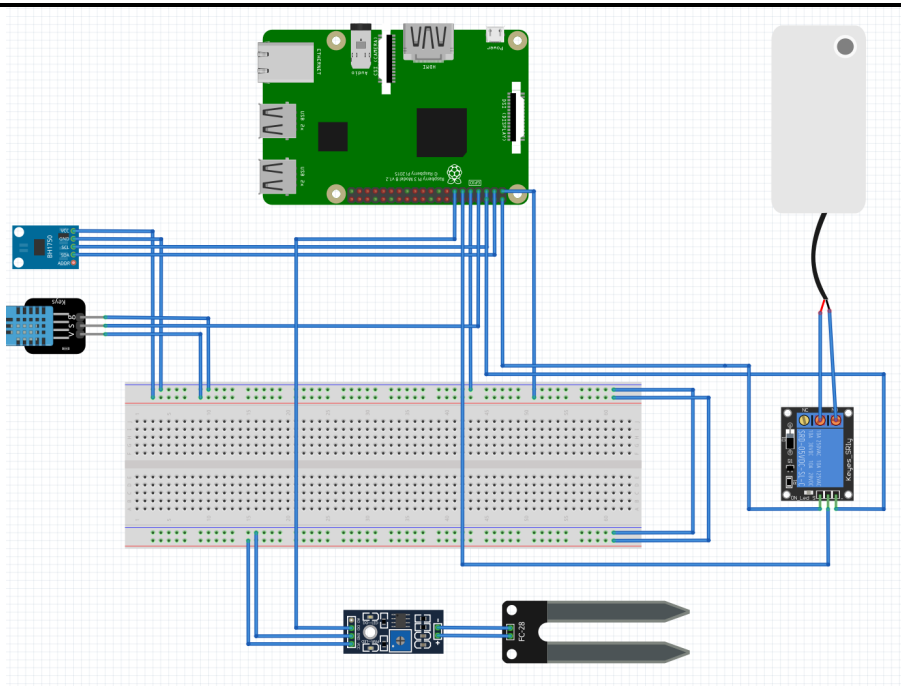


[Z7] KARTA PROJEKTU

Grupa laboratoryjna: L11 Podgrupa: 4	Paweł Kolec (155 873)	
	Adam Nowacki (155 838)	
	Prowadzący zajęcia:	dr inż. Ariel Antonowicz
System monitorowania i sterowania podlewaniami roślin z wykorzystaniem Raspberry Pi		
CEL PROJEKTU	<p>Projekt zakłada stworzenie systemu wspierającego pielęgnację roślin poprzez monitorowanie warunków środowiskowych i automatyczne sterowanie podlewaniami. Wykorzystuje czujniki temperatury i wilgotności powietrza (DHT11), nasłonecznienia (BH1750) oraz wilgotności gleby, a dane w czasie rzeczywistym są prezentowane na stronie w formie wykresów.</p> <p>System automatycznie włącza pompkę, gdy wilgotność gleby spadnie poniżej ustalonego poziomu, a także umożliwia ręczne podlewanie za pomocą przycisku na stronie. Dzięki Raspberry Pi i integracji z przekaźnikiem i czujnikami, zapewnia zdalne sterowanie, oszczędność wody i optymalne warunki dla roślin.</p>	
SCHEMAT POGLĄDOWY		
		
WYKORZYSTANA PLATFORMA SPRZĘTOWA, ELEMENTY POMIAROWE I WYKONAWCZE	Raspberry Pi 3B+, Czujnik temperatury i wilgotności powietrza DHT11, Czujnik nasłonecznienia BH1750, Czujnik wilgotności gleby z modulem LM393, Pompka wodna DC 6V, Przełącznik SMD-03VDC-SL-C	

1. Cel i zakres projektu.

System monitorowania i sterowania podlewaniem roślin z wykorzystaniem Raspberry Pi:

Celem projektu jest stworzenie zintegrowanego systemu wspierającego pielęgnację roślin poprzez monitorowanie warunków środowiskowych oraz automatyczne sterowanie procesem podlewania. System ten ma na celu ułatwienie opieki nad roślinami, zwiększenie efektywności zużycia wody oraz poprawę kondycji roślin dzięki precyzyjnemu dostosowaniu poziomu nawodnienia do ich potrzeb.

Zakres realizacji – komponenty sprzętowe:

1. Raspberry Pi 3B+: Stanowi centralną jednostkę sterującą, obsługującą połączenia z czujnikami, przekaźnikiem i stroną internetową. Umożliwia analizę danych w czasie rzeczywistym oraz sterowanie pompą wodną.

2. Czujnik temperatury i wilgotności powietrza (DHT11):

Monitoruje temperaturę oraz wilgotność powietrza, co pozwala na ocenę warunków mikroklimatycznych.

3. Czujnik nasłonecznienia (BH1750):

Mierzy poziom oświetlenia, dostarczając informacji o ekspozycji roślin na światło.

4. Czujnik wilgotności gleby z modułem LM393:

Umożliwia precyzyjne monitorowanie wilgotności podłoża, co stanowi kluczowy parametr decydujący o konieczności podlewania.

5. Pompka wodna DC 6V i przekaźnik SRD-03VDC-SL-C:

Pompka wodna realizuje proces podlewania, natomiast przekaźnik zapewnia bezpieczne przełączanie pompy na podstawie sygnałów sterujących.

