

1-Wire-net представляет собой информационную сеть, использующую для осуществления цифровой связи одну линию данных и один возвратный (или земляной) провод.

Основой архитектуры 1-Wire-сетей, является топология общей шины, когда каждое из устройств подключено непосредственно к единой магистрали, без каких-либо каскадных соединений или ветвлений. При этом в качестве базовой используется структура сети с одним ведущим или мастером и многочисленными ведомыми.

Стандартная скорость работы 1-Wire-сети составляет 15,4Кбит/сек.

Преимущества:

1. Простое решение адресуемости абонентов,
2. Несложный протокол,
3. Простая структура линии связи,
4. Малое потребление компонентов,
5. Легкое изменение конфигурации сети,
6. Значительная протяженность линий связи,
7. Исключительная дешевизна всей технологии в целом

Недостатки:

1. Необходимость непрерывной временной синхронизации или синхронной работы отдельных устройств в сети,
2. Низкая скорость обмена информацией, и как следствие невозможность высокой динамики при обслуживании быстрых процессов в режиме реального времени,
3. Сложность в реализации мультимастерного режима работы сети

Сфера применения:

1. Идентификация личности.

Каждая микросхема 1-Wire имеет уникальный номер. Это позволяет использовать устройства iButton в качестве простых идентификаторов личности, например, в системах контроля и управления доступом (СКУД). В этом качестве они успешно конкурируют с бесконтактными карточками, использующими технологию RFID.

Имеются устройства iButton с поддержкой криптографии, что позволяет создавать на их основе защищённые хранилища небольших объёмов данных или средства сильной аутентификации. Такие устройства могут конкурировать со смарт-картами в некоторых применениях.

2. Удалённые датчики физических величин

Устройства 1-Wire очень удобны для измерений. Не требуется отдельного питания, возможно подключить по одному проводу целую гирлянду разнообразных датчиков. Система таких датчиков легко контролируется на предмет аварий. Записи о калибровках могут храниться прямо в датчиках.

3. Измерение температуры — одно из самых массовых применений 1-Wire устройств. В сельском хозяйстве применяется для многоточечного контроля температуры в теплицах, ульях, элеваторах, инкубаторах, овощехранилищах. Популярны домашние метеостанции, подключаемые по этому интерфейсу.

4. Маркировка оборудования

Микросхемы 1-Wire популярны для маркировки и хранения параметров дополнительного оборудования к установкам. Например, медицинские и лабораторные приборы, использующие в работе множество различных сменных головок и датчиков, снабжаются микросхемой. При подключении прибор сразу распознаёт сменную головку и корректно устанавливает режим работы. Аналогично может контролироваться наработка узлов с ограниченным ресурсом.

3. I²C

I²C — это последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов. Она синхронная и использует две двунаправленные линии связи: данные (SDA) и такты (SCL). Используется архитектура «ведущий-ведомый» (master-slave), где инициатором обмена всегда выступает ведущий.

Принцип работы I²C основан на двухпроводной схеме, которая включает в себя:

1. SDA (Serial Data Line) — линия данных, по которой передаются данные.
2. SCL (Serial Clock Line) — линия тактовых импульсов, которая синхронизирует передачу данных.

В схеме может быть только один мастер(ведущий), а все остальные подчиненные(ведомые). Мастер инициирует передачу данных, генерируя тактовые импульсы и управляющие сигналы, в то время как подчиненные отвечают на запросы мастера.

Такты на линии SCL генерирует мастер. Данные передаются 8-битными словами, после каждого слова передается бит подтверждения.

В интерфейсе I²C линии SDA и SCL подключены к питанию через подтягивающие резисторы, обычно находящиеся на мастерском устройстве (например, на микроконтроллере). Эти резисторы обеспечивают состояние высокого уровня на линиях данных и тактовых импульсов в отсутствие активного сигнала от мастера или ведомого устройства.

Подтягивающие резисторы также помогают в уменьшении и подавлении искажений и помех на линиях передачи данных, что важно для обеспечения надежной передачи данных в условиях шумного электромагнитного окружения или при длительных соединениях.

Преимущества:

- Простота использования: I²C требует всего два провода для связи, что упрощает проектирование схем.

- Поддержка множества устройств. На одной шине может быть подключено множество устройств, что позволяет создавать сложные системы.
- Гибкость. Возможность работы с различными скоростями передачи данных.
- Стандарт предусматривает «горячее» подключение и отключение устройств в процессе работы системы.

Недостатки:

- Ограниченнaя длина шины. Из-за паразитных емкостей длина шины ограничена несколькими метрами.
- Сложность арбитража. При одновременной работе нескольких мастеров возможны сложности с арбитражем.

Сфера применения:

I²C находит применение в устройствах, предусматривающих простоту разработки и низкую себестоимость изготовления при относительно неплохой скорости работы.

Список возможных применений:

- доступ к модулям памяти NVRAM;
- доступ к низкоскоростным ЦАП/АЦП;
- регулировка контрастности, насыщенности и цветового баланса мониторов;
- регулировка звука в динамиках;
- управление светодиодами, в том числе в мобильных телефонах;
- чтение информации с датчиков мониторинга и диагностики оборудования, например, термостат центрального процессора или скорость вращения вентилятора охлаждения;
- чтение информации с часов реального времени (кварцевых генераторов);
- управление включением/выключением питания системных компонент;

- информационный обмен между микроконтроллерами.

4. CAN

CAN — это стандарт промышленной сети, ориентированный на объединение в единую сеть различных исполнительных устройств и датчиков. Режим передачи — последовательный, широковещательный, пакетный.

Непосредственно стандарт CAN компании Bosch определяет передачу в отрыве от физического уровня — он может быть каким угодно, например, радиоканалом или оптоволокном. Но на практике под CAN-сетью обычно подразумевается сеть топологии «шина» с физическим уровнем в виде дифференциальной пары, определённым в стандарте ISO 11898. Передача ведётся кадрами, которые принимаются всеми узлами сети. Для доступа к шине выпускаются специализированные микросхемы — драйверы CAN-шины.

CAN является синхронной шиной с типом доступа Collision Resolving (CR, разрешение коллизии), который, в отличие от Collision Detect (CD, обнаружение коллизии) сетей (Ethernet), детерминировано (приоритетно) обеспечивает доступ на передачу сообщения, что особо ценно для промышленных сетей управления (fieldbus). Передача ведётся кадрами. Полезная информация в кадре состоит из идентификатора длиной 11 бит (стандартный формат) или 29 бит (расширенный формат, надмножество предыдущего) и поля данных длиной от 0 до 8 байт. Идентификатор говорит о содержимом пакета и служит для определения приоритета при попытке одновременной передачи несколькими сетевыми узлами.

Преимущества:

- Возможность работы в режиме жёсткого реального времени.
- Простота реализации и минимальные затраты на использование.
- Высокая устойчивость к помехам.
- Арбитраж доступа к сети без потерь пропускной способности.
- Надёжный контроль ошибок передачи и приёма.

- Широкий диапазон скоростей работы.
- Большое распространение технологии.

