|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ 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 группыИНБО-12-23 | Губарев Савва Алексеевич  Албахтин Илья Владиславович  Полиэктов Максим Александрович |
| **Проверил:** | Воронцова Евгения Константиновна |

2025 г.

**Содержание**

[Практическая работа 1 3](#_Toc210339207)

[Практическая работа 2 9](#_Toc210339208)

[Практическая работа 3 14](#_Toc210339209)

[Практическая работа 4 **Ошибка! Закладка не определена.**](#_Toc210339210)

[Вывод 29](#_Toc210339211)

# Практическая работа 5

**Часть 1. Анализ компонентов**

*Таблица 1 - Описание устройств*

| **Параметр** | **Освещенность (WB-MSW v.3)** | **Влажность (WB-MS v.2)** | **Преобразователь WB-M1W2** |
| --- | --- | --- | --- |
| **1. Название** | Настенный комбинированный датчик WB-MSW v.3 | Комбинированный датчик WB-MS v.2 | Преобразователь 1-Wire — Modbus RTU WB-M1W2 |
| **2. Тип измерения** | Цифровой | Цифровой | Исполнительное устройство (конвертер интерфейсов) |
| **3. Измеряемые параметры и диапазон** | Освещенность: 0-20000 лк (Люкс) | Относительная влажность: 0-100% | Преобразует данные с 1-Wire датчиков в протокол Modbus RTU |
| **4. Точность / Погрешность** | +-200 лк (Люкс) | +-4% | – |
| **5. Напряжение питания** | 12–24 В (постоянного тока) | 12–24 В (постоянного тока) | 12–24 В (постоянного тока) |
| **6. Уникальный идентификатор в веб-интерфейсе** | /devices/wb-msw-v3\_21/controls/Humidity | /devices/wb-  ms\_11/controls/Humidity | /devices/wb-m1w2\_14/con  trols/External Sensor 1 |
| **7. Протокол передачи данных** | Modbus RTU | Modbus RTU | Modbus RTU (выход), 1-Wire (вход) |
| **8. Интерфейс управления (шина)** | RS-485 | RS-485 | RS-485 (выход), 1-Wire (вход) |
| **9. Описание входов/выходов, схема подключения** | Выход: цифровой сигнал по шине RS-485 (Modbus RTU). Датчик объединяет несколько сенсоров. | Выход: цифровой сигнал по шине RS-485 (Modbus RTU). Датчик объединяет несколько сенсоров. | Вход: 1-Wire шина для подключения датчиков (например, DS18B20). Выход: шина RS-485 для обмена данными по протоколу Modbus RTU с контроллером. |

**Часть 2. Протоколы работы с устройствами**

**1. Modbus RTU**

* **Принцип работы:** Последовательный протокол, использующий архитектуру «ведущий–ведомый». Передача данных осуществляется в двоичном виде по шине RS-485.
* **Преимущества:** Простота, надежность, широкое распространение в промышленности и системах автоматизации, открытый стандарт.
* **Недостатки:** Низкая скорость передачи по сравнению с современными аналогами, отсутствие встроенной безопасности.
* **Сфера применения:** Промышленная автоматизация (ПЛК, датчики, приводы), системы сбора данных, умный дом (в т.ч. оборудование Wiren Board).

**2. 1-Wire**

* **Принцип работы:** Технология, позволяющая организовать сеть датчиков с использованием одного сигнального провода (и общего провода земли). Каждое устройство имеет уникальный 64-битный адрес.
* **Преимущества:** Простота подключения и разводки, низкая стоимость, возможность питания устройств от линии данных («паразитное питание»).
* **Недостатки:** Низкая скорость передачи данных, ограниченная длина линии (до 300 м), критичность к качеству монтажа.
* **Сфера применения:** Датчики температуры (DS18B20), системы мониторинга, системы контроля доступа (iButton).

**3. I²C (Inter-Integrated Circuit)**

* **Принцип работы:** Двухпроводной синхронный последовательный интерфейс (линия данных SDA и тактовая линия SCL). Поддерживает несколько «ведущих» и «ведомых» устройств на одной шине. Адресация устройств 7- или 10-битная.
* **Преимущества:** Простота (малое количество проводов), поддержка многомастерности, низкое энергопотребление, широкий ассортимент периферийных микросхем.
* **Недостатки:** Ограниченная длина (в пределах платы или устройства), чувствительность к помехам, относительно невысокая скорость.
* **Сфера применения:** Датчики (температуры, давления, акселерометры), EEPROM, ЦАП/АЦП, дисплеи, управление микросхемами в пределах одного устройства (например, материнской платы).

**4. CAN (Controller Area Network)**

* **Принцип работы:** Протокол для построения распределенных систем, основанный на сообщениях. Использует дифференциальную передачу данных по двум проводам (CAN\_H, CAN\_L). Любой узел может передавать сообщение, при коллизии приоритет определяется идентификатором сообщения.
* **Преимущества:** Высокая надежность и устойчивость к помехам, децентрализация (отсутствие главного узла), возможность работы в реальном времени, широкий диапазон скоростей.
* **Недостатки:** Более высокая сложность и стоимость по сравнению с другими протоколами, избыточен для простых задач.
* **Сфера применения:** Автомобильная электроника, промышленная автоматизация, авионика, медицинское оборудование.

**Дополнительное задание практической работы №5**

**Часть 1. Оборудование для проекта "Умная теплица"**

**Проект:** Система автоматического контроля и управления микроклиматом в теплице.

*Таблица 2 – Система полива*

| **Параметр** | **Датчик температуры и влажности** | **Реле для управления обогревателем** | **Сервопривод для форточки** |
| --- | --- | --- | --- |
| **1. Название и ссылка** | DHT22 (AM2302) | Модуль реле 5V | Сервопривод SG90 |
| **2. Тип измерения** | Цифровой | Исполнительное устройство | Исполнительное устройство |
| **3. Измеряемые параметры и диапазон** | Температура: -40...+80°C Влажность: 0...100% | Коммутация сетевого напряжения до 250 В / 10 А | Угол поворота: 0-180° |
| **4. Точность** | Температура: ±0.5°C Влажность: ±2% | — | — |
| **5. Напряжение питания** | 3.3–5 В | 5 В (цепи управления) | 4.8–6 В |
| **6. Протокол передачи данных** | Проприетарный цифровой (однопроводной) | Управление по TTL-уровню (0/5В) | Управление ШИМ-сигналом (PWM) |
| **7. Интерфейс управления (шина)** | GPIO (цифровой вход/выход) | GPIO (цифровой выход) | GPIO (ШИМ-выход) |
| **8. Описание входов/выходов, схема подключения** | 3-4 вывода: VCC, GND, DATA. Подключается напрямую к микроконтроллеру. | 3 вывода: VCC, GND, IN (управление). Силовые клеммы: COM, NO, NC. | 3 вывода: VCC (питание), GND, Signal (управляющий ШИМ). |

**Целесообразность выбора:**

* **DHT22:** Выбран за хорошее сочетание точности, цены и простоты подключения. Диапазон температур и влажности полностью перекрывает условия теплицы.
* **Модуль реле:** Позволяет безопасно управлять высоковольтным обогревателем от низковольтного контроллера (например, Arduino или ESP32). Гальваническая развязка защищает контроллер.
* **Сервопривод SG90:** Недорогой и достаточно мощный для открывания/закрывания небольшой форточки. Простота управления с помощью ШИМ.

