

Sidly Care EKG

Raport końcowy

z realizacji zadania pt.

**Opracowanie prototypu systemu telemedycznego
do stałego monitoringu pracy**

SPIS TREŚCI

PLATFORMA IT	4
Sieć klasyfikująca	4
Wstęp	4
Architektura	4
Splotowa sieć neuronowa	5
Implementacja sztucznej sieci neuronowej	6
Przygotowanie środowiska uruchomieniowego	7
Trenowanie sieci	8
Internetowa aplikacja modułu analizującego	10
Zapis wejściowych sygnałów EKG	12
Decymacja i filtracja	12
Obliczanie tętna	14
Klasyfikacja sygnałów EKG	15
Wczytanie modelu sieci neuronowej	15
Uruchomienie	16
Badania testowe	16
Załączniki	16
Załącznik nr 1 – skrypt network.py tworzący splotową sztuczną sieć neuronową.	16
Załącznik nr 2 - skrypt train.py uruchamiający proces trenowania sieci.	19
Załącznik nr 3 - skrypt predict.py uruchamiający proces predykcji, czyli przyporządkowania sygnału EKG do jednej z czterech klas.	21
Załącznik nr 4 - skrypt load.py skrypt do wczytywania i wstępnego przetworzenia danych.	21
Załącznik nr 5 - skrypt util.py – skrypt do serializacji i deserializacji danych.	23
Załącznik nr 6 - skrypt main.py – skrypt realizujący główną funkcjonalność modułu analizującego.	23
Aplikacja mobilna	27
Framework Flutter	27
Wybrane biblioteki	28
Biblioteki wspierające łączenie po Bluetooth Low Energy (BLE)	30
Biblioteka flutter_blue 0.8.0	30

Biblioteka flutter_reactive_ble	31
Biblioteka flutter_bluetooth_serial	32
Biblioteki do rysowania dynamicznych wykresów	33
Biblioteka Oscilloscope	33
Biblioteka fl_chart	35
Biblioteka fl_animated_linechart	36
Interfejs użytkownika - UX	37
Ekran logowania	37
Ekran menu	38
Ekran skanowania urządzeń	39
Ekran pomiarowy	40
Wykonywane obliczenia	41
Badania testowe	45
R Pacjent nr 1 (JUL)	45
Pacjent nr 2 (RM)	46
Pacjent nr 3 (KG)	47
Pacjent nr 4 (AT)	47
Pacjent nr 5 (WJO)	48
Aplikacja internetowa	50
OPASKA POMIAROWA	56
Architektura rozwiązania	56
2.2.1. Wstęp	56
2.2.2. Architektura	56
2.2.3. Toolchain	58
2.2.4. Oprogramowanie modułu EKG	60
2.2.5. Protokół komunikacyjny	63
2.2.6. Warstwa komunikacyjna	65

1. PLATFORMA IT

1.1. Sieć klasyfikująca

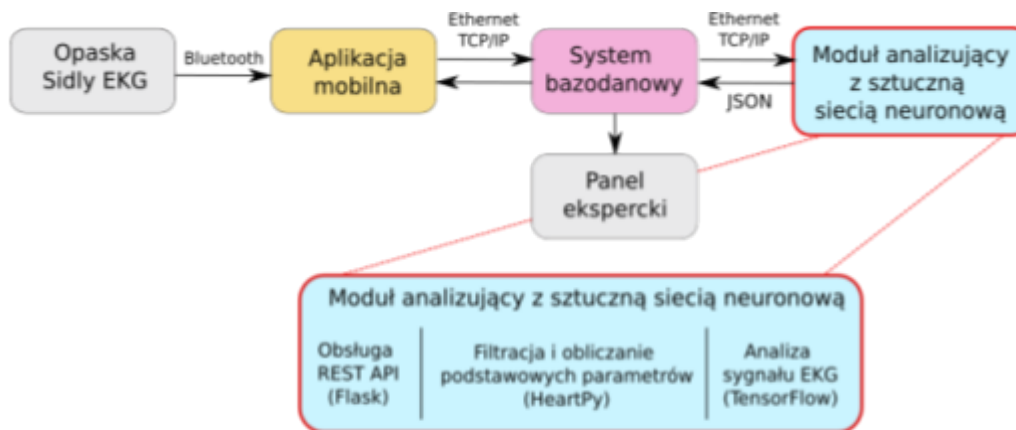
W niniejszy podrozdział zawiera opis etapów realizacji modułu analizującego bazującego na sztucznej sieci neuronowej.

1.1.1. Wstęp

Celem niniejszego zadania było opracowanie modułu analizującego bazującego na sztucznej sieci neuronowej przeznaczonego do detekcji nieprawidłowości pracy serca. Przed opracowaniem modułu dokonano analizy literatury. Na jej podstawie można spotkać, że istnieje wiele lepiej lub gorzej działających rozwiązań, jednak na uznanie zasługuje rozwiązanie przedstawione przez A. Y. Hannun i in. z 2019r. W swojej pracy A. Y. Hannun i in. przedstawiają splotową sieć neuronową, która dokonuje 12 zbiorowej klasyfikacji sygnałów EKG (m.in. sygnał może być przypisany do kategorii szumu, bigeminii lub trigeminii). Z uwagi na duże możliwości zaprezentowanego w pracy rozwiązania i dużą liczbę cytowań pracy (>500, 1/2021), zdecydowano się na wdrożenie takiej architektury sieci w projekcie Sidly Care EKG.

1.1.2. Architektura

Sztuczna sieć neuronowa będzie częścią modułu analizującego projektu Sidly Care EKG, którego architektura została pokazana na Rys. 1.



Rys. 1: Architektura systemu Sidly Care EKG.

Zadaniem modułu analizującego jest odebranie spróbkowanego sygnału EKG z systemu bazodanowego, jego filtrację oraz analizę pod kątem wystąpienia w sygnale nieprawidłowości związanych z arytmia lub innymi zaburzeniami pracy serca. Zakłada się, że moduł analizujący będzie uruchomiony na maszynie wirtualnej z system operacyjnym Ubuntu 20. Dane do modułu będą wysyłane za pomocą protokołu HTTP i metody POST. Dane będą formatowane zgodnie z formatem JSON.

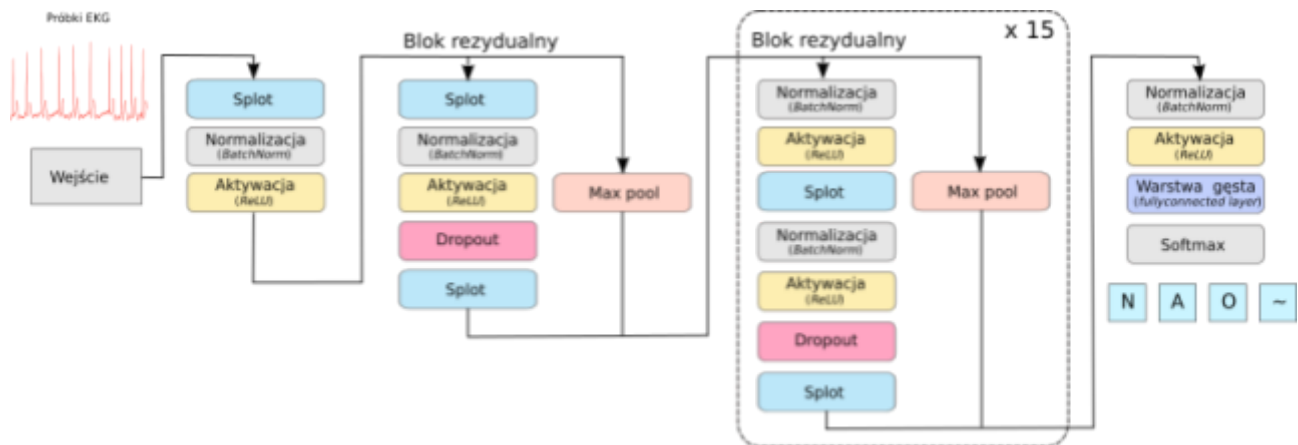
1.1.3.

Splotowa sieć neuronowa

W ramach zadania została opracowana splotowa głęboka sieć neuronowa, która pobiera dane wejściowe jako surowe próbki sygnału EKG (próbkowanie z częstotliwością 200Hz) i następnie generuje jedną prognozę co 256 próbek (co 1,28 s). Sieć wymaga podania na wejście tylko nieprzetworzonych próbek sygnału EKG, bez żadnych innych informacji związanych z pacjentem lub z sygnałem EKG. Sieć składa się z 34 warstw z połączeniami rezydualnymi, które są rozdzielone na 16 bloków rezydualnych. W każdym bloku znajdują się dwie warstwy splotowe. Warstwy splotowe mają $32 \cdot 2^k$ filtrów o szerokości 16, gdzie k jest hiper-parametrem zwiększanym o 1 co czwarty blok rezydualny. Każdy blok redukuje sygnał wejściowy 2 razy, a zatem oryginalny sygnał wejściowy jest redukowany 28 razy. Gdy blok rezydualny redukuje wejście, odpowiednie połączenia rezydualne również redukują sygnał z ich wejść za pomocą operacji *max pooling*.

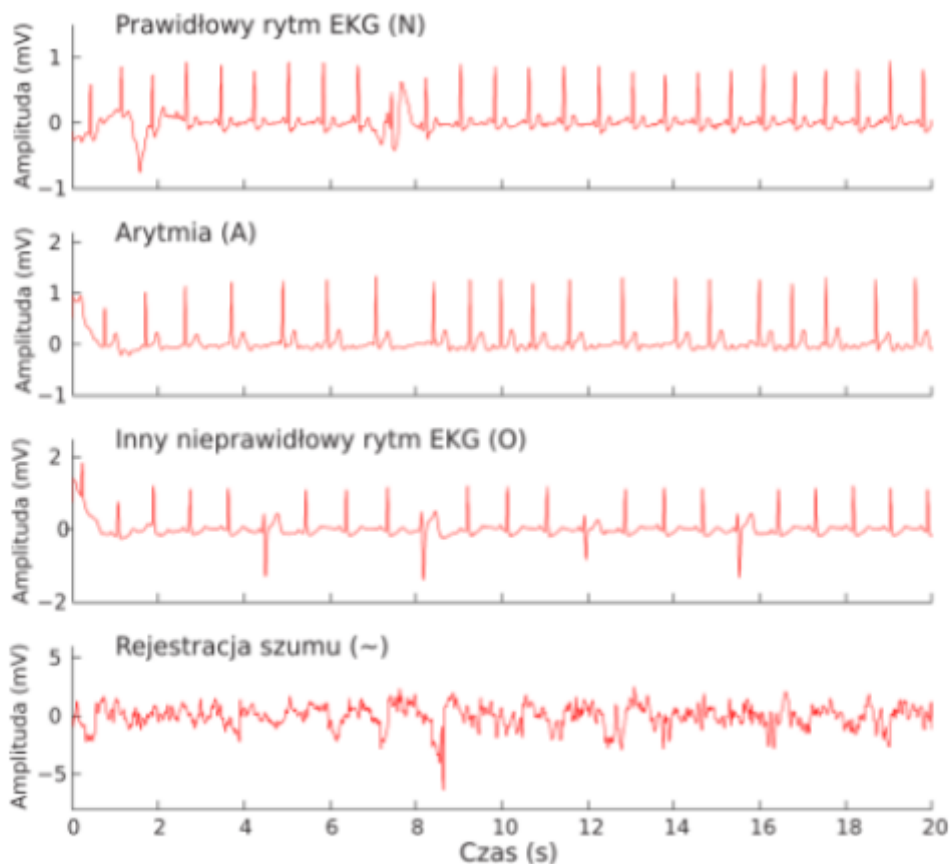
Przed każdą warstwą splotową zastosowana została normalizacja partiami (*batch normalization*) i aktywacja typu ReLU. Należy podkreślić, że pierwsza i ostatnia warstwa sieci mają nieco inną architekturę. Zastosowano również *dropout* o prawdopodobieństwie 20% pomiędzy warstwami splotowymi aby skutecznie przeciwdziałać *overfittingowi*. Na końcu sieci znajduje się w pełni połączona warstwa i warstwa wyjścia typu *softmax*, która co 1.28 s generuje nową predykcję. Predykcja zawiera jedną z 4 możliwych klas sygnału EKG.

Ze względu na zachowanie mobilności urządzenia, w projekcie zakłada się pomiar sygnału EKG podobnego do tego z konfiguracji 1-odprowadzeniowej, natomiast w praktyce lekarskiej zazwyczaj wykonuje się standardowy 12-odprowadzeniowy zapis EKG. 12-odprowadzeniowy sygnał EKG jest wynikiem złożenia 12 różnych krzywych i dzięki temu jego kształt niesie wiele szczegółów, na podstawie, których można diagnozować wiele patologii kardiologicznych. 1-odprowadzeniowy zapis EKG nie jest już tak szczegółowy i zazwyczaj lekarze wykorzystują go tylko do oceny tętna i rytmu pracy serca. Dlatego w projekcie zdecydowano się, że podział typów sygnału EKG na 4 kategorie będzie wystarczający.



Rys. 2. Schemat blokowy zastosowanej splotowej sieci neuronowej.

W oryginalnym rozwiązaniu autorzy zaproponowali 12 klas, jednak w opracowywanym projekcie przyjęto 4 klasy. Są to: klasa prawidłowego sygnału EKG (oznaczenie: N), klasa zaszumionego sygnału (oznaczenie: ~), klasa sygnału z widoczną arytmiami (A). Przykładowe przebiegi sygnałów EKG z przyporządkowanymi klasami zostały przedstawione na Rys. 3.



Rys. 3. Przykładowe przebiegi sygnałów EKG przyporządkowane do odpowiadających im klas.

1.1.3.1. Implementacja sztucznej sieci neuronowej

Do zaimplementowania sztucznej sieci neuronowej zdecydowano użyć się pakietu *TensorFlow* i języka *Python* w wersji 3.8. Do utworzenia sieci, wczytania danych uczących i testujących i predykcji utworzono następujące skrypty:

network.py – skrypt tworzący sztuczną splotową sieć neuronową,

train.py – skrypt uruchamiający proces trenowania sieci,

predic.py – skrypt do uruchomienia predykcji,

load.py – skrypt do wczytywania i wstępnego przetworzenia danych,

util.py – skrypt do serializacji i deserializacji danych.

Pełne kody źródłowe skryptów zamieszczono w podpunkcie załączniki.