Недостатки:

- Небольшое количество данных, которое можно передать в одном пакете (до 8 байт).
- Большой размер служебных данных в пакете (по отношению к полезным данным).
- Отсутствие единого общепринятого стандарта на протокол высокого уровня.

Сфера применения:

Широко распространён в промышленной автоматизации, технологиях домашней автоматизации («умного дома»), автомобильной промышленности и многих других областях. Стандарт для автомобильной автоматики.

2. Практическая работа №6

После подключения к консоли WirenBoard через протокол SSH было произведено подключение к топикам.

```

user@wirenboard-AZFKTK7T: ~
login as: user
user@192.168.1.24's password:

Welcome to Wiren Board 8.5.1 (s/n AZFKTK7T), release wb-2507 (as stable)
Linux wirenboard-AZFKTK7T 6.8.0-wb140 #1 SMP Fri Jul 18 08:08:05 UTC 2025 aarch64
GNU/Linux

System load: 1.83 1.36 1.26 Up time: 1:35
Memory usage: 13% of 1.94G Usage of /: 43% of 2.0G /mnt/data: 1
1% of 12G

17 package updates are available; type 'apt update && apt upgrade' to update them.

Last login: Thu Oct 9 11:24:58 2025 from 192.168.1.90
user@wirenboard-AZFKTK7T:~$ 

```

Рисунок 6.1 – Приветственный баннер контроллера

```
user@wirenboard-AZFKTK7T:~$ mosquitto_sub -t '/devices/wb-mlw2_30/controls/External Sensor 1' -p 1883  
27  
26.9375  
27
```

Рисунок 6.2 - Подписка на индикатор устройства WB-MSW v.3

```
user@wirenboard-AZFKTK7T:~$ mosquitto_pub -t '/devices/wb-mr6c_46/controls/K2/on' -p 1883 -m '1'  
user@wirenboard-AZFKTK7T:~$
```

Рисунок 6.3 - Изменение подсветки кнопки 28

```
connection clusterfly  
address srv2.clusterfly.ru:9991  
remote_username user_2a32bcf7  
remote_password 2Y1x3gLTcfQMd  
try_private false  
notifications true  
notification_topic /client/wb_7/bridge_status  
start_type automatic  
topic /# both 0 "" user_2a32bcf7  
bridge_insecure true  
cleansession false
```

Рисунок 6.4 - Файл конфигурации моста

```
user@wirenboard-AZFKTK7T:~$ mosquitto_sub -v -t "/client/wb_7/bridge_status" -h 192.168.1.24  
/client/wb_7/bridge_status 1  
/client/wb_7/bridge_status 0  
/client/wb_7/bridge_status 1  
/client/wb_7/bridge_status 1
```

Рисунок 6.5 - Отображение подключения к MQTT-мосту

3. Практическая работа №7

Получаемая структура данных:

- case-id - int (id используемого чемоданчика);
- humidity – float (влажность);
- motion – float (показание датчика движения устройства);
- voltage – float (потребляемое напряжение на одном из устройств стенда);
- time – DATETIME (текущее время в виде hh:mm:ss).

Для получения данных был написан data_collector.py.

Листинг 7.1 – Часть 1 логики data_collector.py

```
import paho.mqtt.client as mqtt
import json
import xml.etree.ElementTree as ET
from xml.dom import minidom
from datetime import datetime

# Параметры подключения к MQTT-брокеру
HOST = "192.168.1.24"
PORT = 1883
KEEPALIVE = 60

# MQTT топики для подключения
SUB_TOPICS = {
    '/devices/wb-msw-v3_64/controls/Humidity': 'humidity',
    '/devices/wb-msw-v3_64/controls/Current Motion': 'motion',
    '/devices/battery/controls/Voltage': 'voltage'
}

# Хранение собранных данных в оперативной памяти
RECORD_LIST = []
RECORD_DICT = {'case-id': HOST[-2::]}
for value in SUB_TOPICS.values():
    RECORD_DICT[value] = 0
RECORD_DICT['time'] = ''

LAST_SAVE_TIME = datetime.now()

# Вызывается при подписке на топик
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    for topic in SUB_TOPICS.keys():
        client.subscribe(topic)

# Вызывается при получении сообщения от топика
def on_message(client, userdata, msg):
    global LAST_SAVE_TIME
    payload = msg.payload.decode()
    topic = msg.topic

    param_name = SUB_TOPICS[topic]
    RECORD_DICT[param_name] = payload
    RECORD_DICT['time'] = str(datetime.now())
    if (datetime.now() - LAST_SAVE_TIME).total_seconds() < 5:
        return
    LAST_SAVE_TIME = datetime.now()

    RECORD_LIST.append(RECORD_DICT.copy())

    print(topic + " " + payload)

    # Запись данных в json файл
    with open('data.json', 'w') as file:
        json_string = json.dumps(RECORD_LIST)
        file.write(json_string)
```

Листинг 7.2 – Часть 2 логики data_collector.py

```
# Запись данных в xml файл
    with open('data.xml', 'w') as file:
        root = ET.Element("root")
        for record in RECORD_LIST:
            record_xml = ET.SubElement(root, "record")
            case_id = record['case-id']
            case_id.text = str(case_id)
            humidity = ET.SubElement(record_xml, "humidity")
            humidity.text = record['humidity']
            voltage = ET.SubElement(record_xml, "voltage")
            voltage.text = record['voltage']
            motion = ET.SubElement(record_xml, "motion")
            motion.text = record['motion']
            time = ET.SubElement(record_xml, "time")
            time.text = record['time']

            xml_string = ET.tostring(root, encoding="utf-8")
            parsed = minidom.parseString(xml_string)
            pretty_string = parsed.toprettyxml(indent=" ")

            file.write(pretty_string)

# Создание mqtt клиента и запуск сбора данных
def main():
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(HOST, PORT, KEEPALIVE)

    client.loop_forever()

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

```
[{"case-id": "24", "humidity": "40.2", "motion": "141", "voltage": "4.097", "time": "2025-10-23 14:35:29.891461"}, {"case-id": "24", "humidity": "40.2", "motion": "51", "voltage": "4.097", "time": "2025-10-23 14:35:35.173218"}, {"case-id": "24", "humidity": "40.2", "motion": "26", "voltage": "4.097", "time": "2025-10-23 14:35:40.219006"}, {"case-id": "24", "humidity": "40.2", "motion": "33", "voltage": "4.097", "time": "2025-10-23 14:35:45.220074"}]
```

Рисунок 7.1 – Собранная информация для data.json