**Часть 2. Расчет мощности нагрузки**

**Дано:**

1. Лампа накаливания (EL1): P\_лампа = 60 Вт
2. Светодиодная лента (LED1): P\_лента\_макс = 72 Вт (при питании 24 В)
3. Вентилятор (М2): U = 230 В, I = 0.1 А

**Решение:**

1. **Максимальная потребляемая мощность светодиодной ленты при питании 24 В.**
   * По условию, максимальная потребляемая мощность ленты составляет **72 Вт**. Это значение уже дано для питания 24 В.
   * **Ответ:** P\_лента = 72 Вт.
2. **Активная мощность вентилятора.**
   * Активная мощность для активной нагрузки (вентилятор) рассчитывается по формуле: P = U \* I.
   * P\_вентилятор = 230 В \* 0.1 А = 23 Вт.
   * **Ответ:** P\_вентилятор = 23 Вт.
3. **Суммарная мощность всех трех устройств при их одновременной работе.**
   * Суммарная мощность равна арифметической сумме мощностей каждого устройства.
   * P\_суммарная = P\_лампа + P\_лента + P\_вентилятор
   * P\_суммарная = 60 Вт + 72 Вт + 23 Вт = 155 Вт.
   * **Ответ:** P\_суммарная = 155 Вт.

# Практическая работа 6

Вариант №6.