1.1.3.2. Przygotowanie środowiska uruchomieniowego

Podczas opracowywania programu modułu analizującego Sidly Care EKG skorzystano z wielu zewnętrznych bibliotek, które są niezbędne do uruchomienia skryptu. Dlatego należy je zainstalować w środowisku uruchomieniowym. Zalecane jest uruchamianie skryptów w wirtualnym środowisku uruchomieniowym. Wirtualne środowisko uruchomieniowe, jest swojego rodzaju odizolowanym katalogiem, zawierającym instalację języka Python, zainstalowane na potrzeby programu biblioteki, oraz programy, które będą w nim uruchamiane. Do utworzenia nowego wirtualnego środowiska uruchomieniowego należy użyć programu *virtualenv*. Wcześniej jednak należy sprawdzić czy na maszynie jest zainstalowany Python w wersji 3.8, aby to uczynić należy wywołać:

```
python3 --version
```

Jeżeli wersja zainstalowanego będzie niższa niż 3.8, to wówczas konieczna będzie jego aktualizacja bądź ponowna instalacja. W tym celu należy w terminalu systemu Ubuntu należy uruchomić następujące komendy:

```
sudo apt-get update
```

```
sudo apt-get install python3.8
```

Po zainstalowaniu (wzorując się na poprzednim poleceniu) należy ponownie sprawdzić wersję Python oraz sprawdzić czy wraz z Python został zainstalowany program PIP. W celu sprawdzenia istnienia w systemie programu PIP, należy wywołać polecenie:

```
pip --version
```

Polecenie to wyświetli informacji o wersji programu PIP. Jeżeli, program nie będzie zainstalowany to należy go zainstalować, poleceniem:

```
sudo apt-get install python3-pip
```

Bardzo ważne jest upewnienie się czy PIP został zainstalowany w odpowiedniej dla języka Python 3.8 wersji. Odczytać wersję języka Python dla której został przeznaczony PIP można na końcu komunikatu zwróconego po wykonaniu komendy sprawdzającej wersję PIP. Może wystąpić przypadek, w którym na maszynie będą zainstalowane dwie wersje PIP, jednak dla Python 2, a druga przeznaczona dla Python 3. Bezpośrednie odwołanie się do konkretnej wersji uzyskuje się przez wywołanie: `pip2` lub `pip3` (np. `pip3 --version`). Po prawidłowej instalacji Pythona i PIP, należy zainstalować aplikację *virtualenv*, aby to uczynić należy wykonać:

```
pip install virtualenv
```

Po instalacji, w folderze projektu należy utworzyć nowe wirtualne środowisko uruchomieniowe oraz go aktywować:

```
virtualenv -p python3.8 ecg_env
```

```
source ecg_env/bin/activate
```

Po aktywacji środowiska należy uruchomić skrypt instalujący niezbędne biblioteki:

```
./setup.sh
```

1.1.3.3. Trenowanie sieci

Do wytrenowania sztucznej sieci neuronowej potrzeba dużego zbioru oznaczonych sygnałów EKG. Zebranie takich zbiorów jest zazwyczaj kosztowne i zajmuje wiele czasu. Dlatego zrezygnowano ze zbierania sygnałów EKG za pomocą opaski i postanowiono skorzystać z publicznych baz danych sygnałów EKG. Jedną z najbardziej popularnych baz danych 1-odprowadzeniowego EKG jest baza danych udostępniona na stronie PhysioNet w ramach konkursu „AF Classification from a Short Single Lead ECG Recording - The PhysioNet Computing in Cardiology Challenge 2017”. Baza posiada dwa główne zbiory sygnałów: *training2017.zip*, oraz *sample2017.zip*. Sygnały EKG zostały zebrane za pomocą rejestratora *AliveCor*. Pierwszy zbiór zawierający 8 528 sygnałów przeznaczony jest do trenowania sieci, a drugi z 300 sygnałami może posłużyć do jej testowania. Czas rejestracji sygnałów z bazy mieści się od 9 do ok. 60 sekund. Sygnały EKG były próbkowane z częstotliwością 300 Hz i zostały zostały wstępnie pasmowo przefiltrowane przez rejestrator *AliceCor*. Próbkki sygnałów zostały zapisane w formacie plików Matlabu tj. *.mat*, a ich opis został zawarty w plikach *WFDB*. Klasyfikacja sygnałów do odpowiednich klas została zapisana do plików *REFERENCE-v3.csv* znajdującym się wewnątrz zbiorów. Więcej szczegółów o klasyfikacji sygnałów można znaleźć na stronie <https://physionet.org/content/challenge-2017/1.0.0/>.

W celu pobrania z serwera PhysioNet danych treningowych i testowych należy uruchomić znajdujący się w katalogu *data* skrypt o nazwie *setup.sh*:

```
cd data
```

```
./setup.sh
```

```
cd ..
```

Skrypt ten oprócz pobrania i rozpakowania danych, uruchomi znajdujący się w tym samym katalogu program *build_datasets.py*. W efekcie czego wewnątrz folderu *data* zostaną utworzone dwa katalogi *training2017*, *sample2017* oraz pliki *REFERENCE-v3.csv*, *dev.json*, *train.json*. Pliki JSON posłużą do konfiguracji procesu trenowania i testowania.

Proces trenowania sieci należy rozpocząć uruchamiając skrypt *train.py* z parametrem, wskazującym lokalizację konfiguracyjnego pliku *config.json*.

```
python train.py config.json
```

W pliku *config.json* znajdują się wartości parametry warstw sieci, procesu uczenia oraz lokalizacji, w których znajdują się pliki *dev.json*, *train.json* oraz w których ma być zapisany wytrenowany model sieci. Domyślna zawartość pliku konfiguracyjnego *config.json* została przedstawiona poniżej.


```
{
  "conv_subsample_lengths": [1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2],
  "conv_filter_length": 16,
  "conv_num_filters_start": 32,
  "conv_init": "he_normal",
  "conv_activation": "relu",
  "conv_dropout": 0.2,
  "conv_num_skip": 2,
  "conv_increase_channels_at": 4,

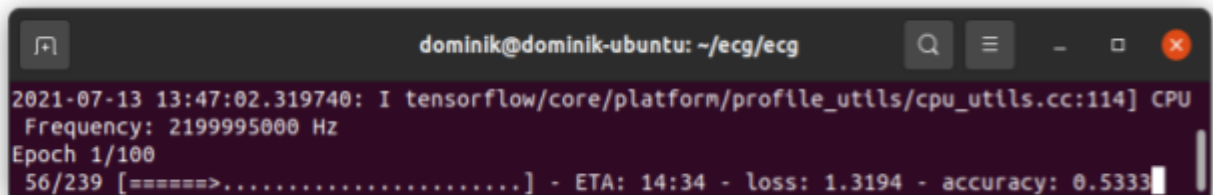
  "learning_rate": 0.001,
  "batch_size": 32,

  "train": "data/train.json",
  "dev": "data/dev.json",

  "generator": true,

  "save_dir": "saved"
}
```

Po uruchomieniu procesu uczenia w terminalu będzie wyświetlany postęp uczenia sieci. Przykładowe okno terminala z uruchomionym procesem uczenia zostało pokazane na Rys. 4.



Rys. 4. Okno terminala systemu Ubuntu 20.0 z uruchomionym procesem uczenia sztucznej splotowej sieci neuronowej.

Proces uczenia uruchomiony na średniej klasie komputerze i domyślnych ustawieniach (patrz plik *config.json*) może potrwać do 6 godzin. Zaleca się aby do trenowania sieci wykorzystać komputer z szybkim procesorem lub kartami graficznymi, który jest predysponowany do tego typu zadań.

Podczas procesu uczenia w katalogu o nazwie wskazanej w polu „*save_dir*” (patrz plik *config.json*), będą zapisywane wagi sieci (w plikach *.hdf5*). Liczba plików będzie odpowiadała liczbie epok występujących w procesie uczenia. Dla użytego zestawu treningowego i domyślnej konfiguracji parametrów, liczba epok nie powinna przekraczać 18. W nazwach plików z wagami sieci będą ujęte wskaźniki:

loss – błąd uczenia sieci (im mniejszy tym lepiej);

accuracy – poprawność modelu w przypisywaniu klas do danych treningowych;

val_loss oraz $val_accuracy$ – są analogiczne do powyższych, z tą zasadniczą różnicą, że $val_$ to miara na części sprawdzającej danych treningowych, czyli tej która nie była wykorzystana do uczenia.

Format zapisu nazwy pliku z wagami sieci jest następujący:

$\{val_loss:.3f\}-\{val_accuracy:.3f\}-\{epoch:03d\}-\{loss:.3f\}-\{accuracy:.3f\}.hdf5$

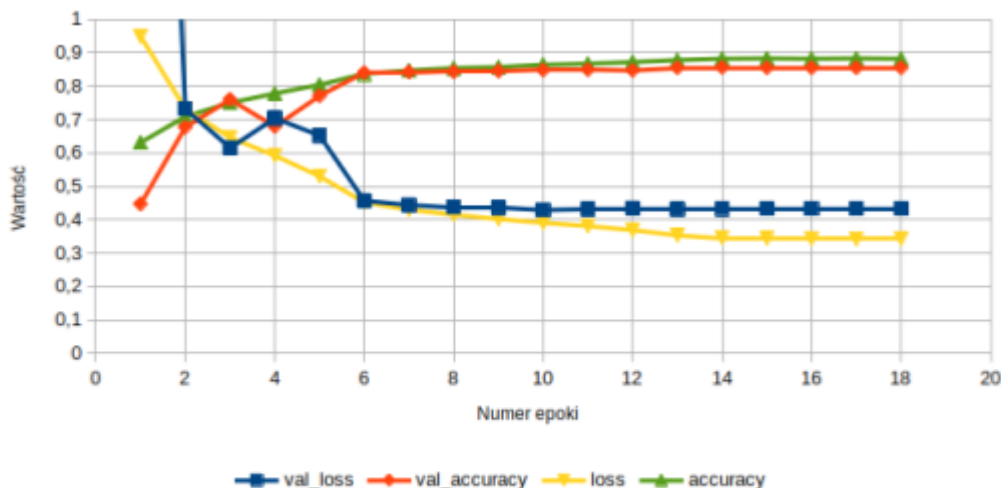
gdzie $epoch$ to numer epoki z jakiej pochodzą wagi.

Po zakończeniu procesu uczenia na podstawie wcześniej opisywanego treningowego zbioru sygnałów zostały wygenerowany plik z wagi o nazwie:

$0.433-0.855-018-0.343-0.882.hdf5$

Plik ten należy przenieść do katalogu głównego.

Zależność wartości błędu uczenia i poprawności modelu w przypisywaniu do klas sygnałów testowych i treningowych względem numeru epoki została zobrazowana na wykresie z Rys. 5. Na podstawie wykresu można ocenić jakość opracowanego modelu.



Rys. 5. Wykres przedstawiający błąd uczenia i poprawność modelu w przypisywaniu do klas dla danych testowych i uczących.

Wartości $accuracy$ i $val_accuracy$ na wykresie wzrastają w podobny sposób. Od ok. 6 epoki różnica między nimi jest niewielka i utrzymuje się na stałym poziomie, co świadczy, że nie wystąpiło zjawisko *overfittingu*, czyli nadmiernego dopasowania sieci do danych. W dużym uproszczeniu można przyjąć, że sieć uczy się, a nie zapamiętuje odpowiedzi.

1.1.4. Internetowa aplikacja modułu analizującego

Jednym z założeń projektowych było przesyłanie spróbkowanego sygnału EKG na wejście modułu analizującego za pomocą sieci Ethernet i protokołu HTTP. Do tego celu użyto metody

POST. Ustalono, że pole danych powinno być formatowane zgodnie z formatem JSON, którego pola przedstawiono w poniższej tabeli.

Format danych przychodzących do modułu analizującego	Format danych wchodzących z modułu analizującego
<pre>{ "Siginfo": { "Units": "mV", "Baseline": 0, "Gain": 1000, "Hand": "left", "Description": "ECG", "fmt": "16" }, "Fs": 512, "Samples": [-145.912 -134.468, -123.596, -127.410, -128.555, -131.225] }</pre>	<pre>{ "Siginfo": { "Units": "mV", "Baseline": 0, "Gain": 1000, "Hand": "left", "Description": "ECG", "fmt": "16" }, "Fs": 512, "HRmean": 89.27, "Result": "N", "SDNN": 55.08, "Samples": [-144.012 -133.464, -120.596, -123.410, -129.555, -130.225] }</pre>

Do zapewnienia komunikacji HTTP zastosowano mikro framework *Flask*. Oprogramowanie jest automatycznie instalowane, gdy wykonywane są kroki z wcześniej opisanego punktu pt. *Przygotowanie środowiska uruchomieniowego*. Skrypt odpowiedzialny za uruchomienie frameworka oraz realizację zadań modułu analizującego został pokazany w załączniku nr 6.

