1. **Cel i zakres projektu.**

Celem projektu jest opracowanie kompleksowego systemu monitorowania parametrów fizjologicznych, takich jak tętno (HR), saturacja tlenu we krwi (SpO_2) oraz sygnał elektrokardiograficzny (EKG), w czasie rzeczywistym. System ten ma zastosowanie w medycynie oraz w obszarach związanych ze sportem i zdrowym stylem życia, umożliwiając śledzenie stanu zdrowia użytkownika.

Sprzętowy zakres projektu:

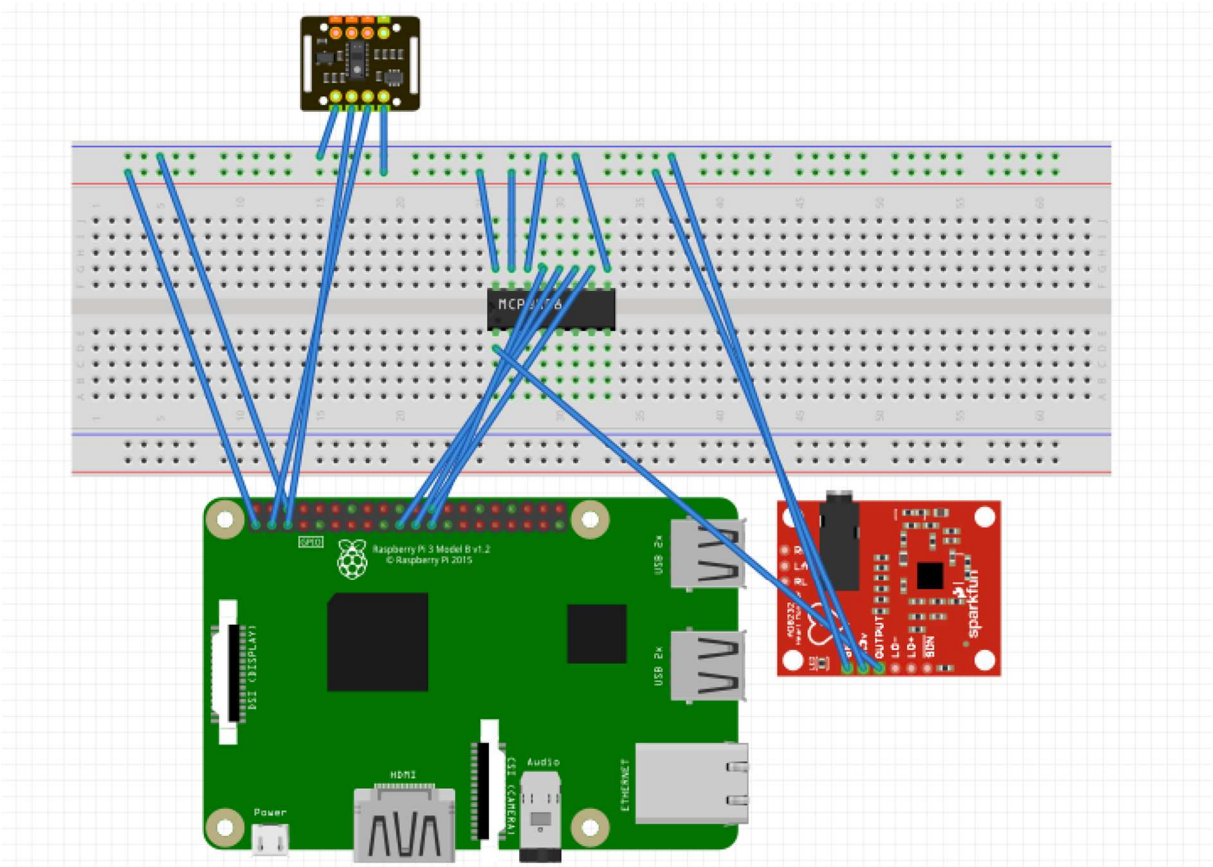
1. **Czujnik MAX30102** - czujnik optyczny wykorzystywany do pomiaru tętna i saturacji krwi, wykorzystujący światło czerwone i podczerwone. Komunikacja z czujnikiem odbywa się za pomocą magistrali I²C, co umożliwia integrację z Raspberry Pi.
2. **Czujnik AD8232** - pozwala na rejestrację sygnału EKG, umożliwiając analizę aktywności elektrycznej serca. Czujnik jest podłączony do Raspberry Pi poprzez konwerter analogowo-cyfrowy MCP3008, umożliwiający przekształcenie sygnału analogowego na cyfrowy.
3. **Jednostka sterująca – Raspberry Pi 3B+** pełni rolę głównej jednostki sterującej systemem, obsługując zarówno czujnik MAX30102, jak i AD8232. Raspberry Pi przetwarza dane pomiarowe i udostępnia je użytkownikowi poprzez przeglądarkę internetową.
4. **Zasilanie i peryferia** - cały system zasilany jest poprzez port USB Raspberry Pi. Raspberry Pi obsługuje komunikację z czujnikami oraz wizualizację wyników pomiarów.
5. **Interfejs użytkownika** - wyniki pomiarów są wyświetlane w przeglądarce internetowej użytkownika za pomocą serwera Flask. Interfejs graficzny prezentuje dane w czasie rzeczywistym na dynamicznych wykresach, uwzględniając tętno, saturację oraz sygnał EKG.