Zakres realizacji – aspekty programistyczne:

1. Oprogramowanie Raspberry Pi:

Implementacja kodu w języku Python, który obsługuje odczyty z czujników, analizuje dane i steruje przekaźnikiem. Oprogramowanie zawiera algorytm porównujący bieżący poziom wilgotności gleby z ustalonym progiem. Dodatkowo, wszystkie zbierane dane (wilgotność gleby, temperatura, wilgotność powietrza, poziom nasłonecznienia) są przechowywane w bazie danych SQLite. Pozwala to na ich archiwizację, analizę historyczną oraz wizualizację na stronie w formie wykresów. Dzięki temu użytkownik może śledzić zmiany warunków środowiskowych w dłuższym okresie i lepiej dostosować parametry systemu do potrzeb roślin..

2. Interfejs użytkownika:

Aplikacja webowa działająca na Raspberry Pi umożliwia:

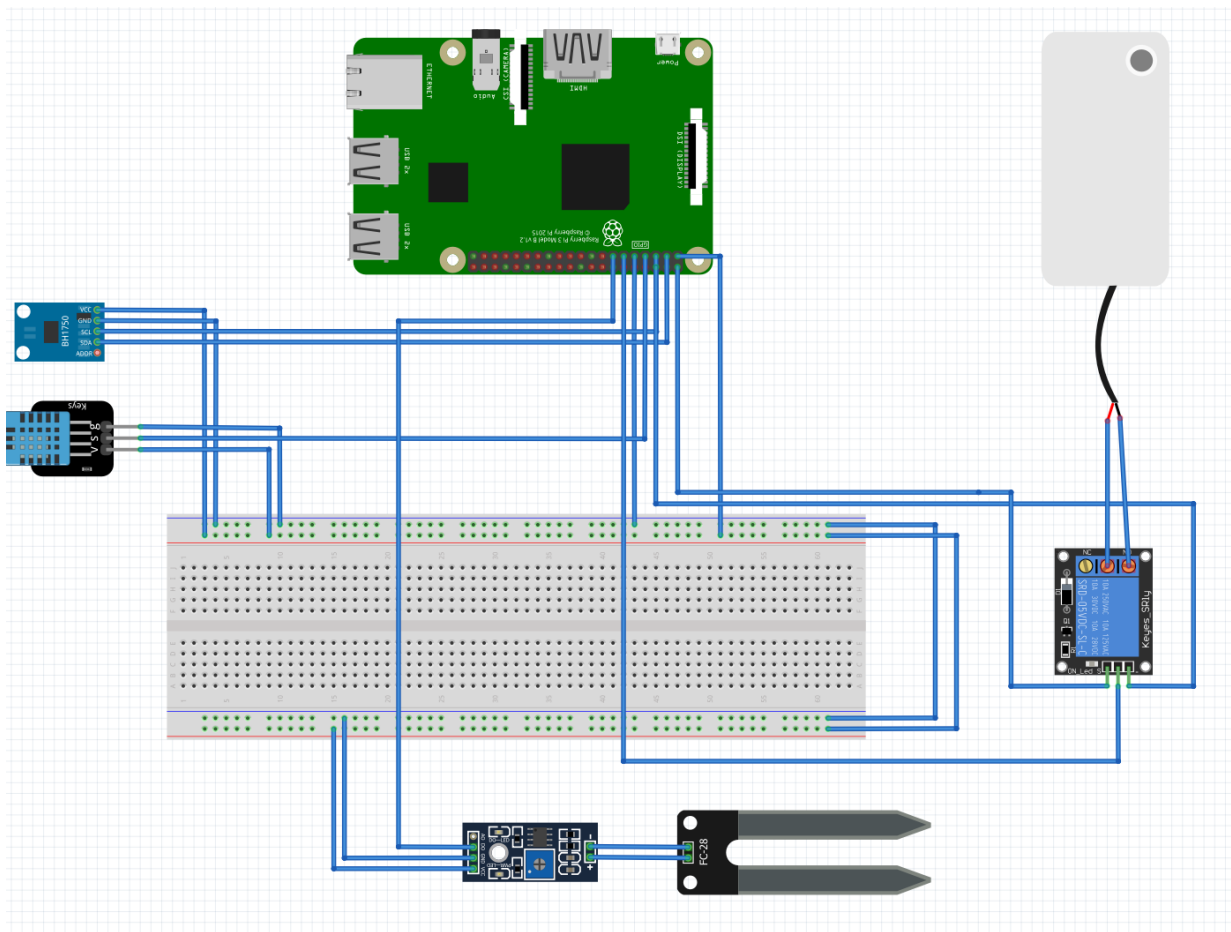
Podgląd parametrów środowiskowych (nasłonecznienie, temperatura i wilgotność powietrza) w formie czytelnych wykresów aktualizowanych w czasie rzeczywistym.

Zdalne uruchomienie ręcznego podlewania poprzez przycisk na stronie.