Główną funkcją skryptu jest *ai_ecg()*. Kod metody został przedstawiony poniżej. Metoda odbiera dane, zapisuje je do pliku, filtruje, oblicza tętno i SDNN, a na końcu uruchamia predykcję dla wejściowego sygnału EKG.

```
@app.route('/ai_ecg', methods=['POST'])
def ai_ecg():
    data = {"Result": None}
    wynik = 'E'
    if request.method == "POST":
        request_data = request.get_json()
        assert isinstance(request_data, object)
        write_to_file(request_data) # zapis otrzymanych danych pliku
        samples = request_data['Samples'] # wyodrębnienie próbek EKG
        fs = request_data['Fs']
```

```
gain = request_data['Siginfo']['Gain']
if "Hand" in request_data['Siginfo']:
    if request_data['Siginfo']['Hand'] == 'right':
        flipped = True
    else:
        flipped = False
else:
    flipped = False
    request_data['Siginfo']['Hand'] = 'left'
samples, fs, samples_AI, fs_AI = filtering(samples, gain, fs, flipped) # filtracja i
resampling sygnału EKG
hr, sdnn = analysing(samples, fs) # obliczanie tętna i sdnn
wynik = predict(samples_AI) # przekazanie próbek na wejście sieci neuronowej
response_data = request_data
response_data["Result"] = wynik # formatowanie wyniku
response_data["HRmean"] = round(hr, 2)
response_data["SDNN"] = round(sdnn, 2)
response_data["Fs"] = fs
samples_list = samples.tolist()
response_data["Samples"] = json.dumps(samples_list)
return jsonify(response_data)
```

1.1.4.1. Zapis wejściowych sygnałów EKG

Za zapis do pliku odpowiada funkcja `write_to_file(jsondata)`, która nieprzetworzone dane wejściowe zapisuje do pliku JSON w folderze `receiveddata`. Każdy plik danych rozpoczyna się przedrostkiem `ecg_` i kolejno występuje stempel czasu otrzymania danych. Zgromadzone dane nie są kasowane dlatego warto raz na jakiś czas skasować zapisane pliki, aby nie dopuścić do wyczerpania całej przestrzeni dyskowej. Pliki z danymi mogą być w przyszłości użyte do powiększenia treningowego zbioru sygnałów EKG lub stworzenia własnej bazy sygnałów. Własną bazę można by poddać ocenie eksperckiej i klasyfikacji sygnałów pod względem występowania nieprawidłowości pracy serca. Eksperci mogliby sklasyfikować sygnały na więcej niż 4 grupy. W konsekwencji czego sztuczna sieć neuronowa mogłaby rozróżniać bardziej szczegółowe nieprawidłowości pracy serca.

```
def write_to_file(jsondata):
    ct = datetime.datetime.now()
    filename = 'receiveddata/ecg'+str(ct)+'.json'
    datafile = open(filename, "w")
    datafile.write(json.dumps(jsondata, indent=4))
    datafile.close()
```

1.1.4.2. Decymacja i filtracja

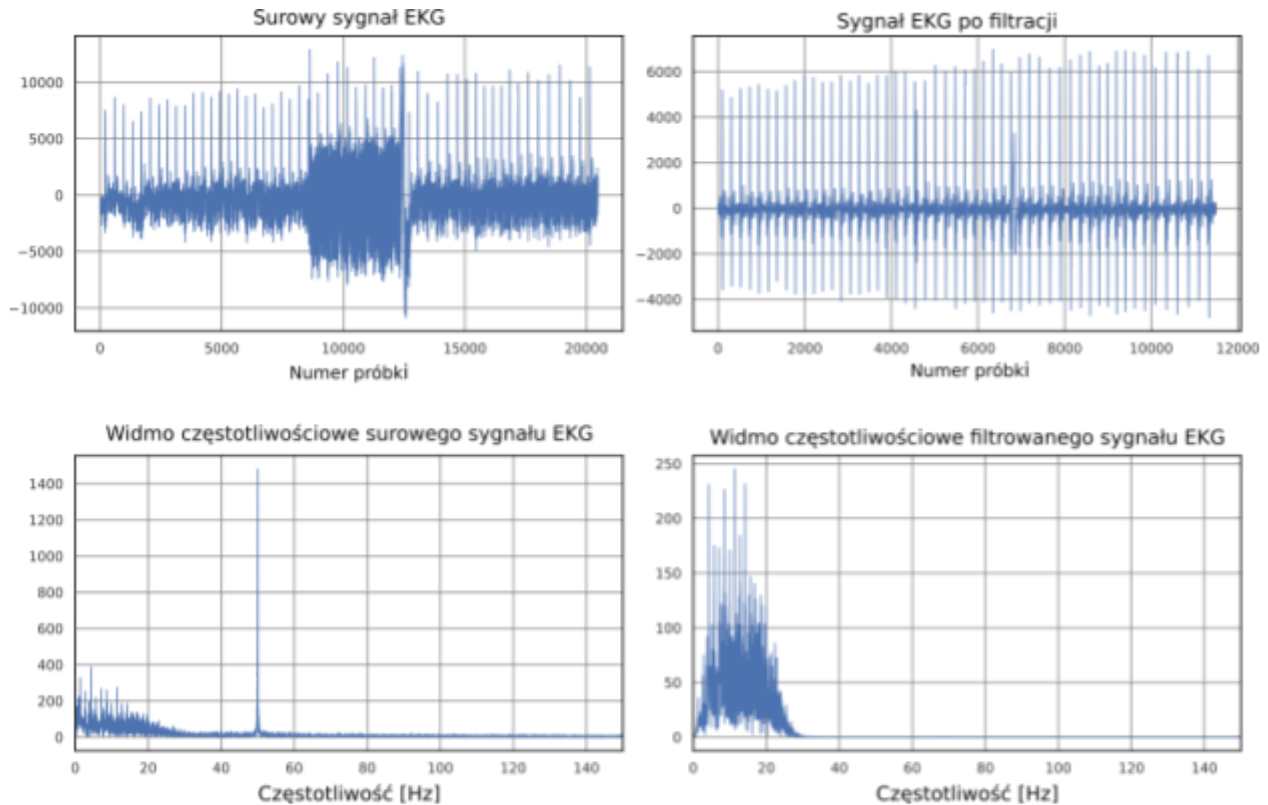
Za proces dostosowania surowego sygnału do wymagań sieci odpowiada funkcja o nazwie `filtering()`, która w zasadzie korzysta z funkcji `filtering()`, `resampling()`, `flipping()` obiektu `ecg_filter`. Kod źródłowy funkcji został pokazany poniżej.

```
def filtering(samples, gain, sample_rate, flipped=True):
```

```
samples = np.array(samples, dtype=np.float32)
if flipped:
    samples = filter.flipping(samples)
samples_f = filter.filtering(samples, sample_rate)
if gain == 0:
    gain = 1
gain_new = 1000 # konwersja na mV
samples_gf = samples_f * (gain_new / gain)
if sample_rate != 300:
    samples_AI, sample_rate_AI = filter.resampling(samples_gf, sample_rate, 300)
else:
    samples_AI = samples_f
    sample_rate_AI = sample_rate
return samples_f, sample_rate, samples_AI, sample_rate_AI
```

W pierwszy etapie przetwarzania sprawdzana jest czy wymagana jest zmiana biegunowości sygnału EKG. Jeżeli jest wymagana to sygnał jest obracany. Kolejno dokonywana jest filtracja sygnału. Podczas filtracji eliminowana jest składowa wolno zmienna (funkcja *remove_baseline_wander()* biblioteki HeartPy), kolejno dokonywana jest filtracja sygnału filtrem pasmowym i filtrem dolnoprzepustowym. Filtr pasmowy ma za zadanie usunięcie składowej sieci elektrycznej o częstotliwości 50 Hz, a filtr dolnoprzepustowy składowych wyższych od 30 Hz. Kolejno tworzona jest kopia sygnału, która jest poddawana dalszemu przetwarzaniu. Dla ułatwienia oznaczy ją jako sygnał nr 2 (w kodzie *samples_AI*), a sygnał oryginalny jako sygnał nr 1 (w kodzie *samples_f*). Kolejno sygnał nr 2 jest poddawany repróbkowaniu do częstotliwości 300Hz (lub decymacji do 256 Hz). Jest to wymagane ponieważ opaska Sidly Care EKG próbkuje sygnał EKG z częstotliwością 512 Hz, a sieć została wytrenowana na danych próbkowanych z częstotliwością 300 Hz. Sygnał nr 1 jest kolejno poddawany analizie i jest zwracany do aplikacji bazodanowej, a sygnał nr 2 jest podawany na wejście sztucznej sieci neuronowej.

Na Rys. 6 przedstawiono surowy sygnał EKG z opaski Sidly Care, jego przefiltrowaną wersję oraz widma częstotliwościowe tych sygnałów.



Rys. 6: Surowy sygnał EKG i jego widmo częstotliwościowe oraz sygnał EKG po filtracji i jego widmo częstotliwościowe.

Porównując widma można zauważyć, że z widma sygnału po filtracji zniknął słupek przy częstotliwości 50 Hz oraz zostały wytłumione słupki powyżej 25 Hz. Wobec tego można stwierdzić, że filtracja sygnału EKG działa zgodnie z założeniem.

1.1.4.3. Obliczanie tętna

Sygnał EKG po filtracji jest poddawany procesowi analizy, który został zrealizowany wewnątrz funkcji *analysing()*. Zadanie tej funkcji właściwie ogranicza się do wywołania metody *process()* biblioteki HeartPy, która to dokonuje obliczenia tętna oraz innych wskaźników takich jak: SDNN, SDSD, RMSSD, SD1, SD2, SD1/SD2 itp. Biblioteka do obliczania wskaźników używa specjalnych algorytmów, które zostały opracowane na potrzeby zaszumionych sygnałów EKG. Więcej informacji o bibliotece można znaleźć w artykule ją opisującym: P. van Gent, H. Farah, N. van Nes b, B. van Arem, HeartPy: *A novel heart rate algorithm for the analysis of noisy signals*, Transportation Research Part F, 2019. Artykuł ten jest wielokrotnie cytowany, co potwierdza słuszność zastosowania biblioteki w opracowanym rozwiązaniu.

W bieżącym etapie rozwoju systemu użyte będą: wartość tętna oraz wskaźnik SDNN.

```
def analysing(samples, sample_rate):
    working_data, measures = hp.process(samples, sample_rate=sample_rate)
    hr = measures['bpm']
```

```
sdnn = measures['sdnn']  
return hr, sdnn
```

1.1.4.4. Klasyfikacja sygnałów EKG

W finalnym etapie sygnały EKG podawane są na wejście sieci neuronowej. Odpowiada za to funkcja *predict()*, która początkowo wyrównuje długość sygnału tak, aby była ona wielokrotnością 256 (STEP=256). Kolejno za pomocą metody *process_x* dokonywana jest normalizacja sygnału i finalnie podany jest on na wejście sieci za pomocą metody *predict()* (należącej do modelu sieci). Metoda zwraca tablicę z wartościami predykcji (tablica *probs*) dla każdej z 4 klas. Liczba predykcji (długość tablicy *probs*) jest równa liczbie próbek sygnału podzielonej przez 256 (STEP). Kolejno wybierana jest wartość modalna, która wskazuje numer klasy do której został przypisany sygnał. Kod metody *predict()* jest pokazany poniżej.

```
def predict(samples):  
    print("Klasyfikacja...")  
    trunc_samp = STEP * int(len(samples) / STEP)  
    ecg = samples[:trunc_samp]  
    x = preproc.process_x([ecg]) # normalizacja  
    probs = model.predict(x) # predykcja  
    prediction = sst.mode(np.argmax(probs, axis=2).squeeze())[0][0]  
    del probs  
    print('Wynik: ' + preproc.int_to_class[prediction])  
    return preproc.int_to_class[prediction]
```

1.1.4.5. Wczytanie modelu sieci neuronowej

Upřednio wytrenowany model sieci należy wczytać do aplikacji. Do tego celu została przygotowana metoda *load_model_ecg()*, która jest uruchamiana podczas uruchamiania głównego skryptu aplikacji – *main.py*. Metoda na początku wczytuje plik *config.json* z ustawieniami sieci, buduje sieć, a na końcu wczytuje wagi sieci. Wagi sieci są umieszczone w pliku *.hdf5*. W rozważanym przypadku plik będzie to plik *0.433-0.855-018-0.343-0.882.hdf5*.

```
def load_model_ecg():  
    print("Ładowanie wag modelu sieci...")  
    global model  
    global preproc  
    preproc = util.load(".")  
    params = json.load(open("config.json"))  
    params.update({  
        "compile": False,  
        "input_shape": [None, 1],  
        "num_categories": len(preproc.classes)  
    })  
    model = network.build_network(**params)  
    model.load_weights("0.433-0.855-018-0.343-0.882.hdf5") # plik z wagami sieci  
    print("Wagi modelu załadowane")
```


1.1.4.6. Uruchomienie

Aplikację modułu analizującego należy uruchomić jako usługę działającą w tle. Można to uczynić wywołując w terminalu komendę:

```
nohup python -u main.py >> logi_ecg_serwera.log &
```

Jeżeli przed wywołaniem powyższej komendy nie zostanie aktywowane wirtualne środowisko uruchomieniowe to należy je aktywować poleceniem:

```
source ecg_env/bin/activate
```

Po uruchomieniu aplikacja będzie oczekiwać na dane na porcie 5000. Dane należy przesłać w formacie JSON metodą POST na adres:

```
http://18.196.183.99:5000/ai_ecg
```

Komunikaty zwracane przez aplikację będą zapisywane w pliku *logi_ecg_serwera.log*.

1.1.5. Badania testowe

Aplikację modułu analizującego poddano badaniom testowym podczas, których wysyłano do aplikacji sygnały EKG zarejestrowane opaską Sidly Care EKG. Wszystkie sygnały pochodziły od osób, które zadeklarowały, że są osobami zdrowymi. Podczas przeprowadzonych badań sieć klasyfikowała zebrane sygnały jako sygnały osób zdrowych lub jako sygnały zaszumione. Sygnał zaszumiony był wykryty gdy pacjent podczas pomiaru wykonywał różne ruchy. Szczegóły badań zostały zawarte w dalszej części raportu.