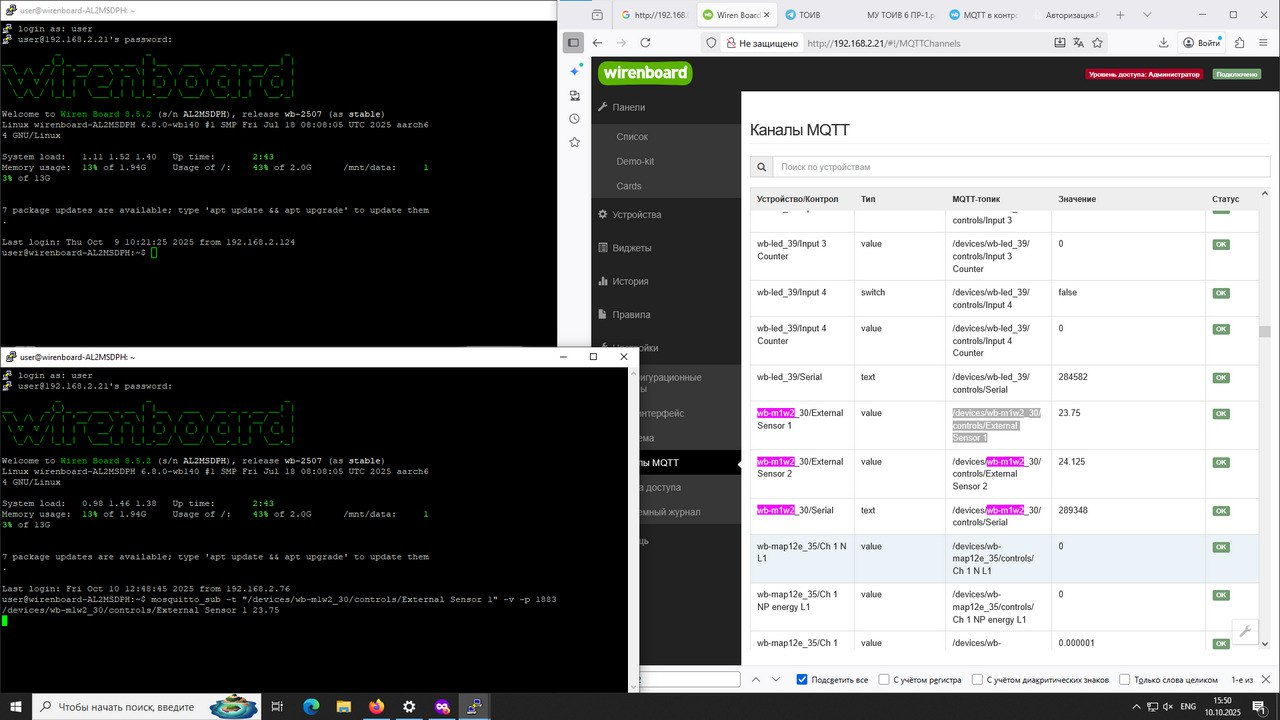


Рисунок – 1 Подключение к консоли контроллера

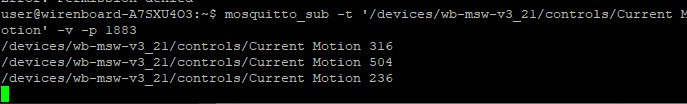


Рисунок 2 – Подписка на датчик движения устройства WB-MSW v.3



Рисунок 3 – Включение красной подсветки

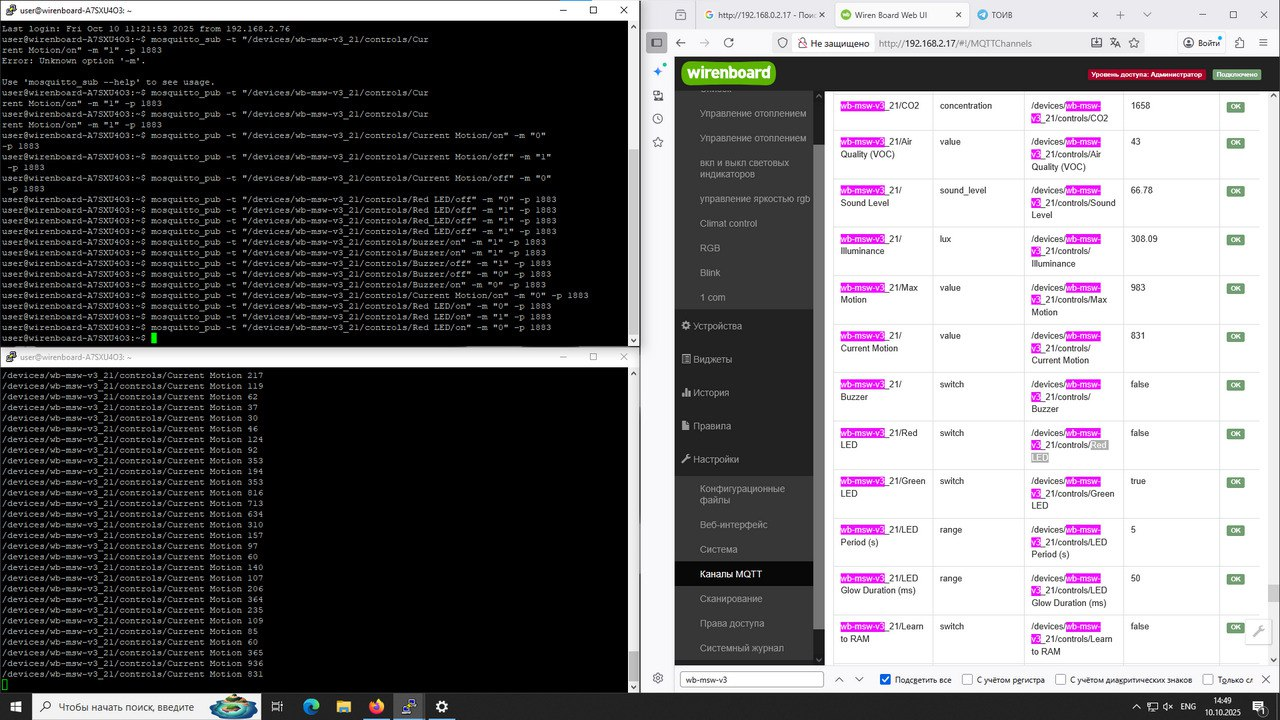


Рисунок 4 – Выключение красной подсветки

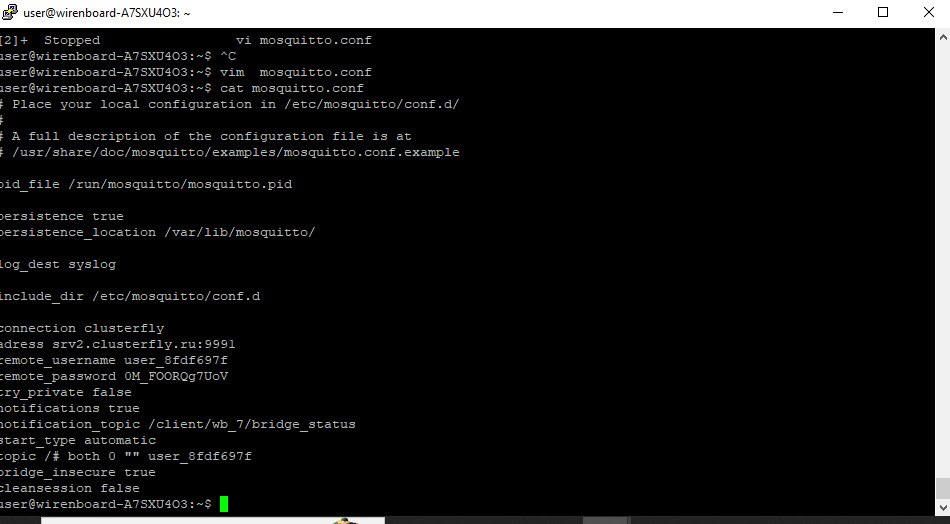


Рисунок 5 – Запись mosquitto.conf

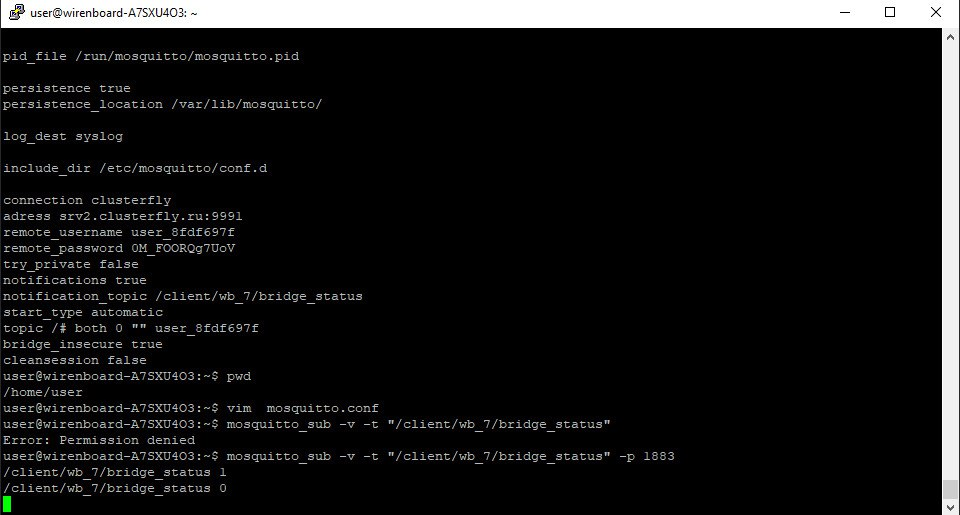


Рисунок 6 – Изменение статуса моста

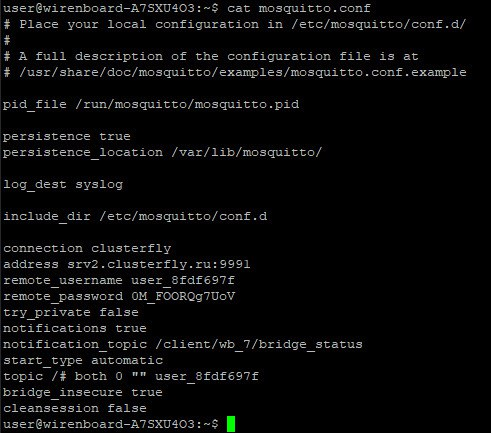


Рисунок 7 – Настройки MQTT-моста, конфигурация

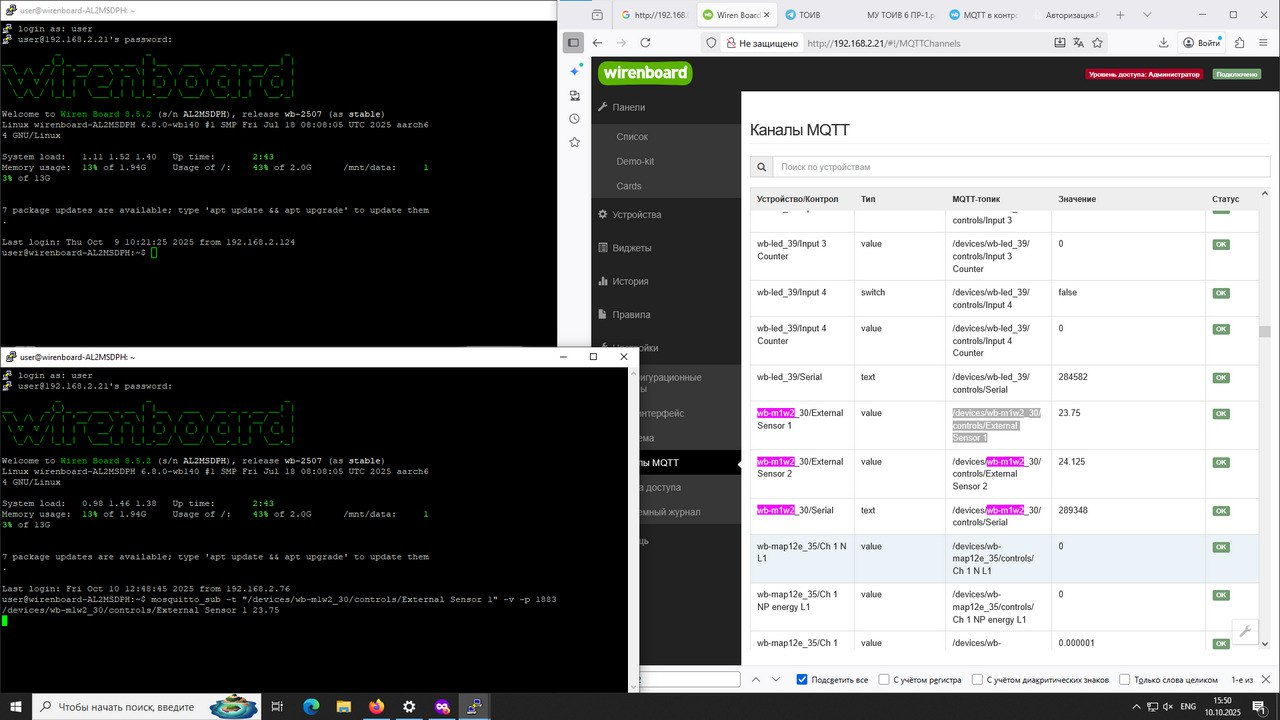


Рисунок 8 – Подписка на датчик температуры