Programistyczny zakres projektu:

1. **Oprogramowanie czujnika MAX30102** - implementacja komunikacji z czujnikiem przez magistralę I²C. Pobieranie danych z rejestrów FIFO czujnika (sygnały czerwone i podczerwone). Algorytmy analizy sygnałów w celu obliczania tętna oraz saturacji tlenu.
2. **Oprogramowanie czujnika AD8232** - odczyt sygnału EKG za pomocą konwertera MCP3008. Implementacja cyfrowych filtrów pasmowych w celu usunięcia zakłóceń i wyodrębnienia charakterystycznych szczytów sygnału EKG. Analiza sygnału w celu wykrycia załamków P, QRS i T.
3. **Przetwarzanie sygnałów** - filtracja sygnałów z czujników (MAX30102 i AD8232) w celu usunięcia zakłóceń. Wykorzystanie algorytmów detekcji szczytów do analizy tętna oraz sygnału EKG.
4. **Interfejs użytkownika** - wizualizacja danych na dynamicznych wykresach w przeglądarce internetowej. Strona internetowa wyświetla wykresy tętna, saturacji oraz sygnału EKG w czasie rzeczywistym.
5. **Testowanie i optymalizacja** - system został przetestowany pod kątem dokładności pomiarów oraz odporności na zakłócenia (np. błędy odczytów spowodowane ruchem użytkownika). Optymalizacja algorytmów przetwarzania sygnałów w celu uzyskania płynnych i dokładnych wyników.

2. Schemat.

Schemat połączeniowy:



3. Założenia projektowe a ich realizacja.

Założenia projektowe:

1. **Pomiar parametrów życiowych -** umożliwienie rejestracji tętna (HR) i saturacji krwi (SpO_2) w czasie rzeczywistym za pomocą czujnika MAX30102. Wykorzystanie czujnika AD8232 do rejestracji sygnału EKG w celu analizy aktywności elektrycznej serca.
2. **Przetwarzanie sygnałów w czasie rzeczywistym -** implementacja algorytmów filtracji sygnałów w celu eliminacji zakłóceń i wyodrębnienia kluczowych informacji, takich jak szczyty sygnałów (dla HR i EKG). Wyświetlanie przetworzonych danych w czasie rzeczywistym na dynamicznych wykresach.
3. **Interfejs użytkownika -** zapewnienie prostego i czytelnego interfejsu graficznego w przeglądarce internetowej, umożliwiającego wizualizację wyników pomiarów (HR, SpO_2 i EKG).