1.1.6. Załączniki

1.1.6.1. Załącznik nr 1 - skrypt util.py – skrypt do serializacji i deserializacji danych.

```
import os
import cPickle as pickle

def load(dirname):
    preproc_f = os.path.join(dirname, "preproc.bin")
    with open(preproc_f, 'r') as fid:
        preproc = pickle.load(fid)
    return preproc

def save(preproc, dirname):
    preproc_f = os.path.join(dirname, "preproc.bin")
    with open(preproc_f, 'w') as fid:
        pickle.dump(preproc, fid)
```


1.1.6.2. Załącznik nr 2 - skrypt main.py – skrypt realizujący główną funkcjonalność modułu analizującego.

```
# coding=utf-8
import json
import numpy as np
import scipy.stats as sst
import heartpy as hp
import network
import util
import gc
import datetime

from flask import Flask, request, jsonify
import ecg_filter as filter

app = Flask(__name__)

STEP = 256
model = None
preproc = None

@app.route('/ai_ecg', methods=['POST'])
def ai_ecg():
    data = {"Result": None}
    wynik = 'E'
    if request.method == "POST":
        request_data = request.get_json()
        assert isinstance(request_data, object)
        write_to_file(request_data) # zapis otrzymanych danych pliku
        samples = request_data['Samples'] # wyodrębnienie próbek EKG
        fs = request_data['Fs']
        gain = request_data['Siginfo']['Gain']
        if "Hand" in request_data['Siginfo']:
            if request_data['Siginfo']['Hand'] == 'right':
                flipped = True
            else:
                flipped = False
        else:
            flipped = False
            request_data['Siginfo']['Hand'] = 'left'
        samples, fs, samples_AI, fs_AI = filtering(samples, gain, fs, flipped) # filtracja i
        resampling sygnału EKG
        hr, sdnn = analysing(samples, fs) # obliczanie tętna i sdnn
        wynik = predict(samples_AI) # przekazanie próbek na wejście sieci neuronowej
        response_data = request_data
        response_data["Result"] = wynik # formatowanie wyniku
        response_data["HRmean"] = round(hr, 2)
        response_data["SDNN"] = round(sdnn, 2)
        response_data["Fs"] = fs
        samples_list = samples.tolist()
```

```
response_data["Samples"] = json.dumps(samples_list)
return jsonify(response_data)

def load_model_ecg():
    print("Ładowanie modelu sieci...")
    global model
    global preproc
    preproc = util.load(".")
    params = json.load(open("config.json"))
    params.update({
        "compile": False,
        "input_shape": [None, 1],
        "num_categories": len(preproc.classes)
    })
    model = network.build_network(**params)
    model.load_weights('model.hdf5')
    print("Model załadowany")

def predict(samples):
    print("Klasyfikacja...")

    trunc_samp = STEP * int(len(samples) / STEP)
    ecg = samples[:trunc_samp]
    x = preproc.process_x([ecg])
    probs = model.predict(x)
    prediction = sst.mode(np.argmax(probs, axis=2).squeeze())[0][0]
    del probs
    print('Wynik: ' + preproc.int_to_class[prediction])
    return preproc.int_to_class[prediction]

def filtering(samples, gain, sample_rate, flipped=True):
    samples = np.array(samples, dtype=np.float32)
    if flipped:
        samples = filter.flipping(samples)
    samples_f = filter.filtering(samples, sample_rate)
    if gain == 0:
        gain = 1
    gain_new = 1000 # konwersja na mV
    samples_gf = samples_f * (gain_new / gain)
    if sample_rate != 300:
        samples_AI, sample_rate_AI = filter.resampling(samples_gf, sample_rate, 300)
    else:
        samples_AI = samples_f
        sample_rate_AI = sample_rate
    return samples_f, sample_rate, samples_AI, sample_rate_AI

def analysing(samples, sample_rate):
    working_data, measures = hp.process(samples, sample_rate=sample_rate)
    hr = measures['bpm']
```

```
sdnn = measures['sdnn']
return hr, sdnn

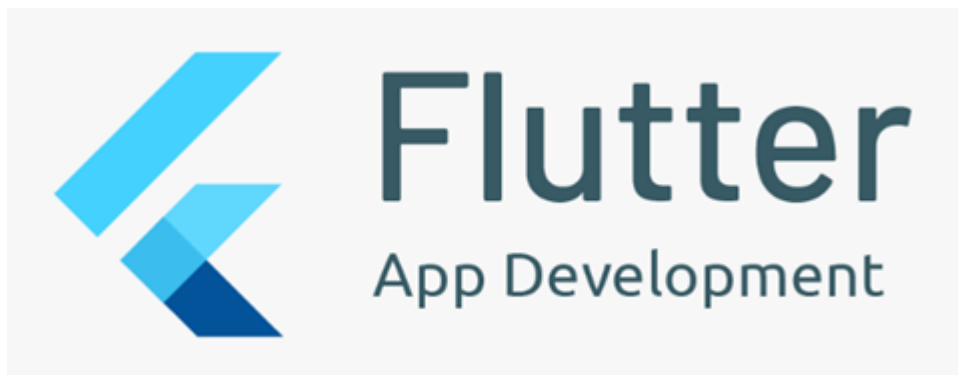
def write_to_file(jsondata):
    ct = datetime.datetime.now()
    filename = 'receiveddata/ecg' + str(ct) + '.json'
    datafile = open(filename, "w")
    datafile.write(json.dumps(jsondata, indent=4))
    datafile.close()

if __name__ == '__main__':
    gc.enable()
    load_model_ecg()
    # app.run(debug=True, threaded=False, host="0.0.0.0")
    # app.run(host="0.0.0.0", use_reloader=False)
    app.run(host="0.0.0.0")
```



1.2. Aplikacja mobilna

1.2.1. Framework Flutter



Rys. 7. Logo frameworka Flutter.

Aplikacja *flutter_ecg* została napisana w za pomocą frameworka *Flutter*, który jest deweloperskim zestawem oprogramowania UI stworzonym i rozwijanym przez Google. Flutter stworzony został w 2017 roku, jednak premiera jego stabilnej wersji miała miejsce w 4 grudnia 2018 r. podczas wydarzenia *Flutter Live*. Mimo tego, że jest to relatywnie nowe narzędzie, dzięki programistom Google oraz społeczności zgromadzonej wokół Fluttera narzędzie rozwija się bardzo dynamicznie i z każdą kolejną wersją przynosi więcej możliwości co przekłada się na większą liczbę zastosowań. Obecna wersja framework'a to 2.2.3.

Programy tworzone przy pomocy *Fluttera* bazują na języku programowania *Dart*. Dart jest obiektowym językiem programowania ogólnego przeznaczenia. Został opracowany przez firmę Google i udostępniony w 2011 roku.

Flutter pozwala na jednoczesne pisanie aplikacji na wiele platform, dzięki czemu za pomocą jednego kodu można stworzyć jednocześnie aplikację na system Android jak i na IOS. *Flutter* od konkurencyjnych rozwiązań pozwalających na tworzenie aplikacji crossplatformowych, wyróżnia się tym, że aplikacje napisane w nim są aplikacjami natywnymi. Powoduje to, że programiści tworzący aplikacje nie są ograniczeni wyłącznie do ekosystemu *SDK Fluttera*, ale mogą korzystać z możliwości oferowanych przez wykorzystywane platformy. Dodatkowo przewagę *Fluttera* na pewno stanowi duża społeczność oraz liczba dostępnych bibliotek.

Wybór metody napisania aplikacji padł właśnie na ten framework, ponieważ wyróżnia go:

- Niski czas produkcji - czas tworzenia aplikacji w *Flutterze* jest znacznie mniejszy niż stworzenie dwóch aplikacji dla różnych platform np. jednej dla systemu Android, a drugiej dla IOS;

- Niskie koszty - zaprojektowanie aplikacji mobilnej we *Flutterze* na dwa systemy opiera się na niemalże jednym kodzie, do którego należy wprowadzić jedynie niewielkie zmiany, dzięki czemu wystarczy jeden zespół na stworzenie działającej aplikacji na dwa różne systemy;
- Możliwość szybkiej weryfikacji zmian wprowadzonych w kodzie - funkcja Hot Reload, pozwala na wprowadzanie zmian przy włączonej aplikacji. W ten sposób wprowadzone usprawnienia można zobaczyć w mniej niż sekundę, bez utraty czasu na uruchamianie aplikacji;
- Szeroki wybór komponentów - mimo że Flutter nie korzysta bezpośrednio z widżetów i rozwiązań dostępnych w przypadku tworzenia aplikacji natywnych, to biblioteka *Fluttera* umożliwia dostęp do wielu różnych komponentów, za pomocą których można tworzyć strukturę menu, czcionki, buttony i schematy układów;
- Możliwość tworzenia przyciągających wzrok animacji.

Przykłady aplikacji stworzonych za pomocą Fluttera:

- Google Ads,
- Xianyu by Alibaba,
- Reflectly,
- Watermaniac,
- PostMuse,
- Hamilton,
- SpaceX Go!,
- Toughest,
- inKino,
- KlasterMe.

1.2.2. Wybrane biblioteki

Flutter pozwala na dodawanie bibliotek, które umożliwiają programistom tworzenie ciekawszych, a także bardziej zaawansowanych aplikacji. Biblioteki tworzone są przez użytkowników *Fluttera* i znajdują się na stronie: <https://pub.dev>. Udostępnione tam biblioteki podlegają ocenie, w której badane są parametry takie jak przestrzeganie konwencji *Dart*, wspieranie *Null Safety*, czy też wsparcie różnych platform (na dziś dzień w tej kategorii aby dostać maksymalną liczbę punktów biblioteka musi wspierać system Android, IOS oraz Web).

Na następnej stronie znajduje się ocena przykładowej biblioteki, którą jest *shared_preferences* 2.0.6, która znajduje się pod linkiem: https://pub.dev/packages/shared_preferences.

3641
LIKES

130 / 130
PUB POINTS

100 %
POPULARITY

We analyzed this package on Jul 13, 2021, and awarded it 130 pub points (of a possible 130):

✓ Follow Dart file conventions	20/20	✓
✓ Provide documentation	20/20	✓
✓ Support multiple platforms	20/20	✓
✓ Pass static analysis	30/30	✓
✓ Support up-to-date dependencies	20/20	✓
✓ Support sound null safety	20/20	✓

Analysed with Pana 0.18.2, Flutter 2.2.3, Dart 2.13.4.

Rys. 8. Przykładowy wynik oceny biblioteki na przykładzie biblioteki `shared_preferences`.

Oprócz przyznania określonej liczby punktów widoczne są także przyznawane przez użytkowników polubienia, a także procent popularności. Te trzy wartości łącznie z zakładką Readme, opisującej działanie biblioteki, pomagają programiście podjąć decyzję o kompletności biblioteki, a także o tym, czy pasuje do poszczególnego projektu.

Przy tworzeniu aplikacji testowane były biblioteki m.in. do komunikacji po Bluetooth Low Energy czy też rysowania wykresów. Poniżej znajduje się opis najważniejszych przetestowanych bibliotek. Wszystkie wykorzystywane biblioteki wspierały Null Safety, a także pozwalały na wykorzystanie zarówno w systemie Android jak i IOS.

1.2.3.

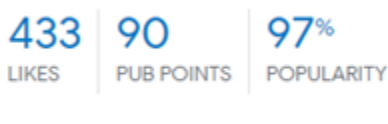
Biblioteki wspierające łączenie po Bluetooth Low Energy (BLE)

1.2.3.1. Biblioteka *flutter_blue* 0.8.0



Rys. 9. Logo biblioteki *flutter_blue*.

Biblioteka *flutter_blue* jest najpopularniejszą biblioteką umożliwiającą komunikację po BLE stworzoną przez pauldemarco.com. Testowaną w aplikacji wersją jest 0.8.0 dostępna na licencji BSD. Zaimplementowana w aplikacji wersja biblioteki została wypuszczona 15 marca 2021r. Na zamieszczonym poniżej rysunku można zobaczyć jak prezentują się wyniki popularności wykorzystania biblioteki, polubienia jej przez użytkowników ale także punkty, które gromadzi się poprzez spełnianie poszczególnych kryteriów.



Rys. 10. Wynik popularności i poprawności biblioteki *flutter_blue*.

Więcej informacji takich jak przykład użycia, dostępne wersje, szczegółowe wyniki punktacji przedstawionej powyżej można zobaczyć pod linkiem: https://pub.dev/packages/flutter_blue.

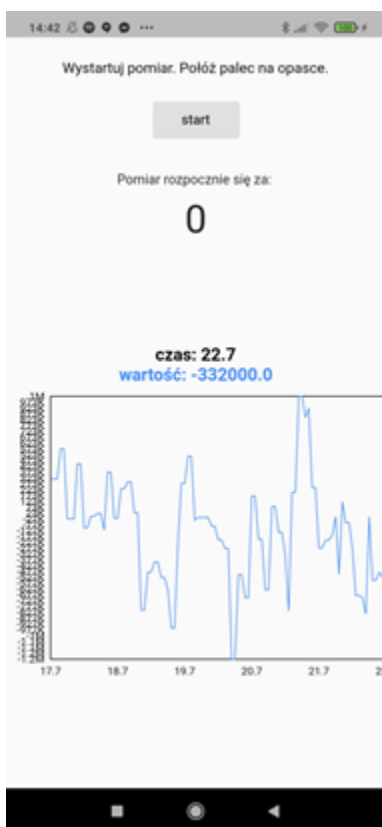
Biblioteka *flutter_blue* jest jedną z najczęściej wybieranych bibliotek umożliwiających cross-platformowe łączenie po BLE. Istnieje wiele materiałów uczących wykorzystywania aplikacji, a także gotowych kodów pozwalających na przetestowanie jej działania.

W zakładce Readme można zobaczyć kompletne instrukcje instalacyjne na obydwa systemy, którymi są Android i IOS, a także przykłady implementacji najważniejszych metod, takich jak skanowanie w poszukiwaniu urządzeń, łączenie z urządzeniem, a także odkrywanie charakterystyk rozgłaszanych przez urządzenie.

Flutter_blue posiada metody umożliwiające weryfikację urządzeń rozgłaszających się po BLE, łączenie z urządzeniami a także wykrywanie i wykorzystywanie dostępnych charakterystyk i deskryptorów.

Mimo dużej popularności a także szerokich opisów działania oraz masowej liczby polubień przez użytkowników biblioteka okazała się być niewystarczająca dla potrzeb pisanej aplikacji. Biblioteka nie była w stanie odebrać wszystkich dostarczanych jej przez opaskę próbek, przez co prezentowany przez nią wynik ekg był nieprawidłowy.

Rysunek widoczny poniżej obrazuje sygnał otrzymywany przez aplikację przy wykorzystaniu tej biblioteki.



Rys. 11. Sygnał otrzymywany przy użyciu biblioteki *flutter_blue*.

1.2.3.2. Biblioteka *flutter_reactive_ble*

Biblioteka *flutter_reactive_ble* w wersji 3.1.1+1 została opublikowana 30.05.2021r. przez meethue.com. Licencja biblioteki to BSD. Biblioteka ta jest mniej popularna niż *flutter_blue*, jednakże spełnia więcej kryteriów dotyczących działania. Na rysunku zamieszczonym na następnej stronie znajdują się wyniki ilości polubień, punkty uzyskane za spełnienie kryteriów, a także wynik popularności aplikacji.

136 | 120 | 92%
LIKES | PUB POINTS | POPULARITY

Rys. 12. Wynik popularności i poprawności biblioteki *flutter_reactive_ble*.