3. Komunikacja i integracja:

System wykorzystuje protokoły komunikacyjne (np. GPIO dla obsługi przekaźnika) oraz biblioteki umożliwiające integrację z czujnikami (np. Adafruit_DHT, smbus)

2.1 Schemat połączeniowy:



2.2 Schemat bazy danych:

Kolumna	Typ	Opis
id	INTIGER	Klucz główny, automatycznie zwiększany (PRIMARY KEY).
timestamp	DATETIME	Znacznik czasu dla każdego wpisu, domyślnie aktualny czas.
temperature	REAL	Temperatura odczytana z czujnika DHT11 (w stopniach C).
humidity	REAL	Wilgotność powietrza z czujnika DHT11 (w procentach).
light_level	REAL	Poziom nasłonecznienia z czujnika BH1750 (w luksach).

2. Założenia projektowe a ich realizacja.

Założenia projektu:

Celem projektu było stworzenie systemu, który:

1. Monitoruje kluczowe parametry środowiskowe (temperatura, wilgotność powietrza, nasłonecznienie, wilgotność gleby) za pomocą czujników.
2. Automatycznie steruje podlewaniem roślin na podstawie odczytów z czujników wilgotności gleby.
3. Udostępnia dane w czasie rzeczywistym oraz umożliwia ręczne uruchamianie podlewania za pośrednictwem interfejsu webowego.
4. Archiwizuje dane w bazie SQLite, pozwalając na późniejszą analizę i wizualizację.

Co udało się zrealizować:

- **Sprzęt i czujniki:** Zintegrowano Raspberry Pi z czujnikami DHT11 (temperatura, wilgotność), BH1750 (nasłonecznienie) oraz czujnikiem wilgotności gleby. System działał poprawnie, zbierając dane z czujników.
- **Sterowanie podlewaniem:** Udało się wdrożyć automatyczne sterowanie pompą, która uruchamiała się, gdy wilgotność gleby spadała poniżej ustalonego poziomu. Dodatkowo, użytkownik mógł ręcznie uruchomić podlewanie przez interfejs webowy.
- **Baza danych:** Dane z czujników były zapisywane w bazie SQLite, a użytkownik mógł je przeglądać na stronie w formie wykresów.
- **Interfejs webowy:** Zbudowano stronę, która pozwalała na monitorowanie danych w czasie rzeczywistym oraz generowanie wykresów historycznych.

Czego nie udało się zrealizować:

- **Dokładność czujników:** Czujnik wilgotności gleby, z powodu swojej wrażliwości na warunki atmosferyczne, wykazywał pewną niestabilność i nieprecyzyjność w długotrwałym użytkowaniu.
- **Brak zdalnego dostępu:** Ze względu na ograniczenia zasobowe, nie udało się zaimplementować pełnej funkcji zdalnego dostępu do systemu spoza sieci lokalnej.

Propozycje rozwoju:

1. **Zastosowanie lepszych czujników:** Zamiana czujnika wilgotności gleby na bardziej precyzyjny, np. pojemnościowy, który nie jest tak podatny na korozję i zapewnia dokładniejsze pomiary.
2. **Zdalny dostęp:** Integracja systemu z chmurą, umożliwiającą dostęp zdalny do danych i sterowanie systemem przez internet.
3. **Nowe funkcje w interfejsie:** Powiadomienia (e-mail/SMS) o konieczności interwencji, np. gdy poziom wody w zbiorniku spadnie poniżej ustalonego progu.
4. **AI i prognozy pogodowe:** Wykorzystanie algorytmów sztucznej inteligencji do analizy danych historycznych i prognozowania potrzeb podlewania, co mogłoby poprawić efektywność systemu.

Projekt spełnił swoje główne cele, jednak z pewnością ma duży potencjał do dalszego rozwoju, który pozwoliłby na jeszcze lepszą automatyzację i zdalne zarządzanie podlewaniem roślin.

3. Listing kodu.

```
<script>
  // Funkcja do pobrania najnowszych danych z serwera
  function fetchLatestData() {
    // Wysyłamy zapytanie GET do endpointu '/latest-data'
    fetch('/latest-data')
      .then(response => response.json()) // Parsujemy odpowiedź jako JSON
      .then(data => {
        // Aktualizujemy dane na stronie w odpowiednich elementach
        document.getElementById('temperature').textContent = data.temperature + ' °C';
// Wyświetlamy temperaturę
        document.getElementById('humidity').textContent = data.humidity + '
%'; // Wyświetlamy wilgotność
        document.getElementById('light').textContent = data.light_level + '
lux'; // Wyświetlamy poziom światła
      })
      .catch(error =>
        // Obsługa błędów, jeśli zapytanie do serwera się nie powiedzie
        console.log('Błąd pobierania danych:', error)
      );
  }

  // Funkcja do ręcznego podlewania roślin
  function waterPlants() {
    // Wysyłamy żądanie POST do endpointu '/api/water'
    fetch('/api/water', { method: 'POST' })
      .then(response => response.json()) // Parsujemy odpowiedź jako JSON
      .then(data => {
        // Wyświetlamy komunikat o statusie podlewania
        document.getElementById('statusMessage').textContent = data.message;
        // Czyścimy komunikat po 5 sekundach
        setTimeout(() => {
          document.getElementById('statusMessage').textContent = '';
        }, 5000);
      })
      .catch(error =>
        // Obsługa błędów, jeśli podlewanie się nie powiedzie
        console.log('Błąd podlewania:', error)
      );
  }

  // Funkcja do przejścia na stronę z wykresami
  function goToCharts() {
    // Zmieniamy lokalizację na '/charts', co powoduje przejście na nową stronę
    window.location.href = '/charts';
  }
}
```

```

}

// Funkcja do odświeżenia strony
function refreshPage() {
    // Odświeżamy bieżącą stronę
    location.reload();
}

// Funkcja uruchamiana po załadowaniu strony
window.onload = function() {
    // Pobieramy najnowsze dane z serwera przy starcie strony
    fetchLatestData();

    // Podłączamy funkcję waterPlants do przycisku o ID 'waterButton'
    document.getElementById('waterButton').addEventListener('click', waterPlants);
    // Podłączamy funkcję goToCharts do przycisku o ID 'chartsButton'
    document.getElementById('chartsButton').addEventListener('click', goToCharts);
    // Podłączamy funkcję refreshPage do przycisku o ID 'refreshButton'
    document.getElementById('refreshButton').addEventListener('click', refreshPage);

    // Automatycznie odświeżamy dane co 3 sekundy
    setInterval(fetchLatestData, 3000);
};
</script>