Zrealizowane cele:

- Udało się w pełni zaimplementować pomiar tętna (HR) oraz saturacji tlenu (SpO_2) za pomocą czujnika MAX30102. Dane są przetwarzane i wyświetlane w czasie rzeczywistym.
- Stworzono dynamiczny interfejs graficzny oparty na serwerze Flask, prezentujący wyniki pomiarów na dynamicznych wykresach w przeglądarce internetowej.
- Udało się wdrożyć funkcję pomiaru sygnału EKG za pomocą czujnika AD8232. Sygnał jest filtrowany, a jego szczyty są wykrywane i wizualizowane.
- System działa stabilnie i pozwala na rejestrację danych w czasie rzeczywistym, przy jednoczesnym zachowaniu mobilności.

Niezrealizowane cele:

- **Precyzyjne usuwanie zakłóceń w sygnale EKG**: Pomimo zastosowania filtrów cyfrowych, sygnał EKG nadal jest podatny na zakłócenia wynikające z ruchu użytkownika oraz niewystarczającego ekranowania urządzenia.
- **Dokładna kalibracja czujnika MAX30102**: Wartości saturacji tlenu mogą być obarczone błędem, co wynika z zależności działania czujnika od warunków zewnętrznych (np. jasności otoczenia, temperatury).

Możliwości rozwoju projektu:

1. **Poprawa jakości sygnału** - dodanie ekranowania obwodów w celu eliminacji zakłóceń elektromagnetycznych. Wprowadzenie zaawansowanych algorytmów filtracji sygnałów, takich jak filtry adaptacyjne.
2. **Rozbudowa funkcjonalności** - implementacja algorytmów wykrywających załamki P, QRS i T w sygnale EKG w celu szczegółowej analizy pracy serca. Dodanie funkcji alarmów informujących o anomaliach, np. zbyt wysokim tętnie.
3. **Długoterminowa rejestracja danych** - wdrożenie mechanizmu rejestracji i przechowywania danych pomiarowych w chmurze w celu analizy długoterminowej.
4. **Integracja z urządzeniami mobilnymi** - rozbudowa interfejsu użytkownika o aplikację mobilną, pozwalającą na przeglądanie wyników pomiarów oraz zarządzanie urządzeniem.