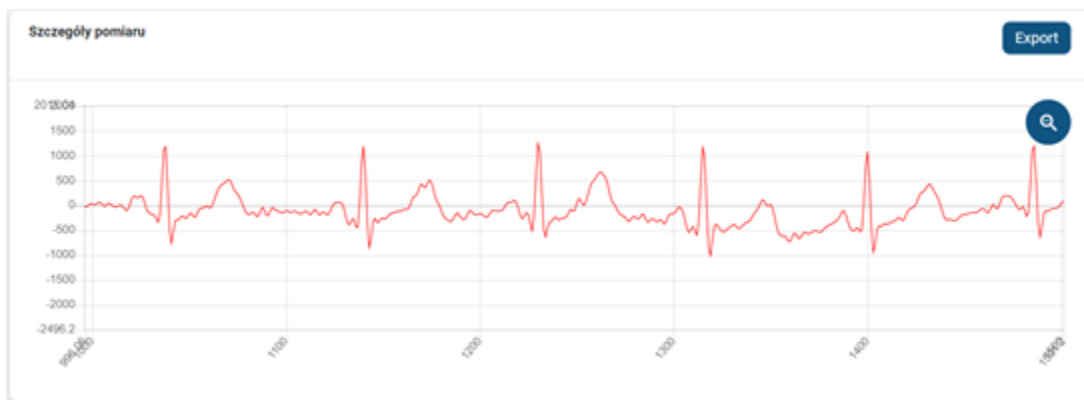
Więcej informacji takich jak przykład użycia, dostępne wersje, szczegółowe wyniki punktacji przedstawionej powyżej można zobaczyć pod linkiem: https://pub.dev/packages/flutter_reactive_ble.

Biblioteka oferuje wszelkie potrzebne do utworzenia połączenia, wyszukiwania urządzeń, ale także odkrywania charakterystyk, pisania i czytania z nich oraz subskrypcji metod. Dodatkowo biblioteka wspiera utrzymywanie statusu połączenia wielu urządzeń BLE.

Zakładka Readme pozwala na poznanie sposobu instalacji biblioteki, a także przykłady sposobów użycia oferowanych przez nią metod.

Flutter_reactive_ble jest biblioteką, która została zaimplementowana w aplikacji. Umożliwia ona odczytywanie wszystkich próbek, dzięki czemu możliwym jest otrzymanie poprawnego wykresu ekg.

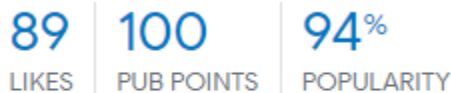
Na rysunku poniżej można zobaczyć otrzymany za pomocą tej biblioteki wykres ekg.



Rys. 13. Sygnał otrzymany przy wykorzystaniu biblioteki *flutter_reactive_ble*.

1.2.3.3. Biblioteka *flutter_bluetooth_serial*

Biblioteka *flutter_bluetooth_serial* testowana była w wersji 0.2.2. Została opublikowana 19.08.2020r. Działa ona na licencji MIT. 4 lipca 2021r. została wypuszczona następna wersja 3.0.0. aplikacji, która obsługuje już null safety, dzięki czemu zwiększyła się jej popularność. Na zrzucie ekranu znajdującym się na następnej stronie można zobaczyć jak przedstawia się popularność a także pozostałe wyniki uzyskane przez bibliotekę.



Rys. 14. Wynik popularności i poprawności biblioteki flutter_bluetooth_serial.

Biblioteka ta nie została wybrana z powodu nie wspierania null safety w momencie testowania bibliotek. Dodatkowo, minusem jest fakt, że wspiera ona tylko jeden system, którym jest Android. Do systemu IOS trzeba by było wybrać inną bibliotekę, co bardzo utrudniłoby pracę.

Mimo możliwości połączenia z wieloma urządzeniami, biblioteka flutter_bluetooth_serial posiadała za wiele braków, żeby można było zaimplementować ją w projekcie.

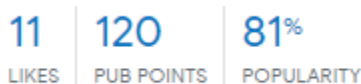
Więcej informacji takich jak przykład użycia, dostępne wersje, szczegółowe wyniki punktacji przedstawionej powyżej można zobaczyć pod linkiem: https://pub.dev/packages/flutter_bluetooth_serial.

1.2.4. Biblioteki do rysowania dynamicznych wykresów

1.2.4.1. Biblioteka Oscilloscope

Oscilloscope to biblioteka umożliwiająca rysowanie wykresów dynamicznych. Testowana w tej aplikacji wersja oscilloscope 0.2.0+1 została opublikowana 26.05.2021r. przez spiralarm.uk. Biblioteka posiada licencję Apache 2.0.

Na rysunku poniżej można zobaczyć ilość polubień od użytkowników, liczbę uzyskanych punktów a także procent popularności. Biblioteka ta jest prosta, nie posiada dużej ilości niepotrzebnych metod, jedynie tyle ile jest potrzebne do uzyskania przejrzystego wykresu. Systemami, na jakich może być wykorzystywana ta biblioteka są: Android, IOS, Linux, MacOS, Web a także Windows.



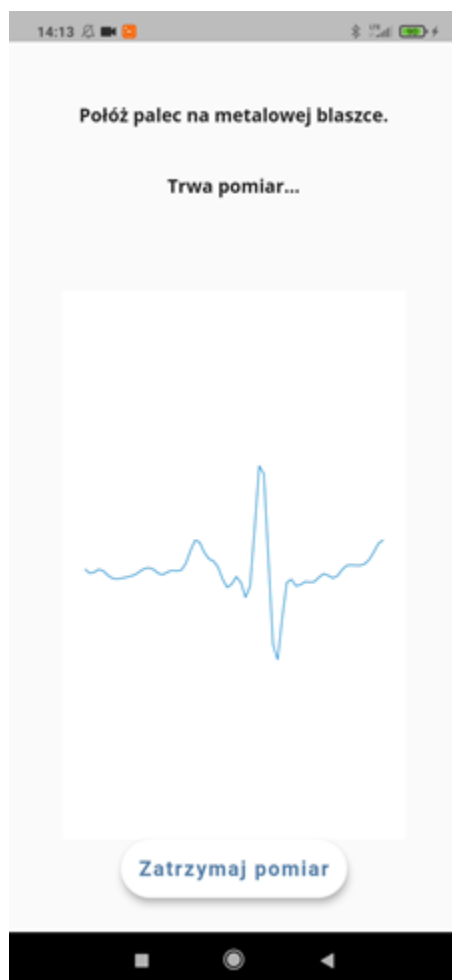
Rys. 15. Wynik popularności i poprawności biblioteki oscilloscope.

Więcej informacji takich jak przykład użycia, dostępne wersje, szczegółowe wyniki punktacji przedstawionej powyżej można zobaczyć pod linkiem: <https://pub.dev/packages/oscilloscope>.

Zakładka Readme posiada szczegółowy opis użytkowania aplikacji, a także opis metod i pól możliwych do modyfikowania i wykorzystywania. Do zmiany są między innymi kolor tła, kolor linii wykresu, szerokość linii. Do generowania wykresu należy podać jedynie wartość dataSet,

która jest zbiorem umieszczanych na wykresie punktów oraz maksymalną i minimalną wartość umieszczoną na osi y wykresu. Niestety wartość ta nie może być dynamicznie podmieniana na wartość największą lub najmniejszą dla aktualnej części danych widocznych na wykresie. Wadą jest to, że nie sposób ustawić czasu przesuwania danych zbliżonego do rzeczywistego, co zaburza odbiór wyniku ekg.

Na rysunku zamieszczonym na następnej stronie widnieje przykład zastosowanej biblioteki.



Rys. 16. Wykres otrzymany przy wykorzystaniu biblioteki oscilloscope.

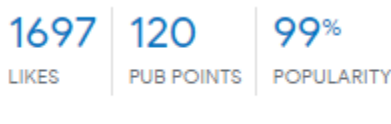
1.2.4.2. Biblioteka *fl_chart*



Rys. 17. Logo biblioteki *fl_chart*.

Biblioteka *fl_chart* jest biblioteką umożliwiającą tworzenie wykresów liniowych, słupkowych i kołowych. Testowana na wersji *fl_chart* 0.36.2 opublikowane 17.01.2021r. przez [ikoshabi.com](https://github.com/ikoshabi/fl_chart). Biblioteka umożliwia tworzenie wykresów na systemy takie jak: Android, IOS, MacOS, Linux, na Web oraz Windows. Całość działa na licencji BSD.

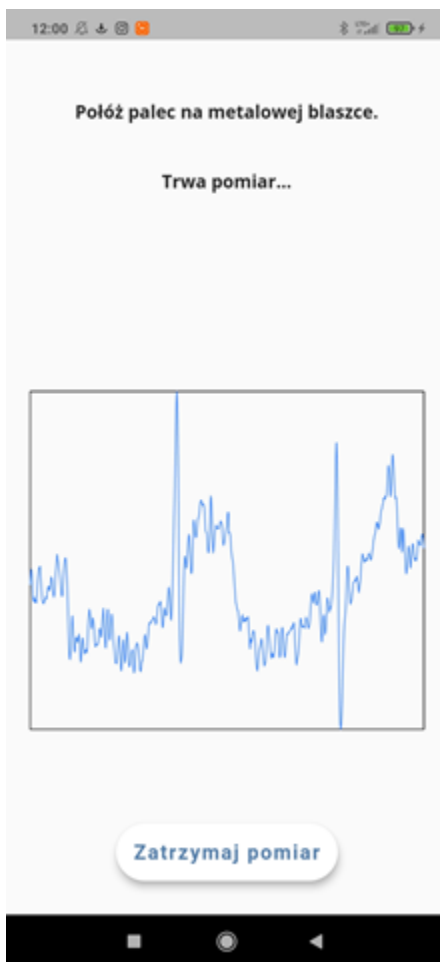
Na poniższym rysunku zaprezentowane zostały wyniki uzyskane dla *fl_chart*. Biblioteka tak jak jej poprzedniczka uzyskała 120/130 punktów, ale cieszy się znacznie większą popularnością od Oscilloscope. Liczba polubień także znacząco przewyższa liczbę uzyskaną przez jej poprzedniczkę.



Rys. 18. Wynik popularności i poprawności biblioteki *fl_chart*.

Więcej informacji takich jak przykład użycia, dostępne wersje, szczegółowe wyniki punktacji przedstawionej powyżej można zobaczyć pod linkiem: https://pub.dev/packages/fl_chart.

W zakładce Readme można zobaczyć przykładowe, możliwe do uzyskania za pomocą tej biblioteki wykresy. Przykładowe wykresy pokazane na stronie są interakcyjne, co także wpływa na chęć zastosowania ich przez programistę. Dodatkowym bonusem jest możliwość zastosowania dynamicznych wartości minimalnej i maksymalnej na osi y wykresu, dzięki czemu wykres może być skalowany do wartości próbek widniejących na wykresie, co sprzyja czytelności wykresu.

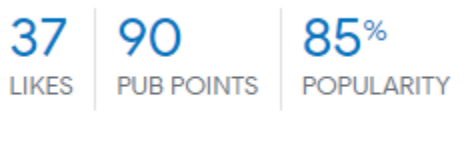


Rys. 19. Wykres otrzymany przy wykorzystaniu biblioteki `fl_chart`.

1.2.4.3. Biblioteka `fl_animated_linechart`

Biblioteka `fl_animated_linechart` testowana była w wersji 1.1.4., która powstała 19.10.2021r. Wspiera ona wiele systemów takich jak Android, IOS, Linux, Mac OS, Web i Windows. Działa na licencji MIT.

Na rysunku przedstawionym poniżej można zobaczyć jak przedstawia się procent popularności, liczba polubień, a także przyznane bibliotece punkty.



Rys. 20. Wynik popularności i poprawności biblioteki `fl_animated_linechart`.

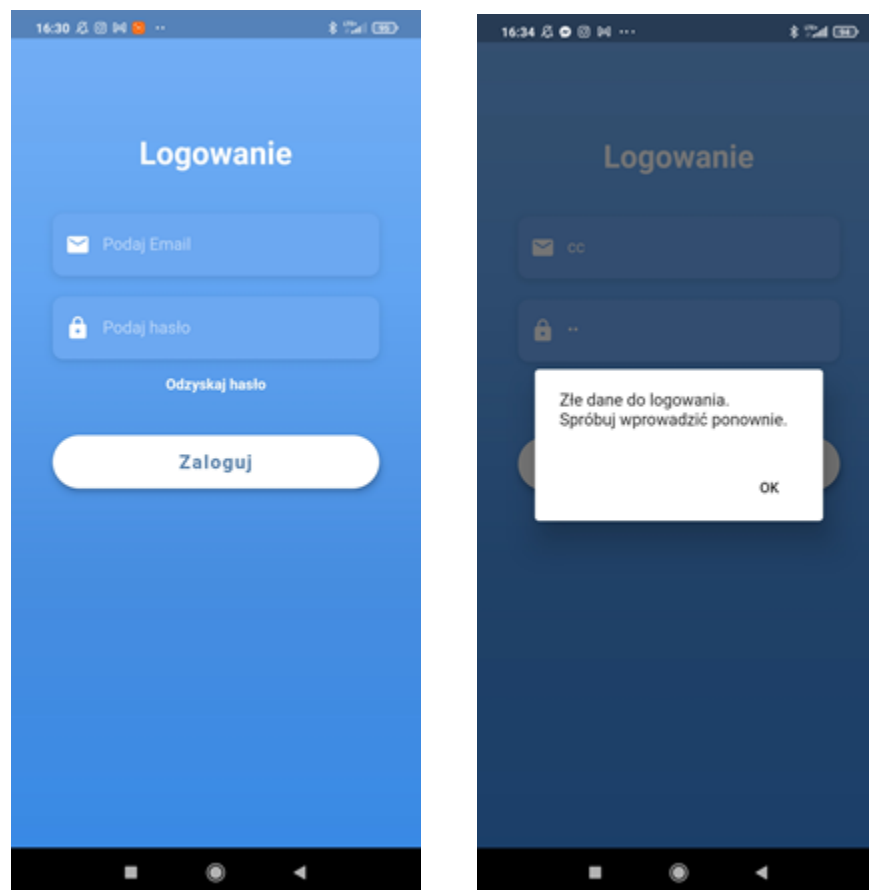
Mimo posiadania wysokich wyników, głównym problem aplikacji jest nie wspieranie null safety, które przekreśliło możliwość zaimplementowania biblioteki w projekcie.

Więcej informacji takich jak przykład użycia, dostępne wersje, szczegółowe wyniki punktacji przedstawionej powyżej można zobaczyć pod linkiem: https://pub.dev/packages/fl_animated_linechart.

1.2.5. Interfejs użytkownika - UX

1.2.5.1. Ekran logowania

Na poniższych zrzutach ekranów można zobaczyć odpowiednio: pusty ekran logowania oraz ekran logowania z okienkiem alertu dialogowego po niewłaściwym podaniu danych do logowania. Oprócz opcji zalogowania się do wcześniej utworzonego konta istnieje jeszcze możliwość wybrania odzyskiwania hasła.