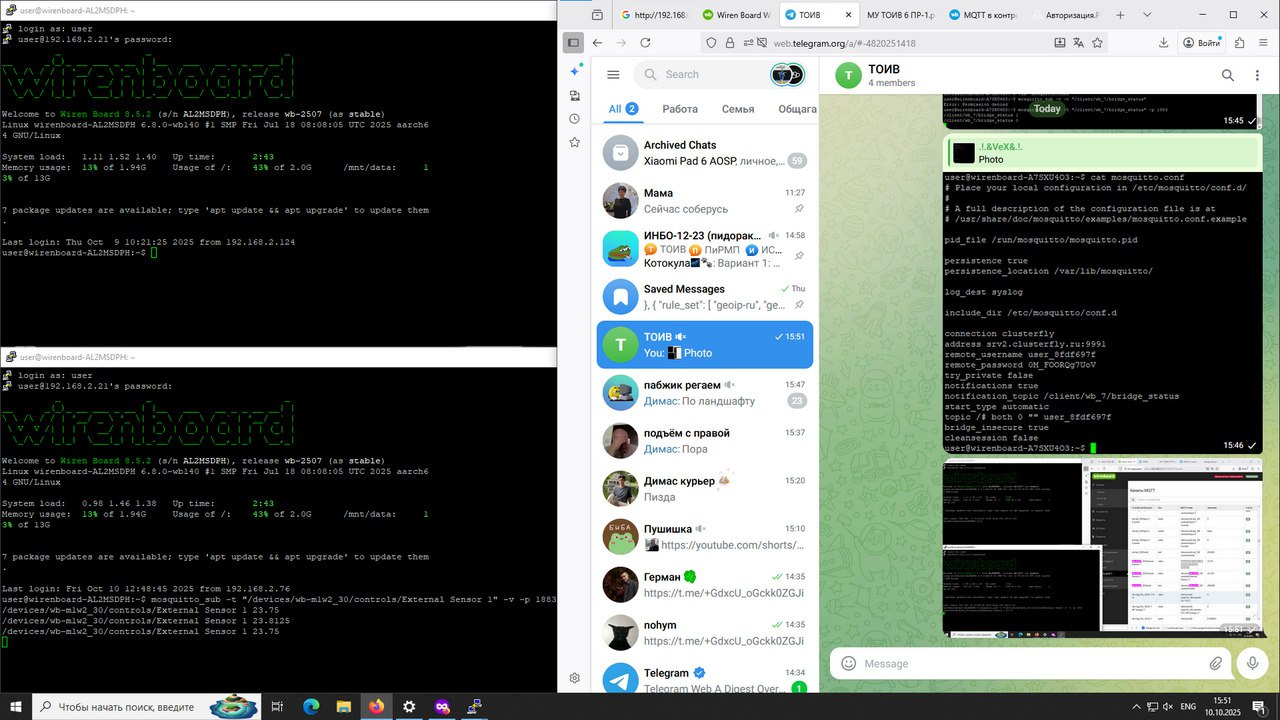


Рисунок 9 – Получение данных с датчика

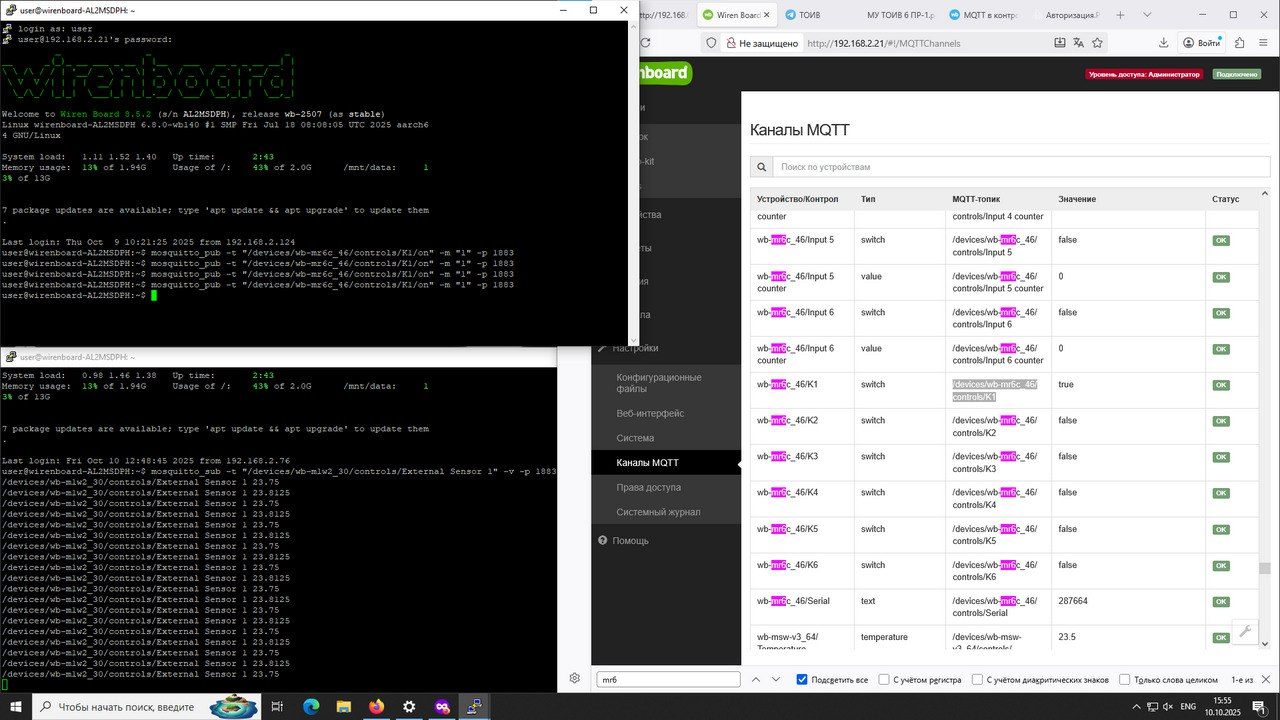


Рисунок 10 – Включённый индикатор

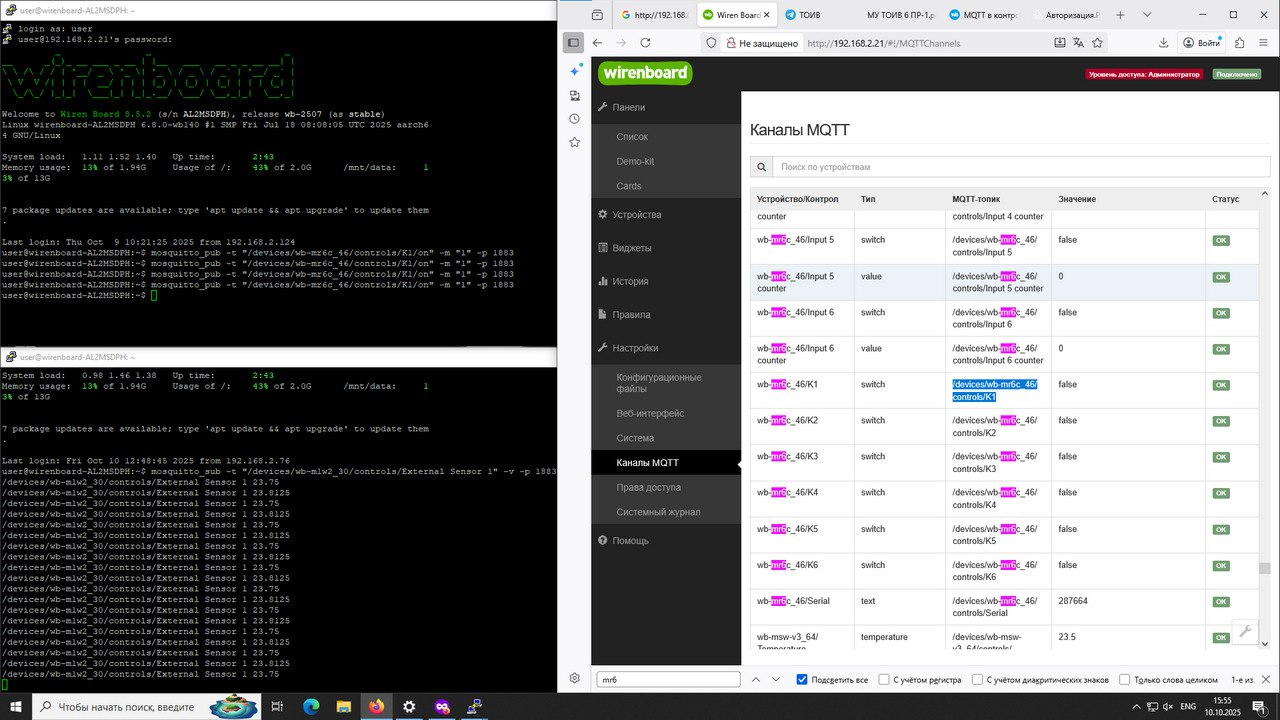


Рисунок 11 – Выключенный индикатор

# Практическая работа 7

Вариант №6

Листинг 1 – Подписка на топики и запись значений с датчиков в json и xml

import paho.mqtt.client as mqtt

import json

import xml.etree.ElementTree as ET

import threading

import time

from datetime import datetime

HOST = "192.168.2.16"

PORT = 1883

KEEPALIVE = 60

SUB\_TOPICS = {

    '/devices/wb-m1w2\_14/controls/External Sensor 2': 'temperature',

    '/devices/wb-msw-v3\_21/controls/Humidity': 'humidity',

    '/devices/wb-ms\_11/controls/Air Quality (VOC)': 'voc',

    '/devices/wb-msw-v3\_21/controls/Sound Level': 'sound'

}

JSON\_LIST = []

CURRENT\_DATA = {key: 0 for key in SUB\_TOPICS.values()}

CURRENT\_DATA['time'] = ''

CURRENT\_DATA['suitcase'] = HOST.split('.')[-1]

def on\_connect(client, userdata, flags, rc):

    print("Connected with result code", rc)

    for topic in SUB\_TOPICS:

        client.subscribe(topic)

        print(f"Subscribed to {topic}")

def on\_message(client, userdata, msg):

    payload = msg.payload.decode()

    topic = msg.topic

    key = SUB\_TOPICS[topic]