```
data_collector.py      data.json      </> data.xml ×
1  <?xml version="1.0" ?>
2  <root>
3  <record>
4  <case-id>24</case-id>
5  <humidity>40.2</humidity>
6  <voltage>4.097</voltage>
7  <motion>141</motion>
8  <time>2025-10-23 14:35:29.891461</time>
9  </record>
10 <record>
11 <case-id>24</case-id>
12 <humidity>40.2</humidity>
13 <voltage>4.097</voltage>
14 <motion>51</motion>
15 <time>2025-10-23 14:35:35.173218</time>
16 </record>
17 <record>
18 <case-id>24</case-id>
19 <humidity>40.2</humidity>
20 <voltage>4.097</voltage>
21 <motion>26</motion>
22 <time>2025-10-23 14:35:40.219006</time>
23 </record>
24 <record>
25 <case-id>24</case-id>
26 <humidity>40.2</humidity>
27 <voltage>4.097</voltage>
28 <motion>33</motion>
29 <time>2025-10-23 14:35:45.220074</time>
30 </record>
31 </root>
32
```

Рисунок 7.2 – Собранная информация для data.xml

Для вывода полученных данных на экран был написан data-parser.py.

Листинг 7.3 – Часть 1 логики data-parser.py

```
import json
import sys
import xml.etree.ElementTree as ET

# Парсинг json файла и вывод
def parse_json_file(filename):
    with open(filename, 'r') as file:
        data = json.load(file)

    print("=" * 80)
    print(f"Данные из JSON файла ({len(data)} записей):")
    print("=" * 80)

    for i, record in enumerate(data, 1):
        print(f"Запись #{i}:")
        print(f"  ID чемодана: {record.get('case-id', 'N/A')} ")
        print(f"  Влажность: {record.get('humidity', 'N/A')} ")
        print(f"  Движение: {record.get('motion', 'N/A')} ")
        print(f"  Напряжение: {record.get('voltage', 'N/A')} ")
        print(f"  Время: {record.get('time', 'N/A')} ")
        print("-" * 40)

# Парсинг xml файла и вывод
def parse_xml_file(filename):
    tree = ET.parse(filename)
    root = tree.getroot()

    records = root.findall('record')

    print("=" * 80)
    print(f"Данные из XML файла ({len(records)} записей):")
    print("=" * 80)

    for i, record in enumerate(records, 1):
        print(f"Запись #{i}:")

        case_id = record.find('case-id')
        humidity = record.find('humidity')
        motion = record.find('motion')
        voltage = record.find('voltage')
        time_elem = record.find('time')

        print(f"  ID чемодана: {case_id.text if case_id is not None else 'N/A'} ")
        print(f"  Влажность: {humidity.text if humidity is not None else 'N/A'} ")
        print(f"  Движение: {motion.text if motion is not None else 'N/A'} ")
        print(f"  Напряжение: {voltage.text if voltage is not None else 'N/A'} ")
        print(f"  Время: {time_elem.text if time_elem is not None else 'N/A'} ")
        print("-" * 40)
```

Листинг 7.4 – Часть 2 логики data-parser.py

```
# Запуск парсера и обработка ошибок

def main():

    try:

        argv = sys.argv

        filename = argv[1]

        parse_json_file(filename)

    except FileNotFoundError as e:

        print(f"Ошибка: Файл не найден - {e}")

    except json.JSONDecodeError as e:

        try:

            parse_xml_file(filename)

        except ET.ParseError as e:

            print(f"Ошибка: Неверный формат файла")

        except Exception as e:

            print(f"Неожиданная ошибка: {e}")

    except Exception as e:

        print(f"Неожиданная ошибка: {e}")

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

```
"C:\Program Files\Python311\python.exe" C:\Users\StudentMOSIT\7\data_parser.py data.json
=====
Данные из JSON файла (4 записей):
=====
Запись #1:
    ID чемодана: 24
    Влажность: 40.2
    Движение: 141
    Напряжение: 4.097
    Время: 2025-10-23 14:35:29.891461
-----
Запись #2:
    ID чемодана: 24
    Влажность: 40.2
    Движение: 51
    Напряжение: 4.097
    Время: 2025-10-23 14:35:35.173218
-----
Запись #3:
    ID чемодана: 24
    Влажность: 40.2
    Движение: 26
    Напряжение: 4.097
    Время: 2025-10-23 14:35:40.219006
-----
Запись #4:
    ID чемодана: 24
    Влажность: 40.2
    Движение: 33
    Напряжение: 4.097
    Время: 2025-10-23 14:35:45.220074
-----
Process finished with exit code 0
```

Рисунок 7.1 – Итог парсинга data.json

```
"C:\Program Files\Python311\python.exe" C:\Users\StudentMOSIT\7\data_parser.py data.xml
=====
Данные из XML файла (4 записей):
=====
Запись #1:
    ID чемодана: 24
    Влажность: 40.2
    Движение: 141
    Напряжение: 4.097
    Время: 2025-10-23 14:35:29.891461
-----
Запись #2:
    ID чемодана: 24
    Влажность: 40.2
    Движение: 51
    Напряжение: 4.097
    Время: 2025-10-23 14:35:35.173218
-----
Запись #3:
    ID чемодана: 24
    Влажность: 40.2
    Движение: 26
    Напряжение: 4.097
    Время: 2025-10-23 14:35:40.219006
-----
Запись #4:
    ID чемодана: 24
    Влажность: 40.2
    Движение: 33
    Напряжение: 4.097
    Время: 2025-10-23 14:35:45.220074
-----
Process finished with exit code 0
```

Рисунок 7.2 – Итог парсинга data.xml

Описание используемых библиотек.

1. **json** — работа с JSON-форматом:

- json.dumps() — преобразует объект Python в JSON-строку.
- json.load() — загружает JSON-данные из файла в объект Python.

2. **sys** — взаимодействие с системой:

- sys.argv — список аргументов командной строки.

3. **xml.etree.ElementTree (as ET)** — работа с XML:

- ET.Element() — создаёт XML-элемент.
- ET.SubElement() — создаёт дочерний элемент.
- ET.tostring() — преобразует XML-дерево в строку.
- ET.parse() — загружает XML из файла.

4. **datetime** — работа с датой и временем:

- datetime.now() — возвращает текущие дату и время.

5. **paho.mqtt.client (as mqtt)** — MQTT-клиент для обмена сообщениями:

- mqtt.Client() — создаёт MQTT-клиент.
- client.on_connect — callback при подключении к брокеру.
- client.on_message — callback при получении сообщения.
- client.subscribe() — подписка на топик.
- client.connect() — подключение к брокеру.
- client.loop_forever() — запуск сетевого цикла.

Кастомные функции, используемые в data-collector.py:

- on_connect() — подписывается на MQTT-топики при подключении;
- on_message() — обрабатывает входящие сообщения:
 - 1) Обновляет RECORD_DICT;
 - 2) Сохраняет данные в JSON и XML каждые 5 секунд.

Кастомные функции, используемые в data-parser.py:

- parse_json_file() — читает и выводит данные из JSON-файла;

- `parse_xml_file()` — парсит и выводит данные из XML-файла.

4. Практическая работа №8

На основе предыдущей работы было осуществлено подключение к топикам варианта и были собраны данные на протяжении 10 минут.