```

```

# --- Funkcja inicjalizująca bazę danych ---
def init_db():
    """Tworzy bazę danych, jeśli jeszcze nie istnieje."""
    # Nawiązanie połączenia z bazą danych (tworzenie jej, jeśli nie istnieje)
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()

    # Tworzymy tabelę 'sensor_data', jeśli jeszcze nie istnieje
    # Tabela zawiera kolumny: id (klucz główny, autoinkrementacja), timestamp (czas
    wstawienia), temperature, humidity, light_level
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS sensor_data (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
            temperature REAL,
            humidity REAL,
            light_level REAL
        )
    """)
    # Zatwierdzamy zmiany (commit)
    conn.commit()
    # Zamykanie połączenia z bazą danych

```

```

conn.close()

# --- Funkcja wstawiająca dane do bazy ---
def insert_data(temperature, humidity, light_level):
    """Zapisuje dane do bazy."""
    # Nawiązanie połączenia z bazą danych
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()

    # Wstawiamy dane (temperatura, wilgotność, poziom światła) do tabeli 'sensor_data'
    cursor.execute("""
        INSERT INTO sensor_data (temperature, humidity, light_level)
        VALUES (?, ?, ?)
    """, (temperature, humidity, light_level))
    # Zatwierdzamy zmiany
    conn.commit()
    # Zamykanie połączenia z bazą danych
    conn.close()

# --- Funkcja pobierająca dane z bazy ---
def get_data(limit=100):
    """Pobiera ostatnie dane z bazy."""
    # Nawiązanie połączenia z bazą danych
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()

    # Pobieramy dane z tabeli 'sensor_data', posortowane po timestamp (najpierw najnowsze)
    # Ograniczamy wynik do ostatnich 'limit' rekordów
    cursor.execute("""
        SELECT timestamp, temperature, humidity, light_level
        FROM sensor_data
        ORDER BY timestamp DESC
        LIMIT ?
    """, (limit,))
    # Zapisujemy wyniki zapytania
    data = cursor.fetchall()
    # Zamykanie połączenia z bazą danych
    conn.close()
    # Zwracamy pobrane dane
    return data

# --- Funkcja pobierająca najnowsze dane z bazy ---
def get_latest_data():
    """Pobiera najnowsze dane z bazy."""
    # Nawiązanie połączenia z bazą danych
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()

```

```

# Pobieramy tylko najnowszy rekord (z posortowanych po 'timestamp' danych)
cursor.execute("""
    SELECT timestamp, temperature, humidity, light_level
    FROM sensor_data
    ORDER BY timestamp DESC
    LIMIT 1
""")
# Zapisujemy pojedynczy rekord (najnowszy)
data = cursor.fetchone()
# Zamykanie połączenia z bazą danych
conn.close()
# Zwracamy pojedynczy rekord
return data

```

```

<script>
// Funkcja do pobrania najnowszych danych z serwera
function fetchLatestData() {
    // Wysyłamy zapytanie GET do endpointu '/latest-data'
    fetch('/latest-data')
    .then(response => response.json()) // Parsujemy odpowiedź jako JSON
    .then(data => {
        // Aktualizujemy dane na stronie w odpowiednich elementach
        document.getElementById('temperature').textContent = data.temperature + ' °C';
// Wyświetlamy temperaturę
        document.getElementById('humidity').textContent = data.humidity + '
%'; // Wyświetlamy wilgotność
        document.getElementById('light').textContent = data.light_level + '
lux'; // Wyświetlamy poziom światła
    })
    .catch(error =>
        // Obsługa błędów, jeśli zapytanie do serwera się nie powiedzie
        console.log('Błąd pobierania danych:', error)
    );
}

// Funkcja do ręcznego podlewania roślin
function waterPlants() {
    // Wysyłamy żądanie POST do endpointu '/api/water'
    fetch('/api/water', { method: 'POST' })
    .then(response => response.json()) // Parsujemy odpowiedź jako JSON
    .then(data => {
        // Wyświetlamy komunikat o statusie podlewania
        document.getElementById('statusMessage').textContent = data.message;
        // Czyścimy komunikat po 5 sekundach
        setTimeout(() => {
            document.getElementById('statusMessage').textContent = '';