4. Listing kodu.

```
# Funkcje pomocnicze
def write_register(register, value):
    """
    Zapisuje wartość do zadanego rejestru w czujniku MAX30102.
    """
    bus.write_byte_data(I2C_ADDRESS, register, value)

def read_fifo():
    """
    Odczytuje dane z FIFO czujnika MAX30102.
    Zwraca wartości dla diody czerwononapromieniowanej (red) i
podczerwieni (ir).
    """
    try:
        data = bus.read_i2c_block_data(I2C_ADDRESS, REG_FIFO_DATA, 6)
        red = (data[0] << 16 | data[1] << 8 | data[2]) & 0x03FFFF
        ir = (data[3] << 16 | data[4] << 8 | data[5]) & 0x03FFFF
        return red, ir
    except OSError:
        return None, None

def moving_average(data, window_size=5):
    """
    Oblicza średnią ruchomą dla danych.
    """
    return np.convolve(data, np.ones(window_size) / window_size,
mode='valid')

def calculate_hr_and_spo2(ir_data, red_data, fs=10):
    """
    Oblicza tętno (HR) oraz saturację tlenu (SpO2) na podstawie danych z
czujnika.
    - `ir_data`: dane podczerwone
    - `red_data`: dane czerwononapromieniowane
    - `fs`: częstotliwość próbkowania (Hz)
    """
    ir_data_filtered = moving_average(ir_data)
    red_data_filtered = moving_average(red_data)

    # Detekcja szczytów w danych podczerwonych
    ir_peaks, _ = find_peaks(ir_data_filtered, distance=fs // 2)
    rr_intervals = np.diff(ir_peaks) / fs
    hr = 60 / np.mean(rr_intervals) if len(rr_intervals) > 0 else 0  #
Obliczenie HR

    # Obliczenie stosunku średnich wartości sygnałów
    ratio = np.mean(red_data_filtered) / np.mean(ir_data_filtered)
    spo2 = 110 - (18 * ratio)  # Przybliżona wartość SpO2
    spo2 = max(90, min(100, spo2))  # Ograniczenie wartości SpO2 do
przedziału [90, 100]
    return hr, spo2


def initialize_sensor():
    """
    Inicjalizuje czujnik MAX30102, konfigurując odpowiednie rejestry.
```

```
    """
    write_register(REG_INTR_ENABLE_1, 0xC0)  # Włączanie przerwań
    write_register(REG_MODE_CONFIG, 0x03)  # Ustawienie trybu pracy
    write_register(REG_SPO2_CONFIG, 0x27)  # Konfiguracja SpO2
    write_register(REG_LED1_PA, 0x3F)  # Maksymalna moc LED1
    write_register(REG_LED2_PA, 0x3F)  # Maksymalna moc LED2

# Główna funkcja zbierająca dane
def data_collector():
    """
    Funkcja w tle zbierająca dane z czujnika w czasie rzeczywistym.
    Dane są zapisywane w globalnym buforze `data_buffer`.
    """
    initialize_sensor()
    start_time = time.time()
    ir_data = []
    red_data = []

    while True:
        red, ir = read_fifo()
        if red and ir:
            ir_data.append(ir)
            red_data.append(red)

            # Aktualizacja danych co 50 próbek
            if len(ir_data) % 50 == 0 and len(ir_data) > 50:
                hr, spo2 = calculate_hr_and_spo2(ir_data, red_data)
                current_time = time.time() - start_time

                # Dodanie danych do bufora
                data_buffer["time"].append(current_time)
                data_buffer["hr"].append(hr)
                data_buffer["spo2"].append(spo2)

                # Utrzymanie ostatnich 100 próbek w buforze
                if len(data_buffer["time"]) > 100:
                    for key in data_buffer:
                        data_buffer[key].pop(0)

            time.sleep(0.1)

# Serwer Flask
app = Flask(__name__)

@app.route("/")
def index():
    """
    Strona główna serwera, wyświetlająca wykresy HR i SpO2.
    """
    return render_template_string("""
    <!DOCTYPE html>
    <html>
    <head>
        <title>Wykresy HR i SpO2</title>
        <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
        <style>
            body {
                font-family: Arial, sans-serif;
                background-color: #f8f9fa;
```

```
                margin: 0;
                padding: 0;
                display: flex;
                flex-direction: column;
                align-items: center;
            }
            h1 {
                color: #343a40;
                text-align: center;
                margin-top: 5px;
                font-size: 2.5rem;
                font-weight: bold;
            }
            .chart-container {
                width: 80%;
                margin: 10px auto;
                padding: 5px;
                background-color: #ffffff;
                border-radius: 8px;
                box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            }
        </style>
    </head>
    <body>
        <h1>Wykres HR (tętno) i SpO2 (saturacja) w czasie
rzeczywistym</h1>

        <div class="chart-container">
            <div id="hr_chart" style="width: 100%; height: 350px;"></div>
        </div>

        <div class="chart-container">
            <div id="spo2_chart" style="width: 100%; height:
350px;"></div>
        </div>

        <script>
            function fetchData() {
                fetch('/data').then(response =>
response.json()).then(data => {
                    var time = data.time;

                    // Wykres HR
                    var hr_trace = {
                        x: time,
                        y: data.hr,
                        mode: 'lines',
                        name: 'HR (BPM)',
                        line: { color: 'red', width: 2 }
                    };
                    Plotly.newPlot('hr_chart', [hr_trace], {
                        title: 'Tętno (HR)',
                        xaxis: { title: 'Czas (s)' },
                        yaxis: { title: 'HR (BPM)' },
                        paper_bgcolor: '#f8f9fa',
                        plot_bgcolor: '#ffffff'
                    });

                    // Wykres SpO2
```

```
                    var spo2_trace = {
                        x: time,
                        y: data.spo2,
                        mode: 'lines',
                        name: 'SpO2 (%)',
                        line: { color: 'blue', width: 2 }
                    };
                    Plotly.newPlot('spo2_chart', [spo2_trace], {
                        title: 'Saturacja tlenu (SpO2)',
                        xaxis: { title: 'Czas (s)' },
                        yaxis: { title: 'SpO2 (%)' },
                        paper_bgcolor: '#f8f9fa',
                        plot_bgcolor: '#ffffff'
                    });
                });
            }
            setInterval(fetchData, 1000);
        </script>
    </body>
    </html>
    """)

@app.route("/data")
def data():
    """
    Endpoint zwracający dane w formacie JSON.
    """
    return Response(json.dumps(data_buffer), mimetype="application/json")

if __name__ == "__main__":
    # Uruchomienie zbierania danych w osobnym wątku
    threading.Thread(target=data_collector, daemon=True).start()
    # Uruchomienie serwera Flask
    app.run(host="0.0.0.0", port=5000, debug=True)
```