Листинг 8.1 – Часть 1 коллектор

```
import paho.mqtt.client as mqtt
import csv
from datetime import datetime

# Параметры подключения к MQTT-брюкеру
HOST = "192.168.1.24"
PORT = 1883
KEEPALIVE = 60

# MQTT топики для подключения
SUB_TOPICS = {
    '/devices/wb-msw-v3_64/controls/Humidity': 'humidity',
    '/devices/wb-msw-v3_64/controls/Current Motion': 'motion',
    '/devices/battery/controls/Voltage': 'voltage'
}

# Хранение собранных данных в оперативной памяти
RECORD_LIST = []
RECORD_DICT = {}
for value in SUB_TOPICS.values():
    RECORD_DICT[value] = 0
RECORD_DICT['time'] = ''

LAST_SAVE_TIME = datetime.now()

# Вызывается при подписке на топик
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    for topic in SUB_TOPICS.keys():
        client.subscribe(topic)

# Вызывается при получении сообщения от топика
def on_message(client, userdata, msg):
    global LAST_SAVE_TIME
    payload = msg.payload.decode()
    topic = msg.topic

    param_name = SUB_TOPICS[topic]
    RECORD_DICT[param_name] = payload
    RECORD_DICT['time'] = str(datetime.now())
    if (datetime.now() - LAST_SAVE_TIME).total_seconds() < 5:
        return
    LAST_SAVE_TIME = datetime.now()

    RECORD_LIST.append(RECORD_DICT.copy())
    print(topic + " " + payload)
```

Листинг 8.2 – Часть 2 коллектор

```
# Запись данных в csv файл
with open('data.csv', 'a', newline='', encoding='utf-8') as csvfile:
    fieldnames = ['humidity', 'motion', 'voltage', 'time']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writerow(RECORD_DICT)

# Создание mqtt клиента и запуск сбора данных
def main():
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(HOST, PORT, KEEPALIVE)

    with open('data.csv', 'w', newline='', encoding='utf-8') as csvfile:
        fieldnames = ['humidity', 'motion', 'voltage', 'time']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
```

Листинг 8.3 – Часть 1 парсер

```
import csv
from datetime import datetime

from matplotlib import pyplot as plt
import sys

# Парсинг csv файла
def parse_csv_file(filename):
    with open(filename, 'r', encoding='utf-8') as csvfile:
        reader = csv.DictReader(csvfile)
        records = list(reader)
        plot_linear(records, 'motion')
        plot_histogram(records, 'voltage')
        plot_pie(records, 'humidity')

def time_to_sec(time):
    return time.hour * 3600 + time.minute * 60 + time.second

def plot_linear(records, parameter):
    start_time = time_to_sec(datetime.strptime(records[0]['time'],
                                                "%Y-%m-%d %H:%M:%S.%f").time())
    times = [time_to_sec(datetime.strptime(record['time'],
                                            "%Y-%m-%d %H:%M:%S.%f").time()) - start_time for record in records]
    values = [float(record[parameter]) for record in records]

    plt.figure(figsize=(10, 5))
    plt.plot(times, values)
    plt.title(f'График {parameter} от времени')
    plt.xlabel('Время')
    plt.ylabel(parameter)
    plt.grid(True)
    plt.show()

def plot_histogram(records, parameter):
    values = [record[parameter] for record in records]

    plt.figure(figsize=(10, 5))
    plt.hist(values, bins=20, edgecolor='black', alpha=0.7)
    plt.title(f'Гистограмма {parameter}'')
```