    CURRENT\_DATA[key] = payload

    CURRENT\_DATA['time'] = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    JSON\_LIST.append(CURRENT\_DATA.copy())

    print(f"{topic} → {payload}")

def save\_json():

    with open('data.json', 'w', encoding='utf-8') as f:

        json.dump(JSON\_LIST, f, ensure\_ascii=False, indent=2)

def save\_xml():

    root = ET.Element("data")

    for entry in JSON\_LIST:

        record = ET.SubElement(root, "entry")

        for key, value in entry.items():

            elem = ET.SubElement(record, key)

            elem.text = str(value)

    tree = ET.ElementTree(root)

    tree.write("data.xml", encoding="utf-8", xml\_declaration=True)

def periodic\_save():

    while True:

        if JSON\_LIST:

            save\_json()

            save\_xml()

            print("Saved data.json и data.xml")

        time.sleep(5)

def main():

    client = mqtt.Client()

    client.on\_connect = on\_connect

    client.on\_message = on\_message

    client.connect(HOST, PORT, KEEPALIVE)

    threading.Thread(target=periodic\_save, daemon=True).start()

    client.loop\_forever()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Листинг 2 – Парсер данных

import paho.mqtt.client as mqtt

import json

from datetime import datetime

# Параметры подключения к MQTT-брокеру

HOST = "192.168.1.16" # IP чемодана

PORT = 1883 # Стандартный порт подключения для Mosquitto

KEEPALIVE = 60 # Время ожидания доставки сообщения, если при отправке оно будет прeвышено, брокер будет считаться недоступным

# Словарь с топиками и собираемыми из них параметрами

SUB\_TOPICS = {

'/devices/wb-m1w2\_14/controls/External Sensor 2': 'temperature',

'/devices/wb-msw-v3\_21/controls/Humidity': 'humidity',

'/devices/wb-ms\_11/controls/Air Quality (VOC)': 'voc',

'/devices/wb-msw-v3\_21/controls/Sound Level': 'sound'

}

JSON\_LIST = []

# Создание словаря для хранения данных JSON

JSON\_DICT = {}

for value in SUB\_TOPICS.values():

JSON\_DICT[value] = 0

def on\_connect(client, userdata, flags, rc):

""" Функция, вызываемая при подключении к брокеру

Arguments:

client - Экземпляр класса Client, управляющий подключением к брокеру

userdata - Приватные данные пользователя, передаваемые при подключениии

flags - Флаги ответа, возвращаемые брокером

rc - Результат подключения, если 0, всё хорошо, в противном случае идем в документацию

"""

print("Connected with result code " + str(rc))

# Подключение ко всем заданным выше топикам

for topic in SUB\_TOPICS.keys():

client.subscribe(topic)

def on\_message(client, userdata, msg):

""" Функция, вызываемая при получении сообщения от брокера по одному из отслеживаемых топиков

Arguments:

client - Экземпляр класса Client, управляющий подключением к брокеру

userdata - Приватные данные пользователя, передаваемые при подключениии

msg - Сообщение, приходящее от брокера, со всей информацией

"""

payload = msg.payload.decode() # Основное значение, приходящее в сообщение, например, показатель температуры

topic = msg.topic # Топик, из которого пришло сообщение, поскольку функция обрабатывает сообщения из всех топиков

param\_name = SUB\_TOPICS[topic]

JSON\_DICT[param\_name] = payload

JSON\_DICT['time'] = str(datetime.now())

JSON\_LIST.append(JSON\_DICT.copy())

print(topic + " " + payload)

# Запись данных в файл

with open('data.json', 'w') as file:

json\_string = json.dumps(JSON\_LIST) # Формирование строки JSON из словаря

file.write(json\_string)

def main():

# Создание и настройка экземпляра класса Client для подключения в Mosquitto

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect(HOST, PORT, KEEPALIVE)

client.loop\_forever() # Бесконечный внутренний цикл клиента в ожидании сообщений

if \_\_name\_\_ == "\_\_main\_\_":

main()