```



```

        }, 5000);
    })
    .catch(error =>
        // Obsługa błędów, jeśli podlewanie się nie powiedzie
        console.log('Błąd podlewania:', error)
    );
}

// Funkcja do przejścia na stronę z wykresami
function goToCharts() {
    // Zmieniamy lokalizację na '/charts', co powoduje przejście na nową stronę
    window.location.href = '/charts';
}

// Funkcja do odświeżenia strony
function refreshPage() {
    // Odświeżamy bieżącą stronę
    location.reload();
}

// Funkcja uruchamiana po załadowaniu strony
window.onload = function() {
    // Pobieramy najnowsze dane z serwera przy starcie strony
    fetchLatestData();

    // Podłączamy funkcję waterPlants do przycisku o ID 'waterButton'
    document.getElementById('waterButton').addEventListener('click', waterPlants);
    // Podłączamy funkcję goToCharts do przycisku o ID 'chartsButton'
    document.getElementById('chartsButton').addEventListener('click', goToCharts);
    // Podłączamy funkcję refreshPage do przycisku o ID 'refreshButton'
    document.getElementById('refreshButton').addEventListener('click', refreshPage);

    // Automatycznie odświeżamy dane co 3 sekundy
    setInterval(fetchLatestData, 3000);
};
</script>

```

```

# --- Funkcja monitorująca wilgotność gleby ---
def monitor_soil_moisture():
    """Monitoruje wilgotność gleby i włącza pompę, gdy jest ona zbyt niska."""
    while True:
        # Sprawdzamy stan czujnika wilgotności gleby (jeśli wilgotność jest niska, czujnik
        zwróci HIGH)
        if GPIO.input(SOIL_SENSOR_PIN) == GPIO.HIGH: # Niska wilgotność
            # Włączamy sygnał na porcie 22, który może sterować dodatkowym układem
            GPIO.output(BUTTON_CONTROL_PIN, GPIO.HIGH) # Włącz sygnał na porcie 22
            # Włączamy pompę, aby podlewać rośliny

```

```

        GPIO.output(PUMP_PIN, GPIO.HIGH) # Włącz pompę
        time.sleep(2) # Pompa działa przez 2 sekundy
        # Po 2 sekundach wyłączamy pompę
        GPIO.output(PUMP_PIN, GPIO.LOW) # Wyłącz pompę
        # Wyłączamy sygnał na porcie 22, kończąc proces sterowania
        GPIO.output(BUTTON_CONTROL_PIN, GPIO.LOW) # Wyłącz sygnał na porcie 22
        time.sleep(5) # Sprawdzaj stan wilgotności co 5 sekund

# Uruchamiamy funkcję monitorującą wilgotność gleby w osobnym wątku, aby działała
# równoległe z głównym procesem
Thread(target=monitor_soil_moisture, daemon=True).start()

# --- Trasy (routes) Flask ---

@app.route("/")
def index():
    """Strona główna (renderuje widok index.html)."""
    return render_template("index.html")

@app.route("/latest-data")
def latest_data():
    """API do pobierania najnowszych danych z czujników (w formacie JSON)."""
    data = get_latest_data()
    if data:
        timestamp, temperature, humidity, light_level = data
        # Zwracamy dane w formacie JSON
        return jsonify({
            "timestamp": timestamp,
            "temperature": temperature,
            "humidity": humidity,
            "light_level": light_level
        })
    # Jeśli brak danych, zwracamy pusty obiekt JSON
    return jsonify({})

@app.route("/charts")
def charts():
    """Strona z wykresami (renderuje widok charts.html)."""
    data = get_data(limit=100) # Pobieramy ostatnie 100 rekordów
    return render_template("charts.html", data=data)

@app.route("/api/data")
def api_data():
    """API do pobierania danych z bazy w formacie JSON."""
    data = get_data(limit=100) # Pobieramy ostatnie 100 rekordów
    return jsonify(data)

```

```

@app.route("/api/data-paginated")
def api_data_paginated():
    """API do pobierania danych z obsługą stronicowania (z parametrów limit i offset)."""
    limit = int(request.args.get("limit", 10)) # Domyślnie 10 rekordów na stronę
    offset = int(request.args.get("offset", 0)) # Domyślnie zaczynamy od pierwszego
    rekordu

    # Nawiązanie połączenia z bazą danych
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()
    # Wykonujemy zapytanie z parametrami limitu i offsetu
    cursor.execute("""
        SELECT timestamp, temperature, humidity, light_level
        FROM sensor_data
        ORDER BY timestamp DESC
        LIMIT ? OFFSET ?
    """, (limit, offset))
    data = cursor.fetchall()
    # Zamykanie połączenia
    conn.close()

    # Zwracamy dane w formacie JSON
    return jsonify(data)

@app.route("/api/water", methods=["POST"])
def water_plants():
    """API do ręcznego podlewania roślin za pomocą przycisku na stronie (włącza pompę)."""
    try:
        # Włączamy pompę, aby podlewać rośliny
        GPIO.output(PUMP_PIN, GPIO.HIGH) # Włącz pompę
        GPIO.output(BUTTON_CONTROL_PIN, GPIO.HIGH) # Włącz sygnał na porcie 22
        time.sleep(2) # Pompa działa przez 2 sekundy
        # Wyłączamy pompę po zakończeniu podlewania
        GPIO.output(PUMP_PIN, GPIO.LOW) # Wyłącz pompę
        GPIO.output(BUTTON_CONTROL_PIN, GPIO.LOW) # Wyłącz sygnał na porcie 22
        # Zwracamy odpowiedź w formacie JSON z komunikatem sukcesu
        return jsonify({"message": "Podlewanie zakończone pomyślnie!"}), 200
    except Exception as e:
        # Jeśli wystąpi błąd, zwracamy odpowiedź z komunikatem o błędzie
        return jsonify({"message": f"Błąd podlewania: {str(e)}"}), 500

# --- Inicjalizacja ---
if __name__ == "__main__":
    # Inicjalizujemy bazę danych przed uruchomieniem aplikacji
    init_db()

    # Uruchamiamy wątek w tle, który będzie zbierał dane z czujników co 4 minuty