Rys. 21. Wygląd strony logowania aplikacji flutter_ecg.

1.2.5.2. Ekran menu

Ekran menu pozwala na poruszanie się po aplikacji. Z poziomu tej strony użytkownik może podłączyć lub zmienić podłączoną opaskę, dokonać pomiaru, a także wejść w czat. Na górze strony znajduje się odnośnik do ustawień i wylogowania z aplikacji.

Ponadto, znajdują się tu informacje o dacie ostatniego pomiaru, stanie podłączenia opaski, a także lista ostatnio wykonanych pomiarów czy też informacje o opiekunie oraz użytkowniku.



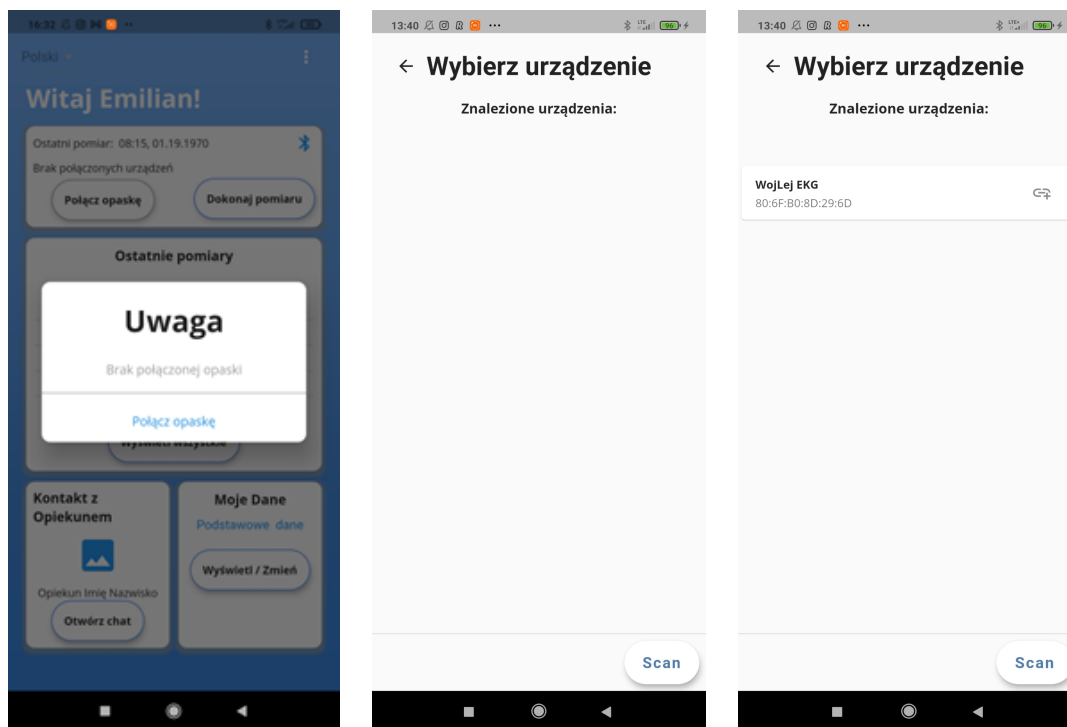
Rys. 22. Wygląd strony głównej aplikacji flutter_ecg.

1.2.5.3. Ekran skanowania urządzeń

Na poniższych zrzutach ekranu znajduje się ekran skanowania dostępnych urządzeń BLE. Po lewej stronie znajduje się widok próby włączenia pomiaru bez wcześniej wybranej opaski. Użytkownik otrzymuje wtedy ekran z alertem, który wybraniu przycisku „Połącz opaskę” przekierowuje do ekranu wyszukiwania. Na środkowym zdjęciu znajduje się ekran, który pokazuje się od razu po wejściu w stronę wyszukiwania urządzeń. Prawa strona pokazuje jak wyświetla się lista dostępnych urządzeń. Użytkownik widzi nazwę znalezionej opaski a także id, które w przypadku aplikacji opartych o system Android jest adresem MAC, natomiast na urządzeniach IOS jest to UUID.

Zrzuty ekranu widoczne poniżej zostały wykonane na urządzeniu działającym na systemie Android.

Po prawej stronie karty urządzenia można zobaczyć przycisk połączenia z urządzeniem. Aby nawiązać łączność z preferowanym urządzeniem należy wykonać proces skanowania, kolejno wybrać urządzenie spośród znalezionych urządzeń i nacisnąć przycisk znajdujący się po prawej stronie.



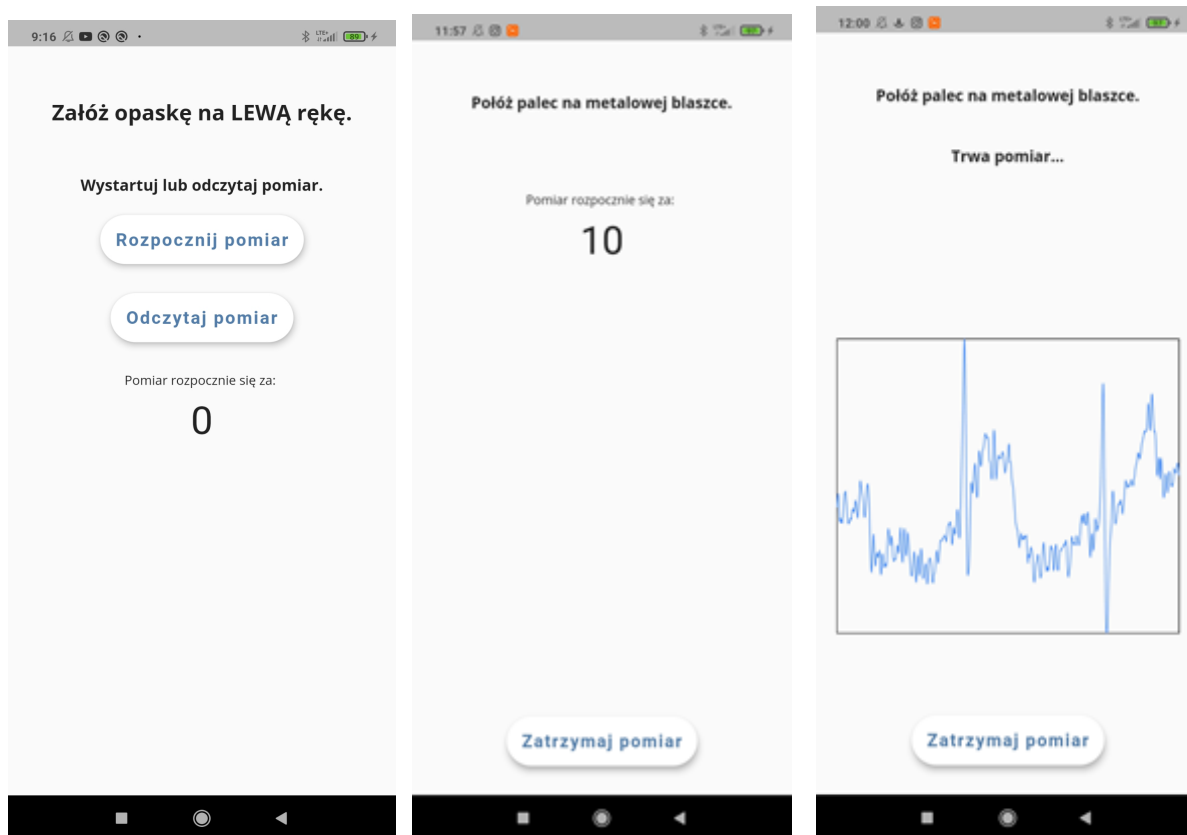
Rys. 23. Wygląd strony wyszukiwania urządzeń BLE aplikacji flutter_ecg.

1.2.5.4. Ekran pomiarowy

Na poniższych rysunkach znajdują się zrzuty ekranu przedstawiające kolejne kroki wykonywania pomiaru. Pierwszym punktem jest wybór startu pomiaru lub odczytu wcześniej zapisanych na opasce danych. Użytkownik proszony jest o założenie opaski na lewą rękę. Prośba ta została umieszczona ze względu na zmienność polaryzacji, powstającej w wyniku zmiany ręki, na której mierzone jest ekg.

Po wyborze pomiaru, użytkownik proszony jest o położenie palca na blaszce. Aplikacja monitoruje wykrycie położenia palca na opasce, a kiedy to zrobi rozpoczyna pomiar. W momencie rozpoczęcia pomiaru zaczyna się odliczanie 10 sekund, w których opaska otrzymuje komendy ustawiające odpowiednie rejestry i przygotowuje się do mierzenia.

Trzeci zrzut ekranu obrazuje wykres odbieranego pomiaru ekg, który pojawia się w momencie zakończenia odliczania. Pomiar trwa 40 sekund, po czym otrzymane dane są przesyłane, a aplikacja wraca do ekranu menu.



Rys. 24. Wygląd strony wykonywania pomiaru aplikacji flutter_ecg.

1.2.6. Wykonywane obliczenia

Zdjęcie poniżej przedstawia rejestr próbki EKG. W każdej ramce przesyłane są 4 kolejne wartości sygnału EKG. Opis rejestru przedstawiony poniżej został zaczerpnięty z dokumentacji technicznej układu MAX30001.

Table 47. FIFO Memory Access and Data Structure Summary

REG	FIFO AND MODE	DATA STRUCTURE (D[23:0])																5	4	3	2	1	0
		23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8						
0x20	ECG Burst	ECG Sample Voltage Data [17:0]														ETAG [2:0]		PTAG [2:0]					
0x21	ECG	ECG Sample Voltage Data [17:0]														ETAG [2:0]		PTAG [2:0]					

Rys. 25. Rejestr z dokumentacji technicznej układu MAX30001.

Próbki sygnału EKG znajdują się na bitach kolejno: 5-7, 8-10, 11-13 oraz 14-16. Kolejne bity mnożone są przez potęgi liczby 256 zaczynając od potęgi równej zero. Poniżej została przedstawiona metoda obliczenia wartości kolejnych czterech próbek sygnału EKG.

```
ecg1 = bytes[5] * 256^0 + bytes[6] * 256^1 + bytes[7] * 256^2;
ecg2 = bytes[8] * 256^0 + bytes[9] * 256^1 + bytes[10] * 256^2;
ecg3 = bytes[11] * 256^0 + bytes[12] * 256^1 + bytes[13] * 256^2;
ecg4 = bytes[14] * 256^0 + bytes[15] * 256^1 + bytes[16] * 256^2;
```

Aby wyciągnąć samą próbkę EKG należy przesunąć 24 bitową otrzymywaną wartość o 6 miejsc w prawo: (sample >> 6). Dzięki temu otrzymana została 18-bitowa liczba.

W kolejnym etapie powinniśmy przekonwertować 18 bitową wartość bez znaku na liczbę ze znakiem, co tworzone jest za pomocą niżej dostępnego fragmentu kodu:

```
if (sample < pow(2, 17) - 1)
{
    return (int)sample;
}
else
{
    int ecg = sample - pow(2, 18);
    return ecg;
}
```

Efektom tych działań jest 18 bitowa wartość ze znakiem, która jest wartością próbki EKG. Wynik w ten sposób otrzymany jest prawidłowy, ale na potrzeby późniejszych obliczeń, wynik ten przeliczany jest na wartość napięcia w mV, za pomocą wzoru widocznego poniżej.

$$V_{\text{ECG}} (\text{mV}) = \text{ADC} \times V_{\text{REF}} / (2^{17} \times \text{ECG_GAIN})$$

gdzie:

- ADC to otrzymana w wyniku wcześniejszych obliczeń wartość EKG,
- V_{REF} otrzymuje wartość 1000mV,
- ECG_GAIN może przyjmować wartości od 20 do 160V/V. W programie przyjęto wartość 40V/V.

Występują drobne różnice w odczycie bitów podczas wysyłania próbek „na żywo” oraz przy odczycie z pamięci urządzenia.

Przykład ramki otrzymanej przy wysyłaniu próbek “na żywo”:

[170, 44, 0, 0, 0, 7, 13, 254, 7, 245, 253, 199, 233, 253, 151, 221, 253, 181, 0, 0].

Przykład ramki otrzymanej podczas odczytu pomiaru z pamięci urządzenia:

[4, 170, 216, 0, 0, 0, 71, 61, 251, 135, 8, 251, 71, 226, 250, 199, 176, 250, 183, 0].

Podczas odczytu na żywo, jako pierwsza otrzymywana jest wartość 170, natomiast przy odczycie danych pierwszą wartością jest 4. W porównaniu do sygnału otrzymywanego od opaski w momencie mierzenia, sygnał odczytywany z pamięci urządzenia posiada taki sam ciąg bitów, jedynie przesuniętych, z powodu zajmowania przez liczbę 4 pierwszej pozycji.

Długość czasu pomiaru trwa 40 sekund, natomiast czas odczytu trwa 9 sekund.