Листинг 8.4 - Часть 2 парсер

```
plt.xlabel(parameter)
plt.ylabel('Частота')
plt.grid(True, alpha=0.3)
plt.show()

def plot_pie(records, parameter):
    values = [float(record[parameter]) for record in records]

    # Создаем диапазоны значений для секторов
    min_val = min(values)
    max_val = max(values)
    range_size = (max_val - min_val) / 5

    ranges = []
    counts = []

    for i in range(5):
        lower = min_val + i * range_size
        upper = min_val + (i + 1) * range_size
        count = sum(1 for v in values if lower <= v < upper)
        if count > 0:
            ranges.append(f'{lower:.2f}-{upper:.2f}')
            counts.append(count)

    # Обработка максимального значения
    count_max = sum(1 for v in values if v == max_val)
    if count_max > 0 and f'{min_val + 4 * range_size:.2f}-{max_val:.2f}' not in ranges:
        ranges.append(f'{min_val + 4 * range_size:.2f}-{max_val:.2f}')
        counts.append(count_max)

    plt.figure(figsize=(8, 8))
    plt.pie(counts, labels=ranges, autopct='%.1f%%',
    startangle=90)
    plt.title(f'Круговая диаграмма распределения {parameter}')
    plt.show()

# Запуск парсера и обработка ошибок
def main():
    try:
        argv = sys.argv
        filename = argv[1]
        parse_csv_file(filename)

    except FileNotFoundError as e:
        print(f"Ошибка: Файл не найден - {e}")
    except csv.Error as e:
        print(f"Ошибка: Неверный формат файла")
    except Exception as e:
        print(f"Неожиданная ошибка: {e}")

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

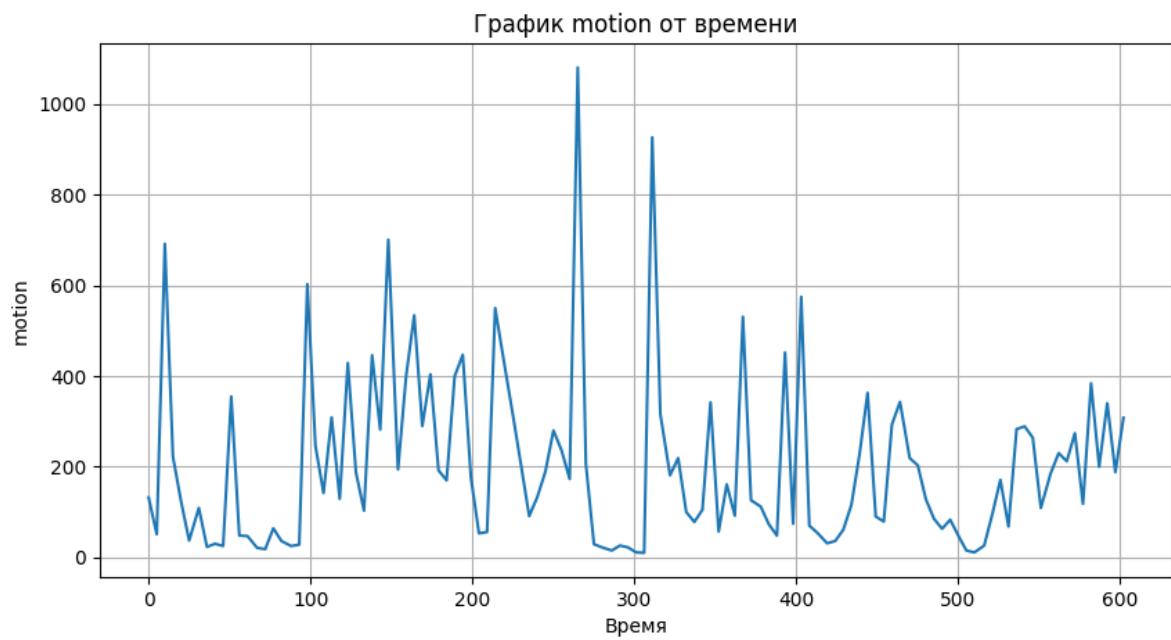


Рисунок 8.1 – Линейный график humidity от времени

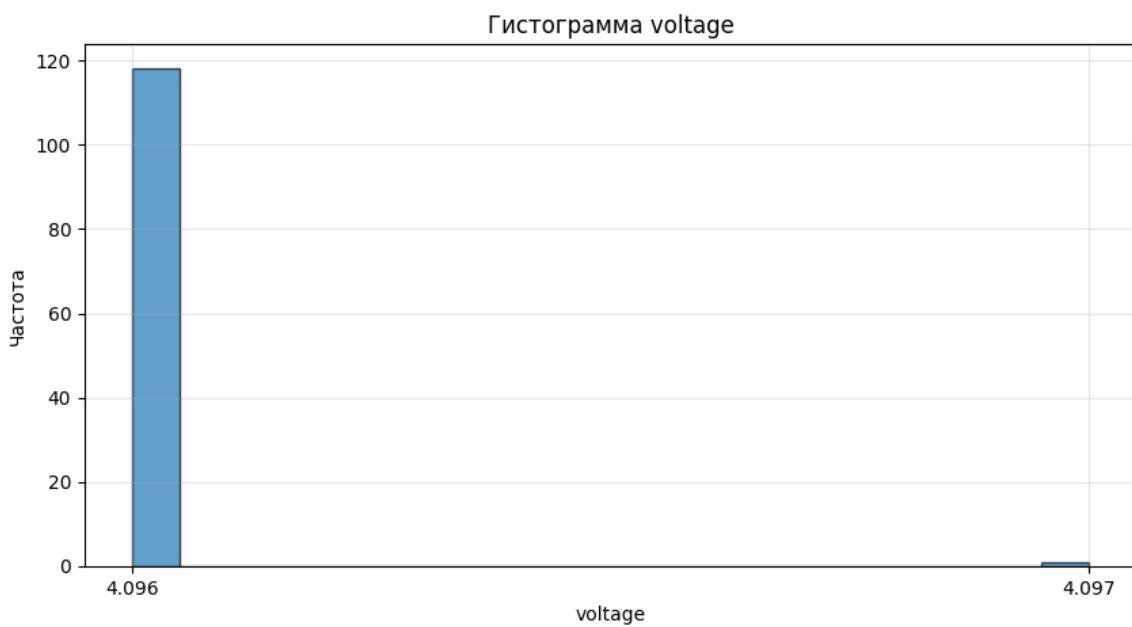


Рисунок 8.2 – Гистограмма график motion от времени

Круговая диаграмма распределения humidity

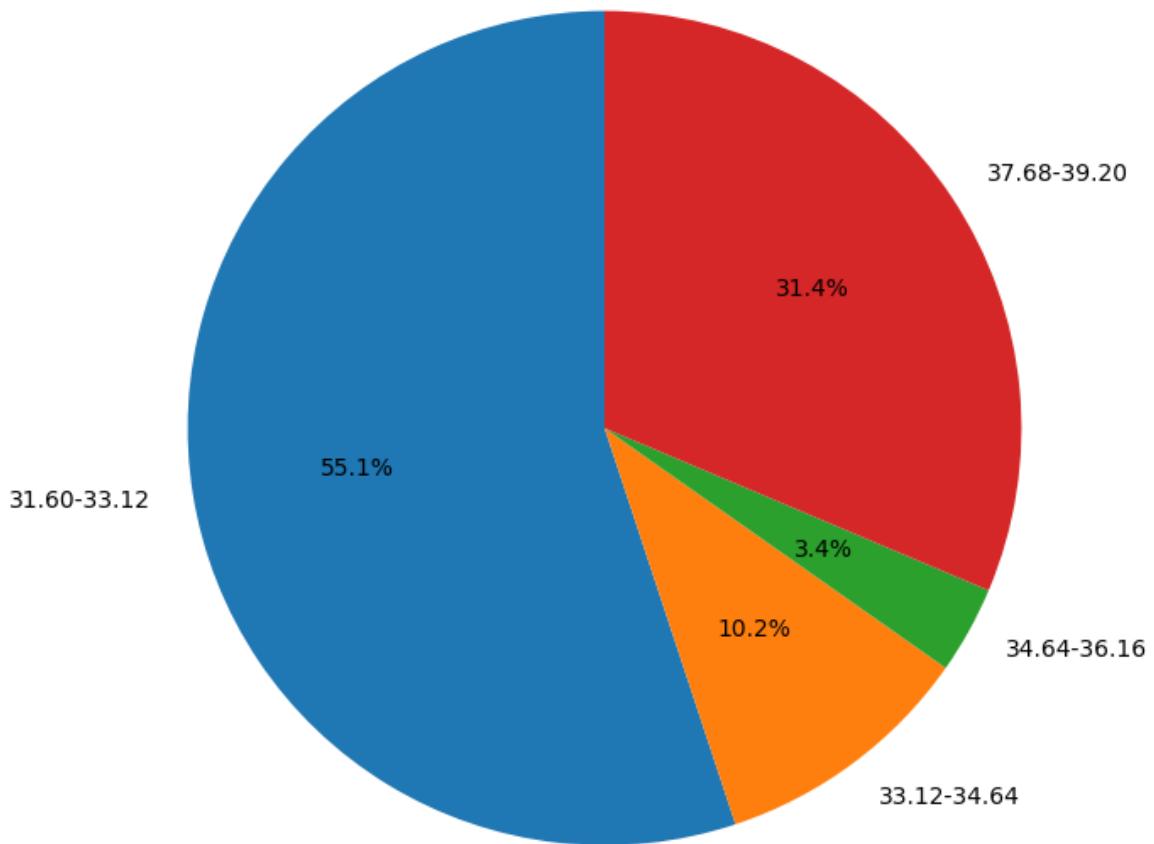


Рисунок 8.3 – Круговая диаграмма voltage по значениям

5. Дополнительные задания

5.1 Практическое занятие №7

Структура данных:

- greenhouse_id: str
- timestamp: str
- temperature: float
- humidity: float
- light: int
- co2: int

```

C:\Users\Noip\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\Noip\Desktop\MySelf\pRAN_d_tics\TOMB - задачи\practices\dops\7-8\emulator.py"
Запуск эмуляции датчиков теплицы GH_001
[13:41:48] Темп: 32.5°C | Вл-ть: 65.5% | Свет: 1784lux | CO2: 585ppm
[13:41:54] Темп: 31.9°C | Вл-ть: 64.9% | Свет: 1768lux | CO2: 516ppm
[13:42:00] Темп: 31.2°C | Вл-ть: 65.4% | Свет: 1751lux | CO2: 514ppm
[13:42:06] Темп: 31.9°C | Вл-ть: 65.8% | Свет: 1738lux | CO2: 526ppm
[13:42:12] Темп: 32.5°C | Вл-ть: 66.1% | Свет: 1722lux | CO2: 525ppm

```

Рисунок 5.1.1 - Пример выходных данных