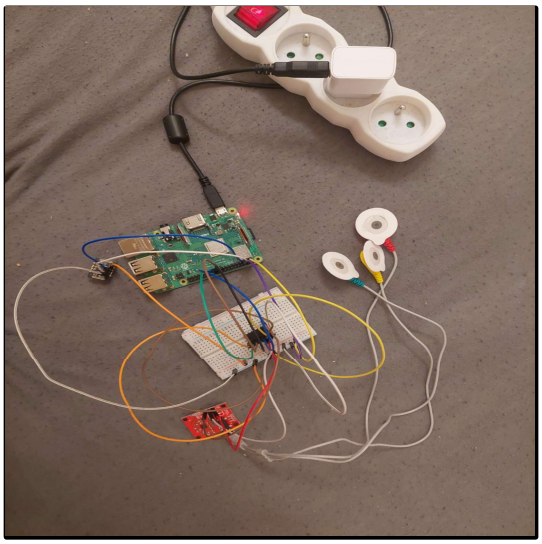
```
def read_channel(channel):
    """
    Odczytuje wartość z zadanego kanału MCP3008.
    - `channel`: numer kanału (0-7)
    """
    adc = spi.xfer2([1, (8 + channel) << 4, 0])
    return ((adc[1] & 3) << 8) + adc[2]

# Funkcja przekształcająca wartość ADC na napięcie
def convert_to_voltage(data, vref=3.3):
    """
    Konwertuje wartość ADC na napięcie.
    - `data`: wartość ADC (0-1023)
    - `vref`: napięcie odniesienia (domyślnie 3.3V)
    """
    return round((data * vref) / 1023.0, 2)

# Funkcja do zbierania danych EKG
def record_ecg(duration=10, sampling_rate=250):
    """
    Zbiera dane EKG przez określony czas.
    - `duration`: czas rejestracji w sekundach
    - `sampling_rate`: częstotliwość próbkowania w Hz
```