# ПРАКТИЧЕСКАЯ РАБОТА №8

Листинг 1 – Визуализация собранных данных на python

|  |
| --- |
| **import** pandas **as** pd  **import** matplotlib.pyplot **as** plt  **import** matplotlib.dates **as** mdates  **import** os  **import** numpy **as** np  **import** json  **def** visualize\_json\_sensor\_data(file\_path="sensor\_data.json"):  # ... (unchanged part for file existence, JSON loading, initial DataFrame creation) ...  print(f"Чтение данных из JSON-файла: {file\_path}")    **try**:  **with** open(file\_path, 'r', encoding='utf-8') **as** f:  data = json.load(f)    df = pd.DataFrame(data)    df['temperature'] = pd.to\_numeric(df['temperature'], errors='coerce')  df['humidity'] = pd.to\_numeric(df['humidity'], errors='coerce')  df['voc'] = pd.to\_numeric(df['voc'], errors='coerce')  df['sound'] = pd.to\_numeric(df['sound'], errors='coerce')  df['time'] = pd.to\_datetime(df['time'], errors='coerce')    df = df.rename(columns={'time': 'datetime'})    df = df.dropna(subset=['temperature', 'humidity', 'sound', 'datetime'])    **if** df.empty:  print("После обработки данных DataFrame оказался пустым. Возможно, нет корректных данных для построения графиков.")  **return**  print("Данные успешно прочитаны и преобразованы:")  print(df.head())  print(f"Минимальные значения: Temp={df['temperature'].min()}, Hum={df['humidity'].min()}, Sound={df['sound'].min()}")  print(f"Максимальные значения: Temp={df['temperature'].max()}, Hum={df['humidity'].max()}, Sound={df['sound'].max()}")    # ... (unchanged parts for Humidity Linear Plot and Sound Histogram) ...  # --- 4. Визуализация для Температуры (Круговая диаграмма распределения) ---  fig3, ax3 = plt.subplots(figsize=(9, 9))    temp\_min = df['temperature'].min()  temp\_max = df['temperature'].max()  # --- ОТЛАДОЧНАЯ ИНФОРМАЦИЯ ---  print(f"\n--- Отладка круговой диаграммы температуры ---")  print(f"Минимальная температура: {temp\_min}")  print(f"Максимальная температура: {temp\_max}")  **if** temp\_min == temp\_max: # Если все значения температуры одинаковы  print("Все значения температуры одинаковы, круговая диаграмма не будет информативной.")  plt.close(fig3)  **return**    # Если разница между мин и макс слишком мала, используем более мелкие интервалы  **if** temp\_max - temp\_min < 1.0: # Если разброс меньше 1 градуса, используем 2 интервала  bins\_temp = np.linspace(temp\_min, temp\_max, num=3) # 2 интервала  **else**: # Иначе 3 интервала  bins\_temp = np.linspace(temp\_min, temp\_max, num=4) # 3 интервала  print(f"Предлагаемые границы интервалов (bins\_temp): {bins\_temp}")    # Убедимся, что все бины уникальны и отсортированы  bins\_temp = sorted(list(set(bins\_temp)))    final\_labels\_temp = []  **for** i **in** range(len(bins\_temp) - 1):  final\_labels\_temp.append(f'{bins\_temp[i]:.2f}-{bins\_temp[i+1]-0.01:.2f}°C') # Увеличим точность меток  print(f"Конечные метки интервалов (final\_labels\_temp): {final\_labels\_temp}")  df['temp\_category'] = pd.cut(df['temperature'], bins=bins\_temp, labels=final\_labels\_temp[:len(bins\_temp)-1], include\_lowest=**True**, right=**False**)  temp\_counts = df['temp\_category'].value\_counts().sort\_index()    print(f"Распределение по категориям (temp\_counts):\n{temp\_counts}")  temp\_counts = temp\_counts[temp\_counts > 0] # Убираем пустые категории  **if** temp\_counts.empty:  print("Не удалось создать круговую диаграмму для температуры: после фильтрации нет данных в категориях.")  plt.close(fig3)  **return**  # ... (unchanged part for pie chart plotting and saving) ...  wedges, texts, autotexts = ax3.pie(temp\_counts, autopct='%1.1f%%', startangle=90, pctdistance=0.85, colors=plt.cm.Pastel2.colors)  ax3.axis('equal')    labels\_with\_percent\_temp = [f'{label} ({percent:.1f}%)' **for** label, percent **in** zip(temp\_counts.index, temp\_counts.values / temp\_counts.sum() \* 100)]  ax3.legend(wedges, labels\_with\_percent\_temp,  title="Диапазоны температуры (°C)",  loc="center left",  bbox\_to\_anchor=(1, 0, 0.5, 1))  ax3.set\_title('Распределение показаний температуры')  plt.tight\_layout()  plt.savefig('temperature\_pie\_chart\_plot.png')  print("Сохранена круговая диаграмма для температуры: 'temperature\_pie\_chart\_plot.png'")    **except** json.JSONDecodeError **as** e:  print(f"Ошибка парсинга JSON-файла: {e}. Проверьте, что файл имеет корректную JSON-структуру.")  **except** Exception **as** e:  print(f"Произошла ошибка при обработке файла: {e}")  **if** \_\_name\_\_ == "\_\_main\_\_":  visualize\_json\_sensor\_data() |



Рисунок 14 – Итоговые диаграммы

Изображение выглядит как диаграмма, текст, линия, График

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 15 – Итоговые диаграммы

Изображение выглядит как круг, снимок экрана, текст, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 16 – Итоговые диаграммы