```

1  > import ...
2
3
4
5  class GreenhouseSensorEmulator: 1 usage
6      """Эмулятор датчиков теплицы"""
7
8  def __init__(self, greenhouse_id="GH_001"):
9      self.greenhouse_id = greenhouse_id
10     self.sensor_readings = {
11         'temperature': {'min': 18.0, 'max': 35.0, 'current': 25.0, 'trend': 0.1},
12         'humidity': {'min': 40.0, 'max': 85.0, 'current': 65.0, 'trend': 0.2},
13         'light_level': {'min': 0, 'max': 2000, 'current': 1200, 'trend': -10},
14         'co2_level': {'min': 300, 'max': 1200, 'current': 500, 'trend': 5}
15     }
16     self.mqtt_topics = [
17         '/devices/wb-msw-v3_64/controls/Temperature': 'temperature',
18         '/devices/wb-msw-v3_64/controls/Humidity': 'humidity',
19         '/devices/battery/controls/Voltage': 'voltage'
20     ]
21
22     def update_sensor_readings(self): 1 usage
23         for sensor, params in self.sensor_readings.items():
24             change = params['trend'] + random.uniform(-1, 1)
25             params['current'] += change
26
27             if params['current'] < params['min']:
28                 params['current'] = params['min']
29                 params['trend'] = abs(params['trend'])
30             elif params['current'] > params['max']:
31                 params['current'] = params['max']
32                 params['trend'] = -abs(params['trend'])
33
34             if random.random() < 0.1:
35                 params['trend'] = random.uniform(-1, 1)
36
37     def generate_sensor_data(self): 1 usage
38         self.update_sensor_readings()
39
40         hour = datetime.now().hour
41         if 6 <= hour <= 18:

```

Рисунок 5.1.2 - emulator.py часть 1

```

5  class GreenhouseSensorEmulator: 1 usage
6      def generate_sensor_data(self): 1 usage
7
8          temp_multiplier = 1.0 + (min(hour - 6, 18 - hour) * 0.05)
9          light_multiplier = 1.0 + (min(hour - 6, 18 - hour) * 0.1)
10         else:
11             temp_multiplier = 0.8
12             light_multiplier = 0.1
13
14         sensor_data = {
15             "greenhouse_id": self.greenhouse_id,
16             "timestamp": datetime.now().isoformat(),
17             "temperature": round(self.sensor_readings['temperature']['current'] * temp_multiplier, 1),
18             "humidity": round(self.sensor_readings['humidity']['current'], 1),
19             "light_level": int(self.sensor_readings['light_level']['current'] * light_multiplier),
20             "co2_level": int(self.sensor_readings['co2_level']['current']),
21         }
22
23
24         return sensor_data
25     def start_emulation(self): 1 usage
26         print("Запуск эмуляции датчиков теплицы {self.greenhouse_id}")
27
28         try:
29             timer = datetime.now()
30             while True:
31                 if (datetime.now() - timer).seconds >= 5:
32                     continue
33                     timer = datetime.now()
34                     data = self.generate_sensor_data()
35
36                     print(f"[{data['timestamp'][11:19]}] ", end="")
37                     print(f"Temp: {data['temperature']:5.1f}°C | ", end="")
38                     print(f"Hum: {data['humidity']:5.1f}% | ", end="")
39                     print(f"Light: {data['light_level']:4d}lux | ", end="")
40                     print(f"CO2: {data['co2_level']:4d}ppm")
41
42             except KeyboardInterrupt:
43                 print("\n\nВынуждено остановлено пользователем")
44
45         if __name__ == "__main__":
46             emulator = GreenhouseSensorEmulator("GH_001")
47             emulator.start_emulation()

```

Рисунок 5.1.3 - emulator.py часть 2

5.2 Практическая работа №8

Команды:

- Была проведена установка брокера mosquito через sudo apt install mosquito
- Для запуска использовались команды (на скрине)
- Сначала эмулятор запускается, подключается к брокеру, а потом сборщик информации также подписывается к топикам брокера.

Сборщик данных собирает информацию в базе данных sqlite

```
krezon@Krezon:/mnt/c/Users/krezo/Education/Sem5/IoT/Projects/8$ sudo systemctl start mosquitto
krezon@Krezon:/mnt/c/Users/krezo/Education/Sem5/IoT/Projects/8$ sudo systemctl enable mosquitto
Synchronizing state of mosquitto.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable mosquitto
krezon@Krezon:/mnt/c/Users/krezo/Education/Sem5/IoT/Projects/8$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
    Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
    Active: active (running) since Thu 2025-10-30 01:32:24 MSK; 21min ago
      Docs: man:mosquitto.conf(5)
            man:mosquitto(8)
   Main PID: 806 (mosquitto)
     Tasks: 1 (limit: 18975)
    Memory: 1.0M (peak: 2.0M)
       CPU: 695ms
      CGroup: /system.slice/mosquitto.service
              └─806 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
```

Рисунок 5.2.1 – Используемые скрипты

id	timestamp	sensor	topic	value
1	2025-10-30T14:14:28.306452	temperature	greenhouse/GH_001/sensors/...	32.6
2	2025-10-30T14:14:28.322020	humidity	greenhouse/GH_001/sensors/...	71.1
3	2025-10-30T14:14:28.331605	light	greenhouse/GH_001/sensors/...	1628
4	2025-10-30T14:14:28.341009	co2	greenhouse/GH_001/sensors/...	548
5	2025-10-30T14:14:33.312467	temperature	greenhouse/GH_001/sensors/...	32.5
6	2025-10-30T14:14:33.329363	humidity	greenhouse/GH_001/sensors/...	72.9
7	2025-10-30T14:14:33.338355	light	greenhouse/GH_001/sensors/...	1628
8	2025-10-30T14:14:33.348040	co2	greenhouse/GH_001/sensors/...	556
9	2025-10-30T14:14:38.319474	temperature	greenhouse/GH_001/sensors/...	34.8
10	2025-10-30T14:14:38.333686	humidity	greenhouse/GH_001/sensors/...	74.3
11	2025-10-30T14:14:38.344059	light	greenhouse/GH_001/sensors/...	1625
12	2025-10-30T14:14:38.354978	co2	greenhouse/GH_001/sensors/...	558
13	2025-10-30T14:14:43.327879	temperature	greenhouse/GH_001/sensors/...	38.1
14	2025-10-30T14:14:43.347042	humidity	greenhouse/GH_001/sensors/...	78.3
15	2025-10-30T14:14:43.357102	light	greenhouse/GH_001/sensors/...	1620
16	2025-10-30T14:14:43.366206	co2	greenhouse/GH_001/sensors/...	563

Рисунок 5.2.2 – Просмотр итоговой бд в sqlite