```

```

from threading import Thread
Thread(target=background_task, daemon=True).start()

# Uruchamiamy aplikację Flask na porcie 1234
app.run(host="0.0.0.0", port=1234)

```

```

// Przykładowe dane dla wykresu (można je zaktualizować z backendu)
const humidityData = [30, 40, 50, 60, 70, 80, 90];
const temperatureData = [22, 23, 24, 25, 26, 27, 28];
const lightData = [100, 200, 300, 400, 500, 600, 700];

const labels = ['0', '1', '2', '3', '4', '5', '6']; // Etykiety czasowe

// Tworzenie wykresu wilgotności
const humidityChart = new Chart(document.getElementById('humidityChart'), {
  type: 'line', // Typ wykresu (linie)
  data: {
    labels: labels, // Etykiety
    datasets: [{
      label: 'Wilgotność', // Opis serii danych
      data: humidityData, // Dane do wykresu
      borderColor: 'rgba(75, 192, 192, 1)', // Kolor linii
      fill: false // Brak wypełnienia pod wykresem
    }]
  }
});

// Tworzenie wykresu temperatury
const temperatureChart = new Chart(document.getElementById('temperatureChart'), {
  type: 'line',
  data: {
    labels: labels,
    datasets: [{
      label: 'Temperatura (°C)', // Opis serii danych
      data: temperatureData,
      borderColor: 'rgba(255, 99, 132, 1)',
      fill: false
    }]
  }
});

// Tworzenie wykresu nasłonecznienia
const lightChart = new Chart(document.getElementById('lightChart'), {
  type: 'line',
  data: {
    labels: labels,
    datasets: [{

```

```

        label: 'Światło (lux)',
        data: lightData,
        borderColor: 'rgba(255, 159, 64, 1)',
        fill: false
    }]
}
});

let currentOffset = 0; // Zaczynamy od najnowszych danych
const limit = 10;     // Liczba punktów na stronę

// Funkcja do pobierania danych wykresu z backendu
function fetchChartData(offset) {
    // Zapytanie o dane z API z parametrami limitu i offsetu
    fetch(`/api/data-paginated?limit=${limit}&offset=${offset}`)
        .then(response => response.json()) // Odbieramy dane w formacie JSON
        .then(data => {
            if (data.length === 0 && offset > 0) {
                // Jeśli brak danych do wyświetlenia, blokujemy przycisk "Poprzednie"
                document.getElementById("prevBtn").disabled = true;
            } else {
                // Przekształcamy dane, aby dostosować je do wykresów
                const labels = data.map(row => row[0]).reverse(); // Czas w odwrotnej
                kolejności

                const temperatures = data.map(row => row[1]).reverse(); // Temperatura
                const humidities = data.map(row => row[2]).reverse(); // Wilgotność
                const lightLevels = data.map(row => row[3]).reverse(); // Nasłonecznienie

                // Aktualizacja wykresów
                temperatureChart.data.labels = labels;
                temperatureChart.data.datasets[0].data = temperatures;
                temperatureChart.update();

                humidityChart.data.labels = labels;
                humidityChart.data.datasets[0].data = humidities;
                humidityChart.update();

                lightChart.data.labels = labels;
                lightChart.data.datasets[0].data = lightLevels;
                lightChart.update();

                // Obsługa przycisków nawigacyjnych
                document.getElementById("prevBtn").disabled = data.length < limit;
                document.getElementById("nextBtn").disabled = offset === 0;
            }
        })
        .catch(error => {