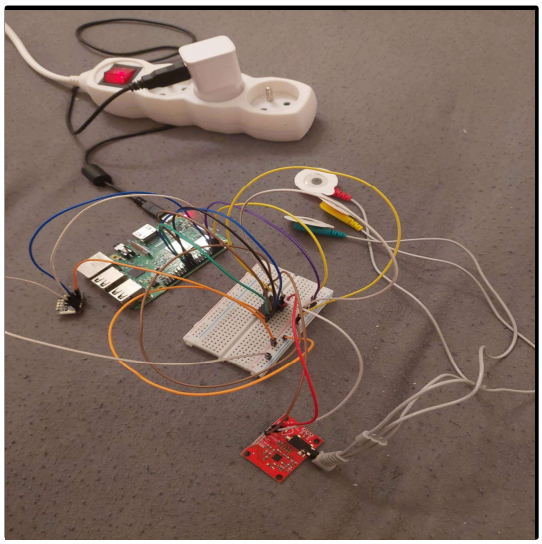
```
    """
    print("Rejestrowanie sygnału EKG...")
    values = []
    times = []
    start_time = time.time()

    try:
        while time.time() - start_time < duration:
            raw_value = read_channel(0)  # Odczyt z kanału 0
            voltage = convert_to_voltage(raw_value)  # Konwersja na
napięcie
            current_time = time.time() - start_time

            values.append(voltage)
            times.append(current_time)
            time.sleep(1 / sampling_rate)  # Oczekiwanie zgodne z
częstotliwością próbkowania

    except KeyboardInterrupt:
        print("Zatrzymano rejestrowanie.")

    print("Rejestrowanie zakończone.")
    return times, values

# Funkcja filtru Butterwortha
def butter_lowpass_filter(data, cutoff=20.0, fs=250, order=4):
    """
    Tworzy i stosuje filtr dolnoprzepustowy Butterwortha.
    - `data`: sygnał wejściowy
    - `cutoff`: częstotliwość odcięcia w Hz
    - `fs`: częstotliwość próbkowania w Hz
    - `order`: rząd filtra
    """
    nyquist = 0.5 * fs  # Częstotliwość Nyquista
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return filtfilt(b, a, data)

# Funkcja do usuwania szumów za pomocą filtru medianowego
def median_filter(data, kernel_size=3):
    """
    Stosuje filtr medianowy do danych.
    - `data`: sygnał wejściowy
    - `kernel_size`: rozmiar okna filtru
    """
    return medfilt(data, kernel_size=kernel_size)

# Funkcja zbierająca dane EKG i aktualizująca bufor danych
def data_collector():
    """
    Zbiera dane EKG w czasie rzeczywistym i aktualizuje bufor danych.
    """
    start_time = time.time()
    values = []
    times = []

    while True:
        raw_value = read_channel(0)  # Odczyt z MCP3008
        voltage = convert_to_voltage(raw_value)  # Konwersja na napięcie
```