Aby wystartować pomiar należy położyć palec na elektrodzie. Po wykryciu dotyku aplikacja może wysłać na opaskę ciąg komend startujących pomiar:

"set_cfg stream bin",

"set_reg ecg 10 000810C4",

"set_reg ecg 12 00004000",

"set_reg ecg 14 00800000",

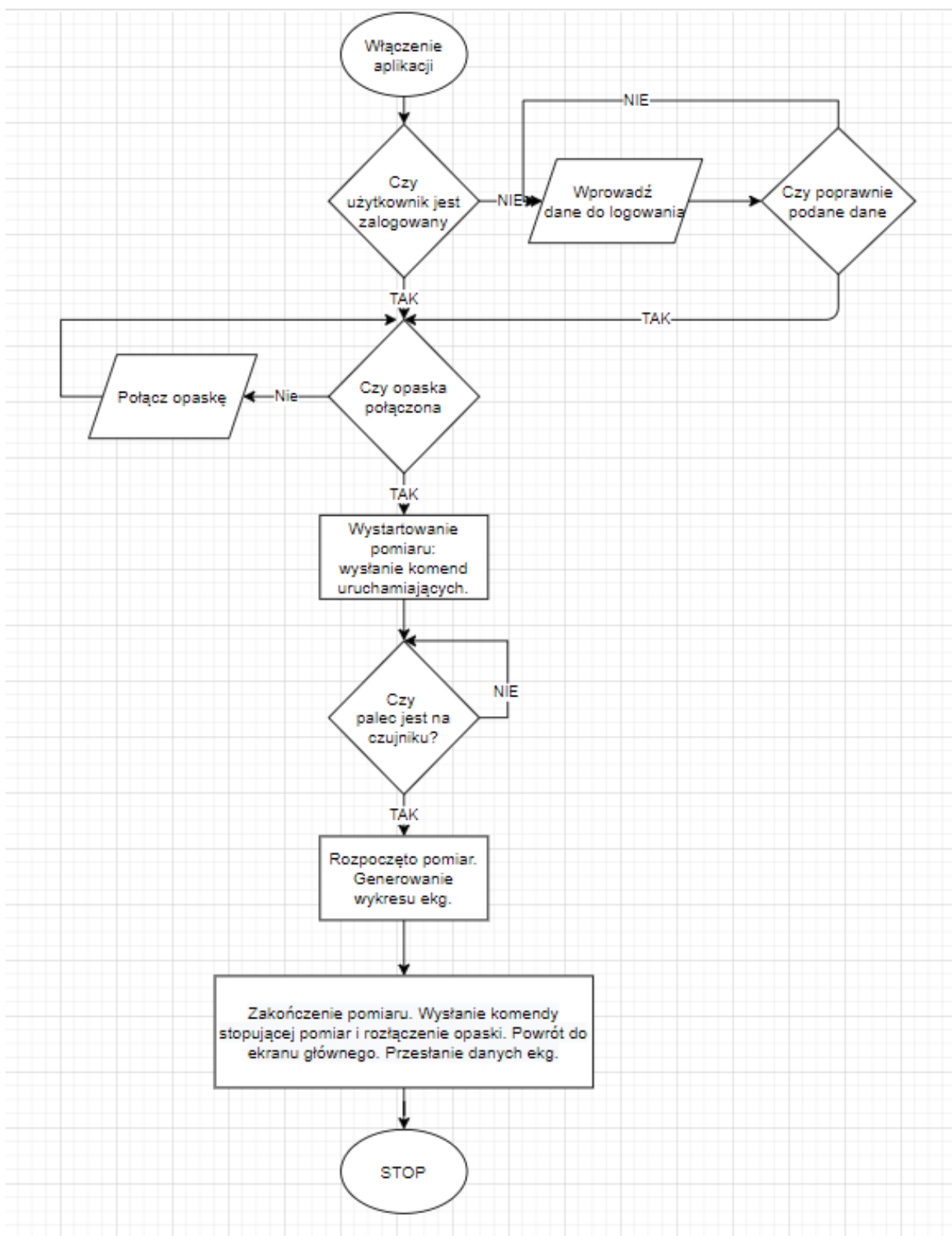
"set_reg ecg 15 00035000",

"set_reg ecg 1d 0035a300".

Komendy te odpowiadają za pomiar sygnału EKG z częstotliwością próbkowania równą 512 Hz. Zostają wysyłane poprzez *writeCharacteristicWithResponse*. Początkowo, wysyłane były charakterystyką, która nie oczekiwała odpowiedzi, jednakże okazała się być nieobsługiwana przez system IOS.

Wystartowanie pomiaru i rozpoczęcie wyświetlania danych na wykresie trwa 10 sekund. Później, przez 40 sekund trwa pomiar, z którego można wyjść poprzez kliknięcie „Zakończ pomiar”. Dane z przerwane go pomiaru nie zostają wysłane na serwer.

Na rysunku zamieszczonym poniżej znajduje się uproszczony schemat współdziałania aplikacji mobilnej i opaski.



Rys. 26. Schemat współdziałania aplikacji mobilnej i opaski.

1.2.7. Badania testowe

Na rysunku poniżej można zobaczyć kolejność danych umieszczonych w tabeli użytkownika.

Pomiary EKG							Pokaż filtrowanie
#	Data	Godzina	Minimalny puls	Średni puls	Maksymalny puls	Status	Wykres

Status określany jest poprzez zastosowanie jednego z czterech symboli, którymi są:

~: oznaczająca zaszumiony sygnał,

A: oznaczające wykrycie arytmii,

N: oznaczające sygnał normalny,

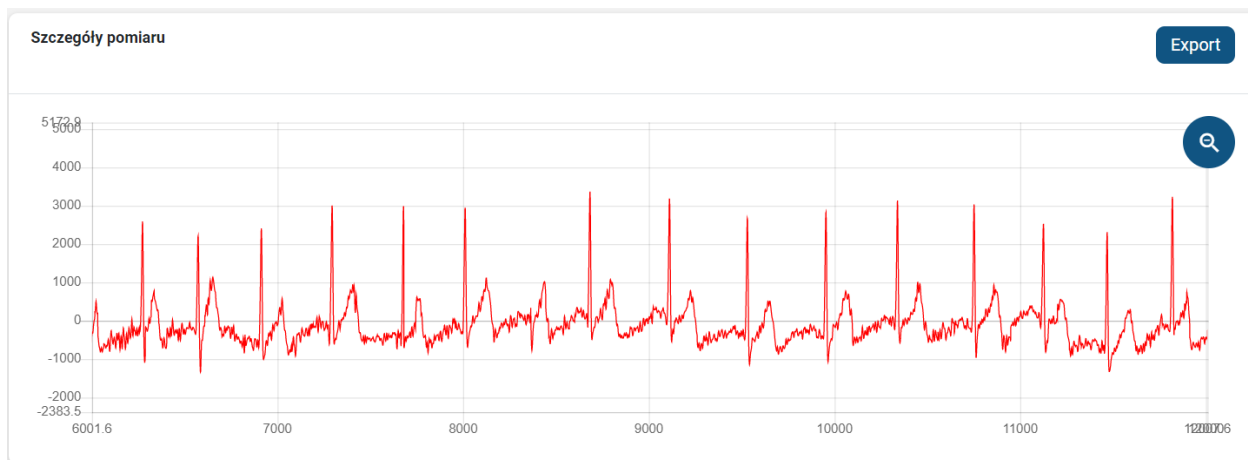
o: któremu odpowiada wykrycie innych chorób serca niż arytmia.

Poniżej znajdują się wyniki pomiaru sygnału EKG kilku losowych osób. Osoby deklarowały, że są zdrowe i wcześniej nie chorowały na żadne choroby kardiologiczne.

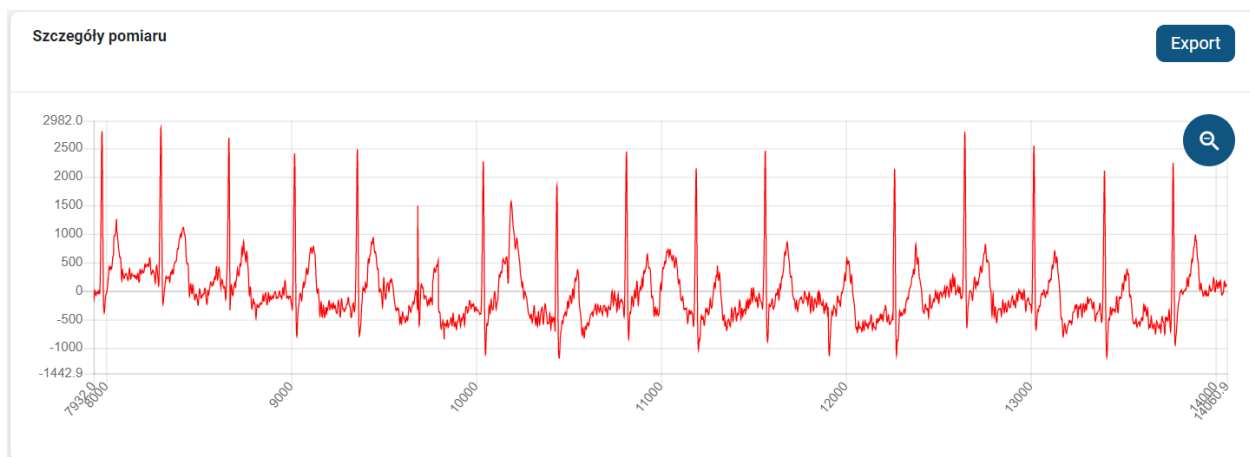
1.2.7.1.R Pacjent nr 1 (JUL)