```

```

        // Obsługa błędów w przypadku problemu z pobraniem danych
        console.error("Błąd podczas pobierania danych:", error);
    });
}

// Załaduj dane na start po załadowaniu strony
window.addEventListener("load", () => {
    fetchChartData(currentOffset); // Pobierz dane dla wykresów
});

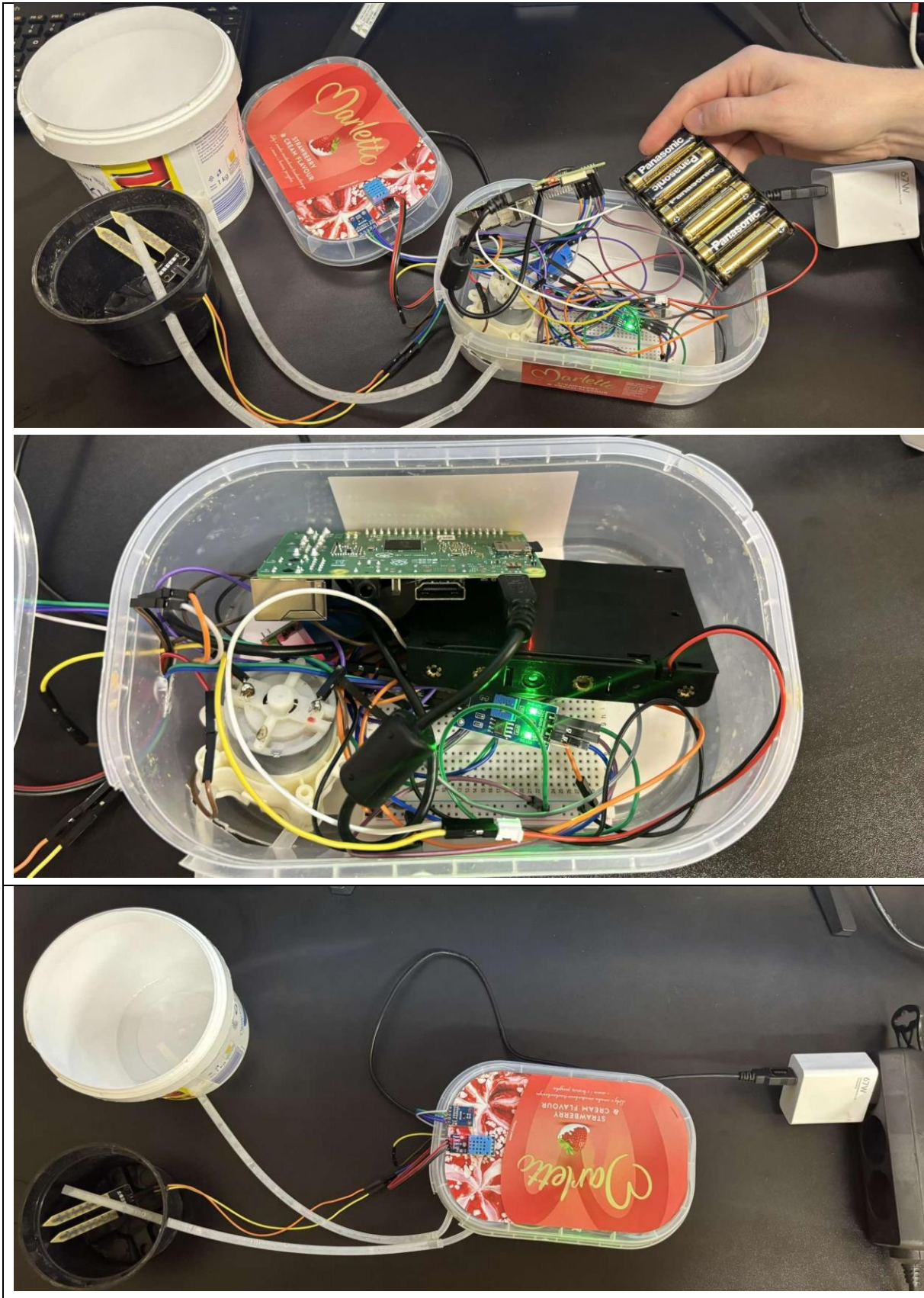
// Obsługa przycisku "Poprzednie"
document.getElementById("prevBtn").addEventListener("click", () => {
    currentOffset += limit; // Zwiększamy offset (przesuwamy do starszych danych)
    fetchChartData(currentOffset); // Pobieramy dane dla starszych punktów
});

// Obsługa przycisku "Następne"
document.getElementById("nextBtn").addEventListener("click", () => {
    if (currentOffset >= limit) {
        currentOffset -= limit; // Zmniejszamy offset (wracamy do nowszych danych)
        fetchChartData(currentOffset); // Pobieramy dane dla nowszych punktów
    }
});

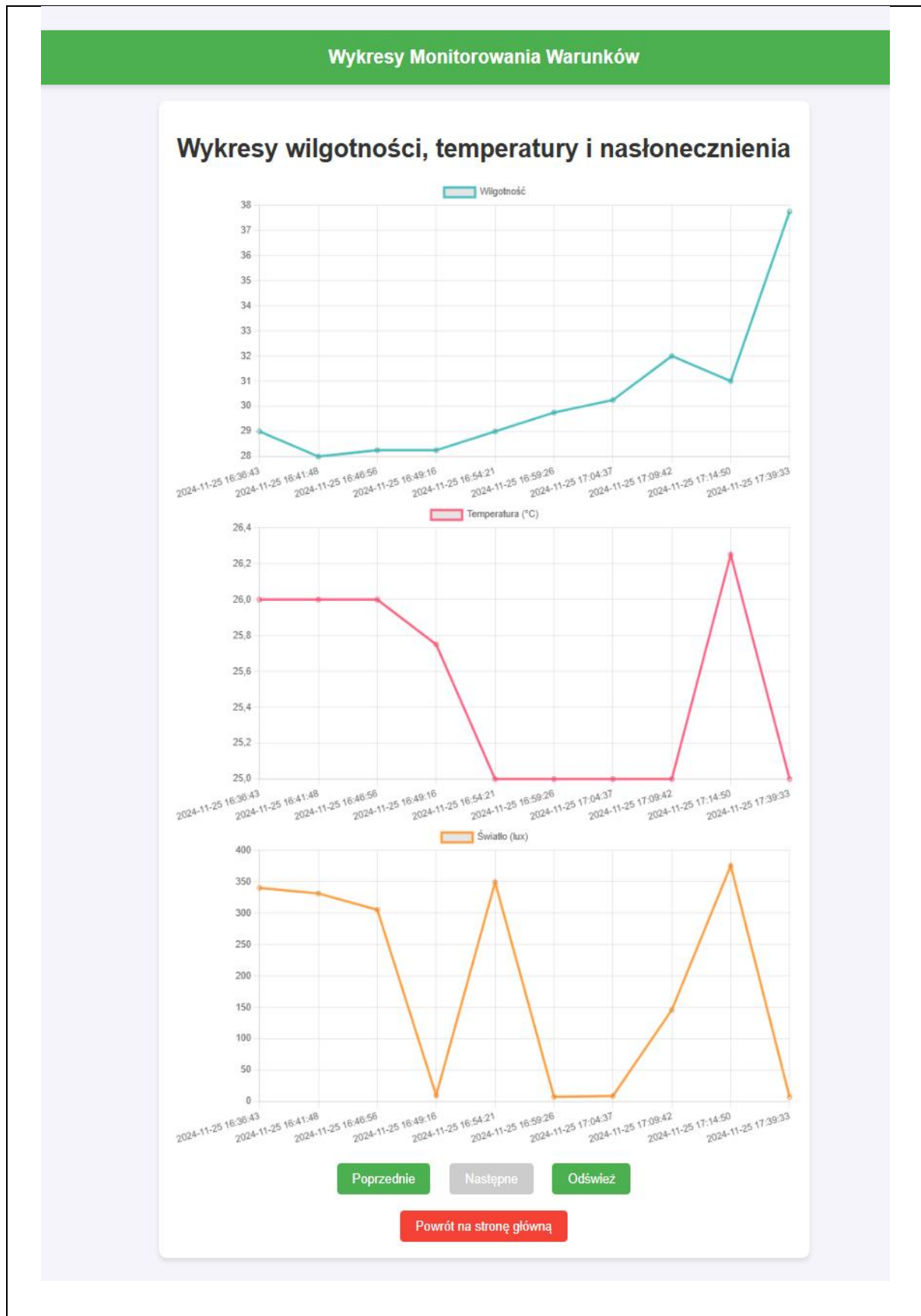
// Obsługa przycisku "Odśwież"
document.getElementById("refreshBtn").addEventListener("click", () => {
    location.reload(); // Odświeżamy stronę (przywracamy dane do początkowego stanu)
});