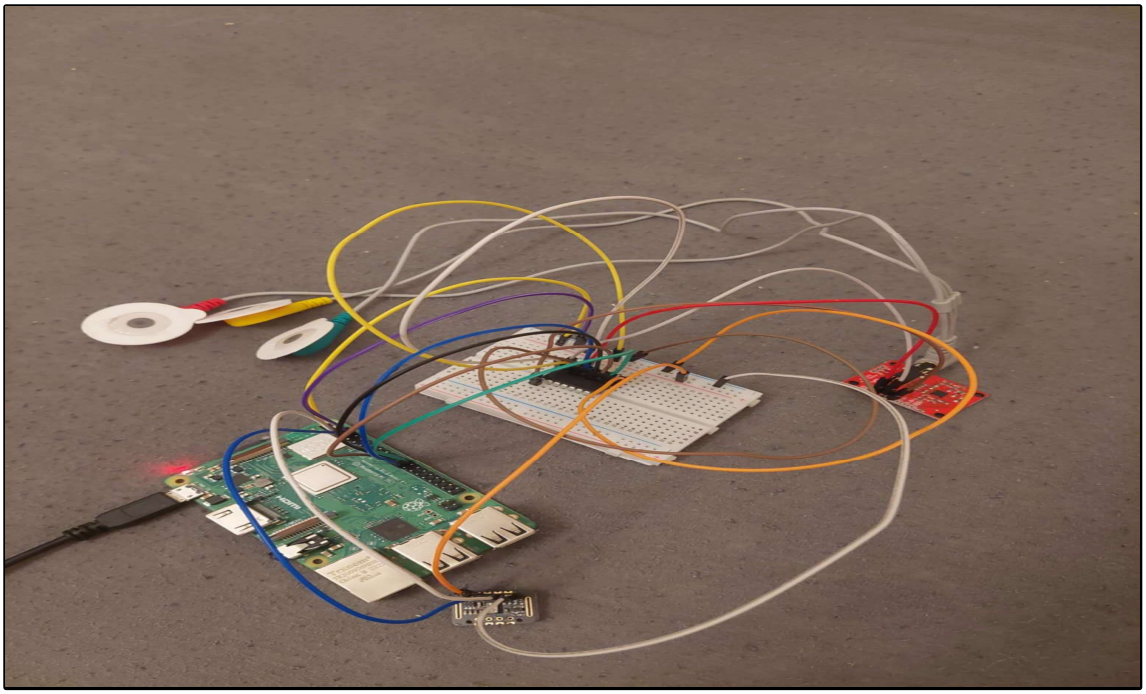
```
        current_time = time.time() - start_time

        values.append(voltage)
        times.append(current_time)

        # Filtrowanie sygnału po zgromadzeniu minimum 50 próbek
        if len(values) > 50:
            filtered_values = butter_lowpass_filter(values, cutoff=20.0,
fs=250, order=4)
            filtered_values = median_filter(filtered_values,
kernel_size=3)

            # Aktualizacja globalnego bufora z ostatnimi 300 punktami
            data_buffer["time"] = times[-300:]
            data_buffer["ekg"] = filtered_values[-300:]

        time.sleep(0.002)  # Wysoka częstotliwość próbkowania (ok. 500
Hz)

# Serwer Flask
app = Flask(__name__)

@app.route("/")
def index():
    """
    Strona główna serwera, wyświetlająca wykres EKG w czasie
rzeczywistym.
    """
    return render_template_string("""
    <!DOCTYPE html>
    <html>
    <head>
        <title>Wykres EKG w czasie rzeczywistym</title>
        <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
        <style>
            body {
                font-family: Arial, sans-serif;
                background-color: #f8f9fa;
                margin: 0;
                padding: 0;
                display: flex;
                flex-direction: column;
                align-items: center;
            }
            h1 {
                color: #343a40;
                text-align: center;
                margin-top: 20px;
                font-size: 2.5rem;
                font-weight: bold;
            }
            .chart-container {
                width: 90%;
                margin: 20px auto;
                padding: 15px;
                background-color: #ffffff;
                border-radius: 8px;
                box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            }
```

```
        </style>
    </head>
    <body>
        <h1>Wykres EKG w czasie rzeczywistym</h1>

        <div class="chart-container">
            <div id="ekg_chart" style="width: 100%; height:
600px;"></div>
        </div>

        <script>
            function fetchData() {
                // Pobieranie danych z serwera
                fetch('/data').then(response =>
response.json()).then(data => {
                    var time = data.time;
                    var ekg = data.ekg;

                    var ekg_trace = {
                        x: time,
                        y: ekg,
                        mode: 'lines',
                        name: 'EKG',
                        line: { color: 'blue', width: 2 }
                    };

                    // Rysowanie wykresu za pomocą Plotly
                    Plotly.newPlot('ekg_chart', [ekg_trace], {
                        title: 'Sygnał EKG',
                        xaxis: { title: 'Czas (s)' },
                        yaxis: { title: 'Napięcie (V)' },
                        paper_bgcolor: '#f8f9fa',
                        plot_bgcolor: '#ffffff'
                    });
                });
            }
            setInterval(fetchData, 50);  // Odświeżanie co 50 ms
        </script>
    </body>
    </html>
    """)

@app.route("/data")
def data():
    """
    Endpoint zwracający dane w formacie JSON.
    """
    # Konwersja na typy kompatybilne z JSON
    data_json = {
        "time": [float(t) for t in data_buffer["time"]],
        "ekg": [float(e) for e in data_buffer["ekg"]]
    }
    return Response(json.dumps(data_json), mimetype="application/json")

if __name__ == "__main__":
    # Uruchomienie zbierania danych w osobnym wątku
    threading.Thread(target=data_collector, daemon=True).start()
    # Uruchomienie serwera Flask
    app.run(host="0.0.0.0", port=5000, debug=True)
```

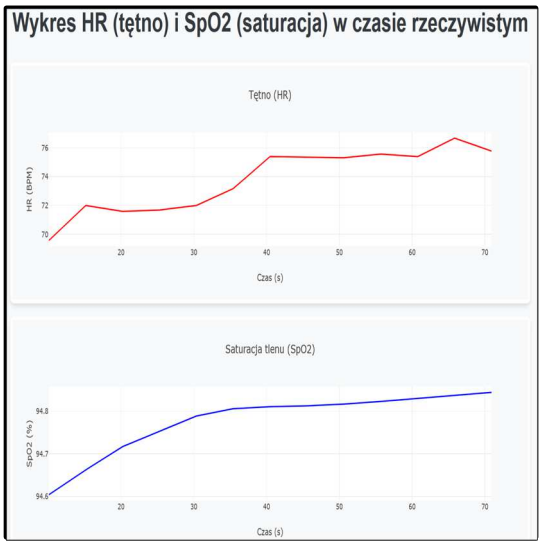
5. Zdjęcia zrealizowanego układu.