```

1 import dataclasses
2
3 import random
4 import time
5 import paho.mqtt.client as mqtt
6 from datetime import datetime
7
8 @dataclasses.dataclass 3 usages
9 class SensorData:
10     greenhouse_id: str
11     timestamp: str
12     temperature: float
13     humidity: float
14     light: int
15     co2: int
16
17     def as_dict(self):
18         return dataclasses.asdict(self)
19
20     def __str__(self):
21         return f'{self.timestamp} {self.temperature:5.1f}c | '
22         f'{self.humidity:5.1f}% | '
23         f'{self.light:4d}lux | '
24         f'{self.co2:4d}ppm'
25
26 class SensorEmulator: 1 usage
27     """3МИЛЯТОД ДАТЧИКОВ ТЕПЛИЦЫ"""
28     def __init__(self, greenhouse_id='GH_001', broker_host='localhost', broker_port=1883):
29         self.greenhouse_id = greenhouse_id
30         self.broker_host = broker_host
31         self.broker_port = broker_port
32
33         self.mqtt_client = mqtt.Client()
34         self.mqtt_client.on_connect = self.on_connect
35         self.mqtt_client.on_publish = self.on_publish
36         self.sensor_readings = {
37             'temperature': {'min': 18.0, 'max': 35.0, 'current': 25.0, 'trend': 0.1},
38             'humidity': {'min': 40.0, 'max': 85.0, 'current': 65.0, 'trend': 0.2},
39             'light': {'min': 0, 'max': 2000, 'current': 1200, 'trend': -10},
40             'co2': {'min': 300, 'max': 1200, 'current': 500, 'trend': 5}
41         }
42
43     self.mqtt_topics = {
44         'temperature': f'greenhouse/{greenhouse_id}/sensors/temperature',
45         'humidity': f'greenhouse/{greenhouse_id}/sensors/humidity',
46         'light': f'greenhouse/{greenhouse_id}/sensors/light',
47         'co2': f'greenhouse/{greenhouse_id}/sensors/co2'
48     }
49
50     def on_connect(self, client, userdata, flags, rc): 3 usages
51         if rc == 0:
52             print(f'Успешное подключение к MQTT брокеру {self.broker_host}:{self.broker_port}')
53         else:
54             print(f'Ошибка подключения к MQTT брокеру. Код: {rc}')
55
56     def on_publish(self, client, userdata, mid): 2 usages
57         print(f'Сообщение опубликовано (mid: {mid})')
58
59     def update_sensor_readings(self): 1 usage
60         for sensor, params in self.sensor_readings.items():
61             change = params['trend'] + random.uniform(-1, 1)
62             params['current'] += change
63             if params['current'] < params['min']:
64                 params['current'] = params['min']
65             elif params['current'] > params['max']:
66                 params['current'] = params['max']
67             params['trend'] = abs(params['trend'])
68             if random.random() < 0.1:
69                 params['trend'] = random.uniform(-1, 1)
70
71     def generate_sensor_data(self) -> SensorData: 1 usage
72         self.update_sensor_readings()
73         hour = datetime.now().hour
74         if 6 <= hour <= 18:
75             temp_multiplier = 1.0 + (min(hour - 6, 18 - hour) * 0.05)
76             light_multiplier = 1.0 + (min(hour - 6, 18 - hour) * 0.1)
77         else:
78             temp_multiplier = 0.8
79             light_multiplier = 0.1
80
81         return SensorData(
82             timestamp=datetime.now().isoformat(),
83             greenhouse_id=self.greenhouse_id,
84             temperature=params['current'] * temp_multiplier,
85             humidity=params['current'],
86             light=params['current'] * light_multiplier,
87             co2=params['current']
88         )

```

Рисунок 5.2.3 - emulator(1).py часть 1

```

26 class SensorEmulator: 1 usage
27     def __init__(self, greenhouse_id='GH_001', broker_host='localhost', broker_port=1883):
28         ...
29
30         self.mqtt_topics = {
31             'temperature': f'greenhouse/{greenhouse_id}/sensors/temperature',
32             'humidity': f'greenhouse/{greenhouse_id}/sensors/humidity',
33             'light': f'greenhouse/{greenhouse_id}/sensors/light',
34             'co2': f'greenhouse/{greenhouse_id}/sensors/co2'
35         }
36
37     def on_connect(self, client, userdata, flags, rc): 3 usages
38         if rc == 0:
39             print(f'Успешное подключение к MQTT брокеру {self.broker_host}:{self.broker_port}')
40         else:
41             print(f'Ошибка подключения к MQTT брокеру. Код: {rc}')
42
43     def on_publish(self, client, userdata, mid): 2 usages
44         print(f'Сообщение опубликовано (mid: {mid})')
45
46     def update_sensor_readings(self): 1 usage
47         for sensor, params in self.sensor_readings.items():
48             change = params['trend'] + random.uniform(-1, 1)
49             params['current'] += change
50             if params['current'] < params['min']:
51                 params['current'] = params['min']
52             elif params['current'] > params['max']:
53                 params['current'] = params['max']
54             params['trend'] = abs(params['trend'])
55             if random.random() < 0.1:
56                 params['trend'] = random.uniform(-1, 1)
57
58     def generate_sensor_data(self) -> SensorData: 1 usage
59         self.update_sensor_readings()
60         hour = datetime.now().hour
61         if 6 <= hour <= 18:
62             temp_multiplier = 1.0 + (min(hour - 6, 18 - hour) * 0.05)
63             light_multiplier = 1.0 + (min(hour - 6, 18 - hour) * 0.1)
64         else:
65             temp_multiplier = 0.8
66             light_multiplier = 0.1
67
68         return SensorData(
69             timestamp=datetime.now().isoformat(),
70             greenhouse_id=self.greenhouse_id,
71             temperature=params['current'] * temp_multiplier,
72             humidity=params['current'],
73             light=params['current'] * light_multiplier,
74             co2=params['current']
75         )

```

Рисунок 5.2.4 - emulator(1).py часть 2