```


- Zdjęcia zrealizowanego układu.



- Zrzuty ekranu aplikacji (jeśli występują).



System Monitorowania Roślin

Aktualne dane oraz możliwość sterowania nawadnianiem roślin

Temperatura

25 °C

Wilgotność

37.75 %

Poziom Światła

7.25 lux

Podlej Rośliny

Zobacz Wykresy

Odśwież Stronę

System Monitorowania Roślin

Aktualne dane oraz możliwość sterowania nawadnianiem roślin

Temperatura

25 °C

Wilgotność

37.75 %

Poziom Światła

7.25 lux

Podlej Rośliny

Zobacz Wykresy

Odśwież Stronę

Podlewanie zakończone pomyślnie!

- **Podsumowanie i wnioski.**

Projekt systemu monitorowania i sterowania podlewaniem roślin z wykorzystaniem Raspberry Pi był bardzo udanym przedsięwzięciem, które pozwoliło na skuteczną automatyzację procesu pielęgnacji roślin. Celem projektu było stworzenie systemu, który za pomocą czujników monitoruje warunki środowiskowe (temperatura, wilgotność powietrza, nasłonecznienie, wilgotność gleby) i automatycznie steruje podlewaniem roślin. Udało się zrealizować wszystkie założenia, a także dodać kilka dodatkowych funkcji.

Z powodzeniem połączono sprzęt z oprogramowaniem, co umożliwiło stworzenie systemu, który działa w sposób płynny i efektywny. Czujniki DHT11, BH1750 oraz czujnik wilgotności gleby zbierały dane o warunkach w pomieszczeniu i przetwarzały je na dane wejściowe do algorytmu sterującego pompą nawadniającą. System automatycznie włączał pompkę w przypadku, gdy wilgotność gleby spadała poniżej ustalonego progu, a także umożliwiał ręczne sterowanie podlewaniem przez stronę internetową.

Dodatkowo, dane zbierane przez system były przechowywane w bazie danych SQLite, co pozwoliło na ich późniejszą analizę oraz wizualizację w postaci wykresów na stronie internetowej. Interfejs webowy, stworzony w frameworku Flask, umożliwiał zdalne monitorowanie aktualnych parametrów środowiskowych oraz sterowanie systemem podlewania, co zwiększało komfort użytkowania.

Wszystkie te elementy działały bez zarzutu, zapewniając stabilną pracę systemu. Kluczowym osiągnięciem była integracja różnych komponentów sprzętowych z oprogramowaniem, co pozwoliło na stworzenie funkcjonalnego systemu automatycznego podlewania roślin, który odpowiadał na zmieniające się warunki w czasie rzeczywistym.

Projekt był również dobrą okazją do nauki o integracji różnych technologii, takich jak Raspberry Pi, czujniki, bazy danych oraz aplikacje webowe. Realizacja pozwoliła na zdobycie doświadczeń w tworzeniu systemów automatycznych, które mogą być wykorzystywane w różnych dziedzinach, takich jak rolnictwo, ogrodnictwo czy automatyzacja domowa.