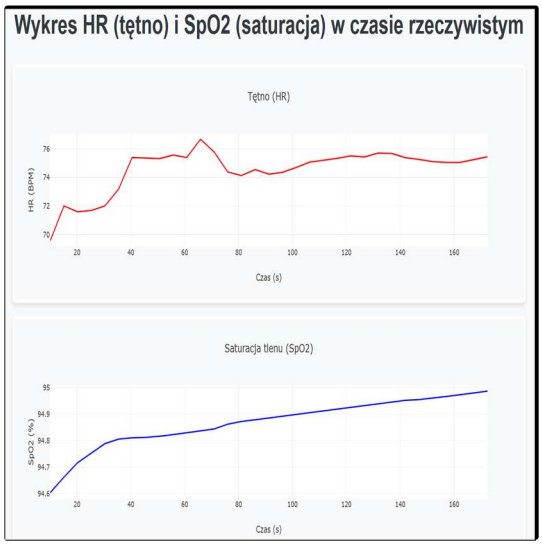






6. Zrzuty ekranu aplikacji (jeśli występują).





7. Podsumowanie i wnioski.

Projekt realizowany na platformie Raspberry Pi z wykorzystaniem czujników MAX30102 oraz AD8232 miał na celu stworzenie systemu monitorującego podstawowe parametry życiowe, takie jak tętno, saturacja tlenu oraz zapis sygnału EKG w czasie rzeczywistym. Był to ambitny projekt, łączący aspekty sprzętowe, takie jak integracja czujników z Raspberry Pi, z zaawansowanym oprogramowaniem do przetwarzania danych i ich wizualizacji.

Realizacja projektu zakończyła się sukcesem pod wieloma względami. Udało się poprawnie skonfigurować czujniki MAX30102 i AD8232, zapewniając stabilny odczyt sygnałów, oraz opracować algorytmy umożliwiające przetwarzanie danych, w tym filtrowanie i analizę sygnałów. Czujnik MAX30102 został skonfigurowany do rejestrowania sygnałów RED i IR, co pozwoliło na wyznaczenie tętna oraz saturacji tlenu. Czujnik AD8232 umożliwił zapis sygnału EKG, który następnie był filtrowany i prezentowany w formie wykresu. Kluczowym elementem było stworzenie serwera Flask, który w czasie rzeczywistym prezentował wyniki na interaktywnych wykresach. Interfejs użytkownika został zaprojektowany w sposób czytelny i estetyczny, co umożliwiło łatwe monitorowanie parametrów.

Podczas realizacji napotkano jednak pewne trudności. Szczególnie wymagającym aspektem było uzyskanie stabilnych odczytów sygnału EKG. Zakłócenia wynikające z niedokładnego kontaktu elektrod z ciałem oraz szumy mechaniczne i elektryczne były wyzwaniem. Mimo zastosowania zaawansowanych metod filtrowania, w niektórych przypadkach zakłócenia wpływały na jakość sygnału. Również konfiguracja rejestrów czujnika MAX30102 wymagała szczególnej uwagi i dokładności, co początkowo spowalniało pracę.

Podsumowując, projekt osiągnął założone cele i stworzył solidną podstawę do dalszego rozwoju. System pozwala na skuteczne monitorowanie tętna, saturacji tlenu i sygnału EKG, a także ich wizualizację w czasie rzeczywistym. Dzięki realizacji tego projektu zdobyliśmy cenne doświadczenie w pracy z zaawansowanymi czujnikami biomedycznymi oraz w tworzeniu systemów wbudowanych, co może stanowić bazę dla bardziej zaawansowanych aplikacji w dziedzinie inżynierii biomedycznej.