```

26     class SensorEmulator: 1 usage
27         def generate_sensor_data(self) -> SensorData: 1 usage
28             return SensorData(
29                 greenhouse_id=self.greenhouse_id,
30                 timestamp=datetime.now().isoformat(),
31                 temperature=round(self.sensor_readings['temperature']['current'] * temp_multiplier, 1),
32                 humidity=round(self.sensor_readings['humidity']['current'], 1),
33                 light=int(self.sensor_readings['light']['current'] * light_multiplier),
34                 co2=int(self.sensor_readings['co2']['current'])
35             )
36         def publish(self, sensor_data: SensorData): 2 usages (1 dynamic)
37             msgs = [
38                 (self.mqtt_topics["temperature"], sensor_data.temperature),
39                 (self.mqtt_topics["humidity"], sensor_data.humidity),
40                 (self.mqtt_topics["light"], sensor_data.light),
41                 (self.mqtt_topics["co2"], sensor_data.co2)
42             ]
43             for topic, value in msgs:
44                 self.mqtt_client.publish(topic, str(value))
45             # print(sensor_data)
46
47         def start_emulation(self, interval): 1 usage
48             print(f"Запуск эмуляции датчиков теплицы {self.greenhouse_id}")
49             self.mqtt_client.connect(self.broker_host, self.broker_port, 60)
50             try:
51                 while True:
52                     data = self.generate_sensor_data()
53
54                     self.publish(data)
55                     time.sleep(interval)
56             except KeyboardInterrupt:
57                 print("\n\nЭмуляция остановлена пользователем")
58
59     if __name__ == "__main__":
60         emulator = SensorEmulator(
61             greenhouse_id="GH_001",
62             broker_host="localhost",
63             broker_port=1883
64         )
65         emulator.start_emulation(interval=1)

```

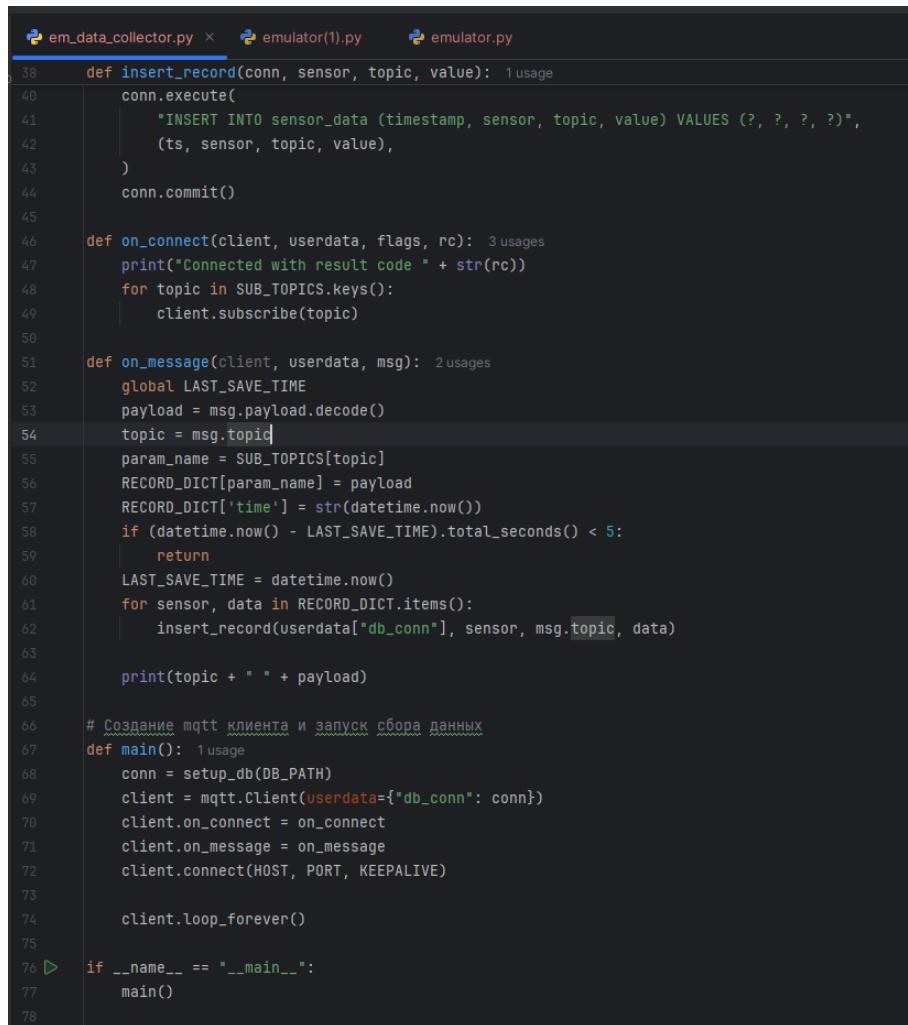
Рисунок 5.2.5 - emulator(1).py часть 3

```

1 import paho.mqtt.client as mqtt
2 import sqlite3
3 from datetime import datetime
4
5 HOST = "localhost"
6 PORT = 1883
7 KEEPALIVE = 60
8 greenhouse_id = "GH_001"
9
10 SUB_TOPICS = {
11     f"greenhouse/{greenhouse_id}/sensors/temperature": "temperature",
12     f"greenhouse/{greenhouse_id}/sensors/humidity": "humidity",
13     f"greenhouse/{greenhouse_id}/sensors/light": "light",
14     f"greenhouse/{greenhouse_id}/sensors/co2": "co2"
15 }
16
17 RECORD_DICT = {}
18 DB_PATH = "mqtt_sensors.db"
19
20 LAST_SAVE_TIME = datetime.now()
21
22
23 def setup_db(db_path: str): 1 usage
24     conn = sqlite3.connect(db_path)
25     c = conn.cursor()
26     c.execute("""
27     CREATE TABLE IF NOT EXISTS sensor_data (
28         id INTEGER PRIMARY KEY AUTOINCREMENT,
29         timestamp TEXT,
30         sensor TEXT,
31         topic TEXT,
32         value REAL
33     )
34     """)
35     conn.commit()
36     return conn
37
38
39 def insert_record(conn, sensor, topic, value): 1 usage
40     ts = datetime.now().isoformat()

```

Рисунок 5.2.6 - em_data_collector.py часть 1



```
em_data_collector.py × emulator(1).py emulator.py
38     def insert_record(conn, sensor, topic, value): 1 usage
39         conn.execute(
40             "INSERT INTO sensor_data (timestamp, sensor, topic, value) VALUES (?, ?, ?, ?)",
41             (ts, sensor, topic, value),
42         )
43         conn.commit()
44
45
46     def on_connect(client, userdata, flags, rc): 3 usages
47         print("Connected with result code " + str(rc))
48         for topic in SUB_TOPICS.keys():
49             client.subscribe(topic)
50
51     def on_message(client, userdata, msg): 2 usages
52         global LAST_SAVE_TIME
53         payload = msg.payload.decode()
54         topic = msg.topic
55         param_name = SUB_TOPICS[topic]
56         RECORD_DICT[param_name] = payload
57         RECORD_DICT['time'] = str(datetime.now())
58         if (datetime.now() - LAST_SAVE_TIME).total_seconds() < 5:
59             return
60         LAST_SAVE_TIME = datetime.now()
61         for sensor, data in RECORD_DICT.items():
62             insert_record(userdata["db_conn"], sensor, msg.topic, data)
63
64         print(topic + " " + payload)
65
66 # Создание mqtt клиента и запуск сбора данных
67 def main(): 1 usage
68     conn = setup_db(DB_PATH)
69     client = mqtt.Client(userdata={"db_conn": conn})
70     client.on_connect = on_connect
71     client.on_message = on_message
72     client.connect(HOST, PORT, KEEPALIVE)
73
74     client.loop_forever()
75
76 ▶ if __name__ == "__main__":
77     main()
78
```

Рисунок 5.2.7 - em_data_collector.py часть 2

ВЫВОД

В ходе данных практических работ было проведено ознакомление с клиентской программой РиТТу: подключение и использование. Также был проведён сбор данных для файлов различных расширений и проведён графический анализ полученных данных.