Листинг 2 – Визуализация собранных данных на python

|  |
| --- |
| **import** pandas **as** pd  **import** matplotlib.pyplot **as** plt  **import** matplotlib.dates **as** mdates  **import** os  **import** numpy **as** np  **import** json  **def** visualize\_wb\_msw\_v4\_data(file\_path="wb\_msw\_data.json"):  """  Читает данные из JSON-файла, содержащего показания датчиков WB-MSW v.4 (влажность, движение)  и напряжения, и строит по одному графику каждого типа для каждого параметра.  """    **if** **not** os.path.exists(file\_path):  print(f"Ошибка: Файл '{file\_path}' не найден. Убедитесь, что файл находится в той же директории или укажите полный путь.")  **return**  print(f"Чтение данных из JSON-файла: {file\_path}")    **try**:  # Чтение JSON-файла  **with** open(file\_path, 'r', encoding='utf-8') **as** f:  data = json.load(f)    # Создаем DataFrame из извлеченных данных  df = pd.DataFrame(data)    # Преобразование типов данных  df['humidity'] = pd.to\_numeric(df['humidity'], errors='coerce')  df['motion'] = pd.to\_numeric(df['motion'], errors='coerce')  df['voltage'] = pd.to\_numeric(df['voltage'], errors='coerce')  df['time'] = pd.to\_datetime(df['time'], errors='coerce')    # Переименовываем 'time' в 'datetime' для согласованности  df = df.rename(columns={'time': 'datetime'})    # Убираем строки с NaN в ключевых столбцах для графиков  df = df.dropna(subset=['humidity', 'motion', 'voltage', 'datetime'])    **if** df.empty:  print("После обработки данных DataFrame оказался пустым. Возможно, нет корректных данных для построения графиков.")  **return**  print("Данные успешно прочитаны и преобразованы:")  print(df.head())  print(f"Минимальные значения: Humidity={df['humidity'].min()}, Motion={df['motion'].min()}, Voltage={df['voltage'].min()}")  print(f"Максимальные значения: Humidity={df['humidity'].max()}, Motion={df['motion'].max()}, Voltage={df['voltage'].max()}")    # --- 2. Визуализация для Датчика влажности (Линейный график) ---  fig1, ax1 = plt.subplots(figsize=(12, 6))  ax1.plot(df['datetime'], df['humidity'], marker='o', linestyle='-', color='teal', markersize=4, label='Влажность (%)')    ax1.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m-%d %H:%M:%S'))  ax1.xaxis.set\_major\_locator(mdates.AutoDateLocator())  fig1.autofmt\_xdate()    ax1.set\_title('Изменение влажности во времени')  ax1.set\_xlabel('Время')  ax1.set\_ylabel('Влажность (%)')  ax1.grid(**True**, linestyle='--', alpha=0.7)  ax1.legend()  plt.tight\_layout()  plt.savefig('humidity\_wb\_msw\_v4\_time\_series\_plot.png')  print("\nСохранен линейный график для влажности: 'humidity\_wb\_msw\_v4\_time\_series\_plot.png'")    # --- 3. Визуализация для Датчика движения (Гистограмма частот) ---  fig2, ax2 = plt.subplots(figsize=(10, 6))    # Отфильтровываем значения движения, если 0 означает отсутствие движения, а ненулевые - наличие/интенсивность  # Если 0 - это тоже значимое состояние, можно не фильтровать  df\_motion\_filtered = df[df['motion'] >= 0] # Включаем 0, если 0 - это "нет движения"  **if** **not** df\_motion\_filtered.empty **and** df\_motion\_filtered['motion'].nunique() > 1: # Проверяем, что есть разнообразие данных  bins\_motion = np.linspace(df\_motion\_filtered['motion'].min(), df\_motion\_filtered['motion'].max(), num=max(2, df\_motion\_filtered['motion'].nunique())) # Подбираем количество бинов  **if** len(bins\_motion) == 1: # Если всего одно уникальное значение  bins\_motion = np.array([bins\_motion[0]-0.5, bins\_motion[0]+0.5]) # Создаем бин вокруг него    counts\_motion, bin\_edges\_motion, \_ = ax2.hist(df\_motion\_filtered['motion'], bins=bins\_motion, edgecolor='black', alpha=0.7, color='skyblue')    bin\_labels\_motion = [f"[{bin\_edges\_motion[i]:.1f}, {bin\_edges\_motion[i+1]:.1f}]" **for** i **in** range(len(bin\_edges\_motion)-1)]  ax2.set\_xticks([(bin\_edges\_motion[i] + bin\_edges\_motion[i+1]) / 2 **for** i **in** range(len(bin\_edges\_motion)-1)])  ax2.set\_xticklabels(bin\_labels\_motion, rotation=45, ha='right')    ax2.set\_title('Частота показаний датчика движения')  ax2.set\_xlabel('Интервалы значений движения') # Единицы измерения зависят от WB-MSW v.4  ax2.set\_ylabel('Частота записей')  ax2.grid(axis='y', linestyle='--', alpha=0.7)  plt.tight\_layout()  plt.savefig('motion\_wb\_msw\_v4\_histogram\_plot.png')  print("Сохранена гистограмма для движения: 'motion\_wb\_msw\_v4\_histogram\_plot.png'")  **else**:  print("Нет достаточных данных для построения гистограммы движения (все значения одинаковы, равны 0 или пусты).")  plt.close(fig2)  # --- 4. Визуализация для Напряжения (Круговая диаграмма распределения) ---  fig3, ax3 = plt.subplots(figsize=(9, 9))    voltage\_min = df['voltage'].min()  voltage\_max = df['voltage'].max()  # Отладочная информация  print(f"\n--- Отладка круговой диаграммы напряжения ---")  print(f"Минимальное напряжение: {voltage\_min}")  print(f"Максимальное напряжение: {voltage\_max}")  **if** voltage\_min == voltage\_max: # Если все значения напряжения одинаковы  print("Все значения напряжения одинаковы, круговая диаграмма не будет информативной.")  plt.close(fig3)  **return**    # Определяем категории для напряжения  # Пороги нужно адаптировать под ожидаемые значения (например, 3.3В, 5В, 12В или 24В на стенде)  # Предположим, что "норма" может быть 3.3-5В, в зависимости от того, что измеряется.  # В ваших данных есть "4.109", что указывает на низковольтное питание.    # Динамическое создание бинов: 3-4 бина в зависимости от разброса  **if** voltage\_max - voltage\_min < 0.5: # Очень маленький разброс  bins\_voltage = np.linspace(voltage\_min, voltage\_max, num=2) # 1 интервал  **elif** voltage\_max - voltage\_min < 2: # Небольшой разброс  bins\_voltage = np.linspace(voltage\_min, voltage\_max, num=3) # 2 интервала  **else**: # Большой разброс  bins\_voltage = np.linspace(voltage\_min, voltage\_max, num=4) # 3 интервала  bins\_voltage = sorted(list(set(bins\_voltage))) # Убедимся, что уникальны и отсортированы    final\_labels\_voltage = []  **for** i **in** range(len(bins\_voltage) - 1):  final\_labels\_voltage.append(f'{bins\_voltage[i]:.2f}-{bins\_voltage[i+1]-0.001:.2f}В') # Точность до сотых  df['voltage\_category'] = pd.cut(df['voltage'], bins=bins\_voltage, labels=final\_labels\_voltage[:len(bins\_voltage)-1], include\_lowest=**True**, right=**False**)  voltage\_counts = df['voltage\_category'].value\_counts().sort\_index()    print(f"Распределение по категориям напряжения (voltage\_counts):\n{voltage\_counts}")    voltage\_counts = voltage\_counts[voltage\_counts > 0] # Убираем пустые категории  **if** voltage\_counts.empty:  print("Не удалось создать круговую диаграмму для напряжения: после фильтрации нет данных в категориях.")  plt.close(fig3)  **return**  wedges, texts, autotexts = ax3.pie(voltage\_counts, autopct='%1.1f%%', startangle=90, pctdistance=0.85, colors=plt.cm.Pastel2.colors)  ax3.axis('equal')    labels\_with\_percent\_voltage = [f'{label} ({percent:.1f}%)' **for** label, percent **in** zip(voltage\_counts.index, voltage\_counts.values / voltage\_counts.sum() \* 100)]  ax3.legend(wedges, labels\_with\_percent\_voltage,  title="Диапазоны напряжения (В)",  loc="center left",  bbox\_to\_anchor=(1, 0, 0.5, 1))  ax3.set\_title('Распределение показаний напряжения')  plt.tight\_layout()  plt.savefig('voltage\_distribution\_plot.png')  print("Сохранена круговая диаграмма для напряжения: 'voltage\_distribution\_plot.png'")    # plt.show() # Раскомментируйте, если хотите видеть графики на экране    **except** json.JSONDecodeError **as** e:  print(f"Ошибка парсинга JSON-файла: {e}. Проверьте, что файл имеет корректную JSON-структуру.")  **except** Exception **as** e:  print(f"Произошла ошибка при обработке файла: {e}")  # Вызов функции для выполнения  **if** \_\_name\_\_ == "\_\_main\_\_":  visualize\_wb\_msw\_v4\_data() |

Изображение выглядит как линия, График, текст, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 17 – Итоговые диаграммы

Изображение выглядит как диаграмма, линия, График, текст

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 18 – Итоговые диаграммы

Изображение выглядит как круг, снимок экрана, диаграмма, текст

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 18 – Итоговые диаграммы

# Вывод

В ходе выполнения лабораторной работы были протестированы основные функции учебного стенда: контроль наличия сетевого напряжения, определение повышенного энергопотребления, работа автоматов и внешних силовых устройств, мониторинг качества воздуха и система защиты от протечек. Были реализованы примеры диммирования лампы и вентилятора, демонстрирующие управление нагрузкой через контроллер. Дополнительно написаны сценарии автоматизации: включение/выключение вентилятора по датчику движения и изменение высоты звукового сигнала в зависимости от яркости лампы.