Pomiary numer: 537, 538

537 19-07-2021 16:05:52 0 83 0 N



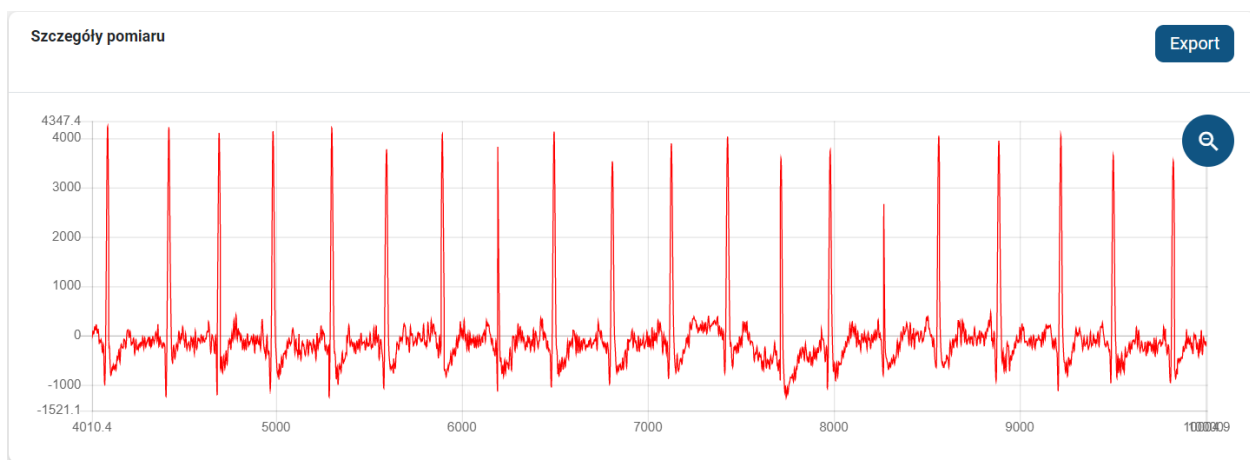
538 19-07-2021 16:06:45 0 84 0 N



1.2.7.2. Pacjent nr 2 (RM)

Pomiary numer: 528

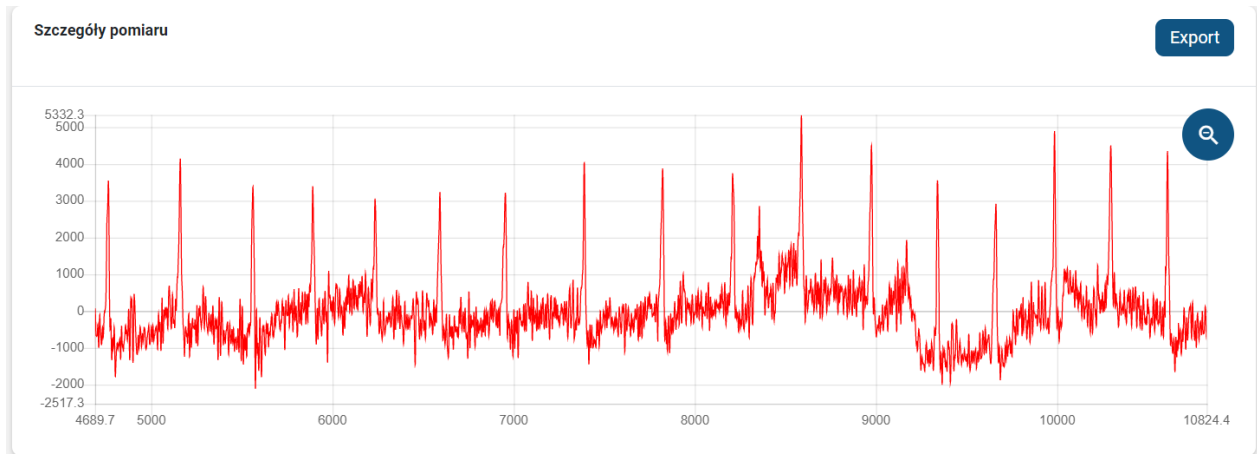
528 19-07-2021 14:28:50 0 102 0 N



1.2.7.3. Pacjent nr 3 (KG)

Pomiary numer: 536

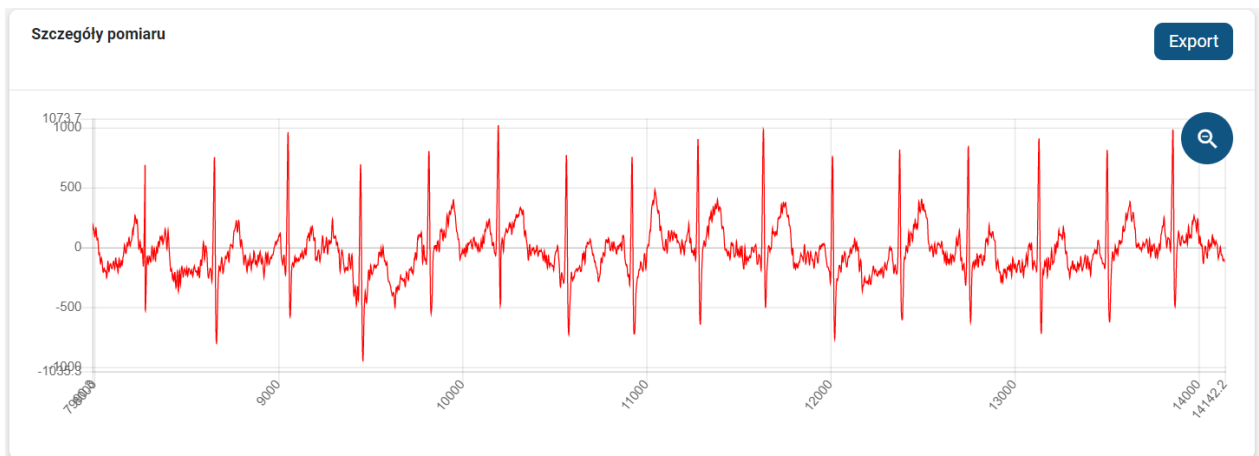
536 19-07-2021 15:58:12 0 82 0



1.2.7.4. Pacjent nr 4 (AT)

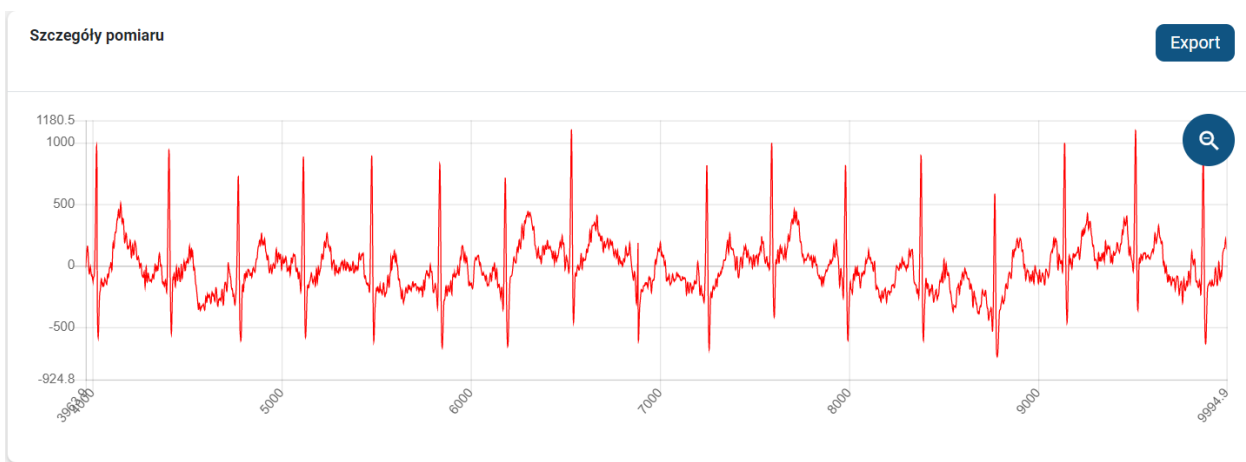
Pomiary numer: 526, 525

526 19-07-2021 13:42:27 0 83 0 N



Pomiary numer: 525

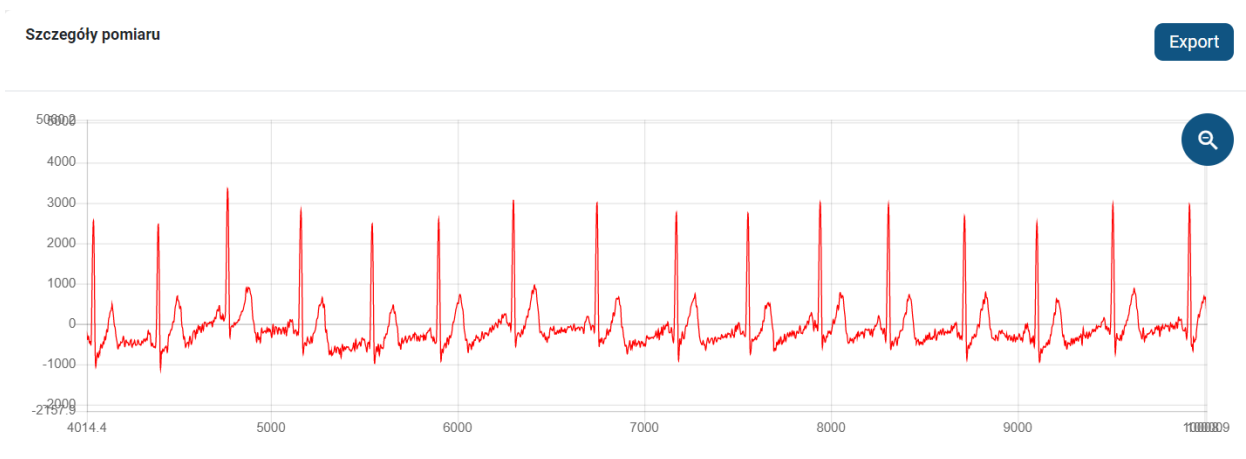
525 19-07-2021 13:41:01 0 84 0 N



1.2.7.5. Pacjent nr 5 (WJO)

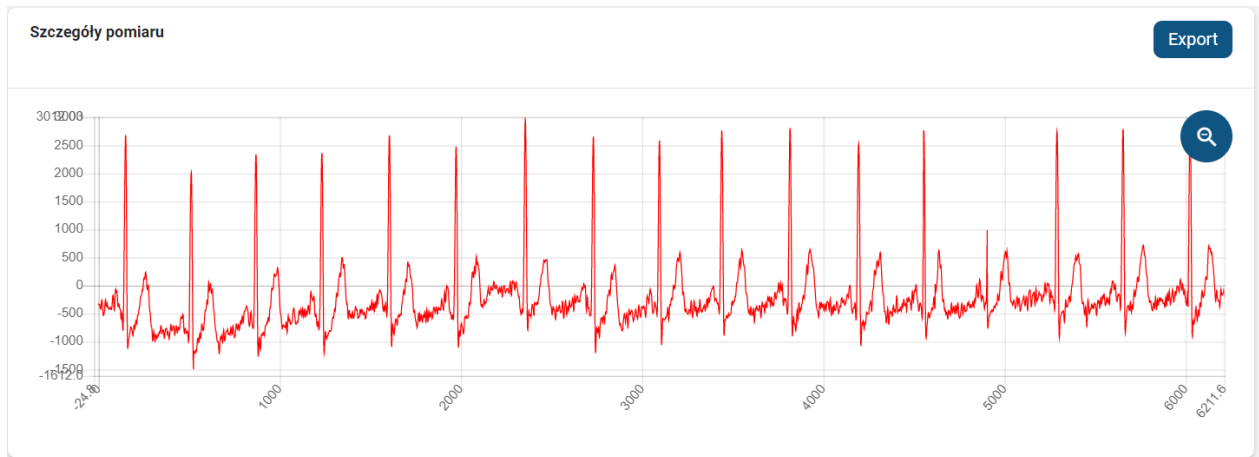
Pomiary numer: 533, 534

533 19-07-2021 14:42:22 0 81 0 N





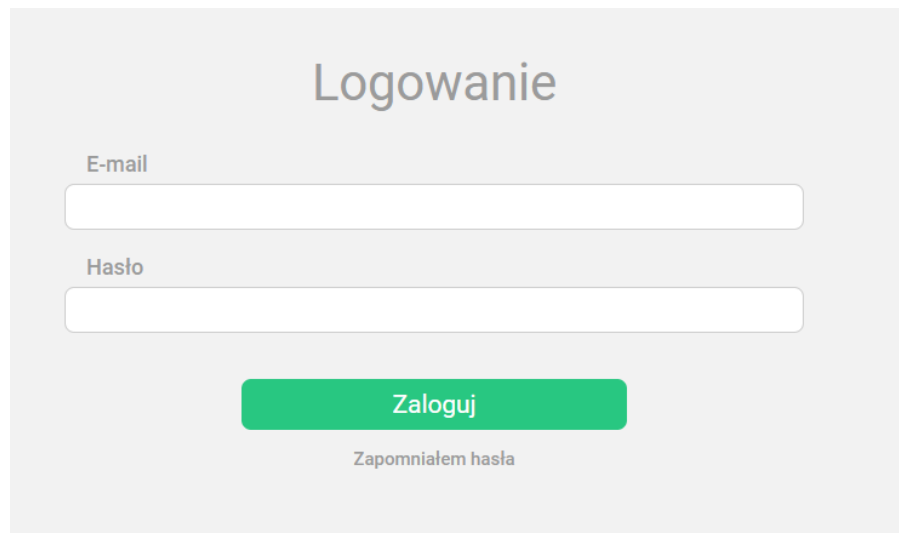
534 19-07-2021 14:43:28 0 84 0 N



1.3. Aplikacja internetowa

1.3.1. Wstęp

Aplikacja internetowa mieści się pod adresem <https://sidly-ekg.pl/> i została dostosowana głównie do opiekuna i managera. Dla użytkownika wskazane jest korzystanie tylko z aplikacji mobilnej. Po wejściu na stronę, widoczne jest okno logowania, w którym należy podać adres e-mail i hasło.



Logowanie

E-mail

Hasło

Zaloguj

Zapomniałem hasła

Rys. 27. Okno logowania do aplikacji internetowej.

Po wejściu w drugą zakładkę, jest możliwość rejestracji konta managera. Aby utworzyć konto, należy podać takie dane jak imię, nazwisko, adres e-mail, hasło managera oraz IMEI opaski.

Rejestracja konta managera

E-mail

Imię

Nazwisko

Hasło

Powtórz hasło

IMEI opaski (0/15)

Nazwa firmy (nieobowiązkowe)

Numer NIP (nieobowiązkowe)

Kod PREMIUM (nieobowiązkowe)

Q Kod premium dostaje się przy zakupie dodatkowego pakietu. Daje dostęp do dodatkowych opcji.

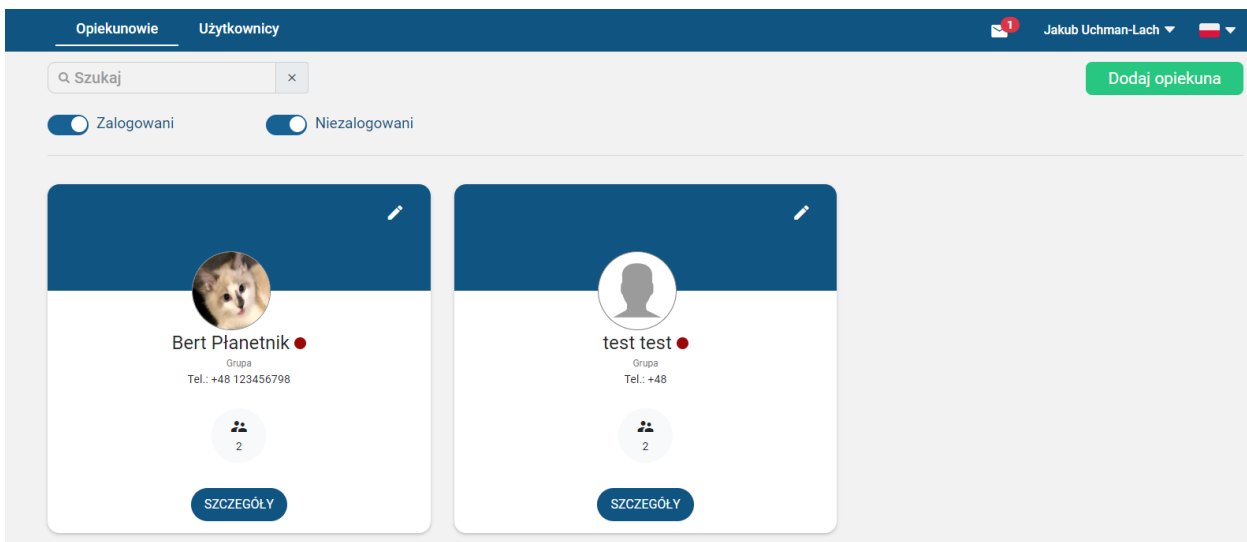
☐ Akceptuję regulamin i politykę prywatności

☐ Zgodnie z art.6 ust.1 lit. a ogólnego rozporządzenia o ochronie danych osobowych z dnia 27 kwietnia 2016 r. (Dz. Urz. UE L 119 z 04.05.2016) wyrażam zgodę na przetwarzanie moich danych osobowych w celach marketingowych przez Sidly Sp. z o. o. z siedzibą w Warszawie;

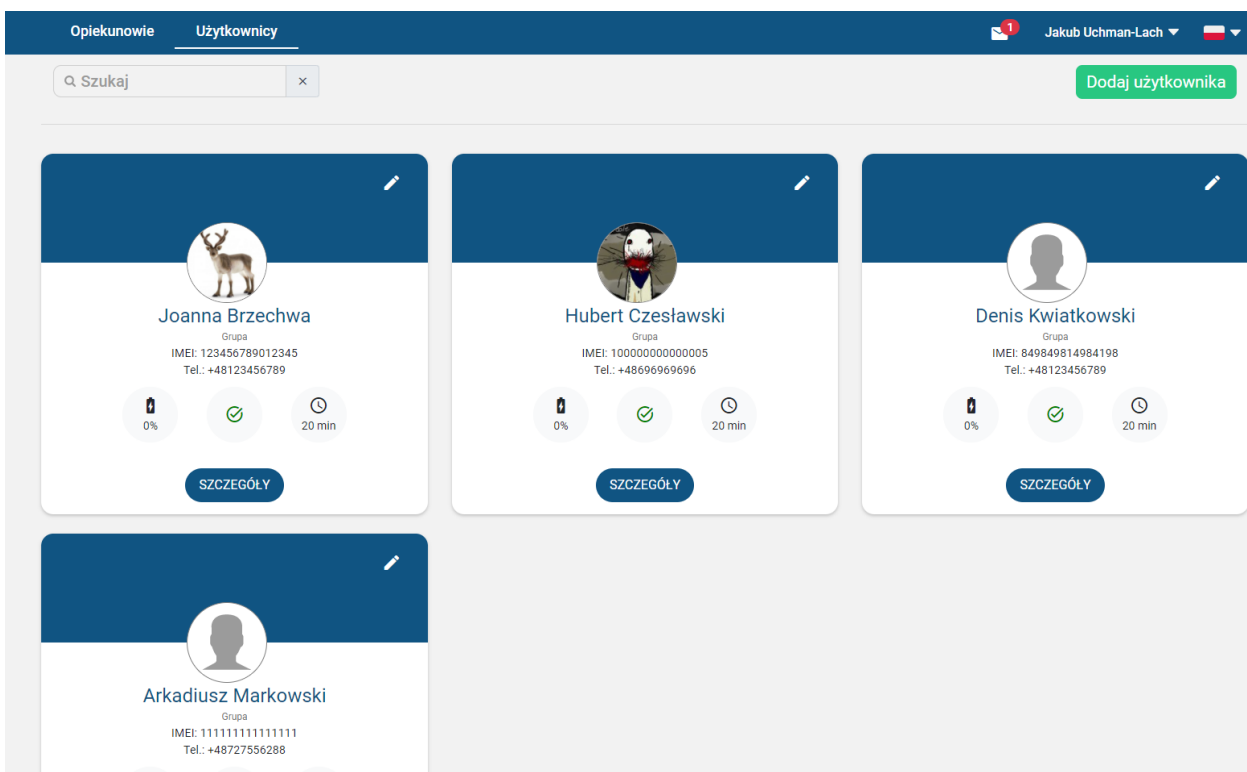
Rys. 28. Okno rejestracji konta managera.

1.3.2. Konto managera i opiekuna

Manager tworzy konta opiekunów i użytkowników. Posiada wgląd do wszystkich ich danych oraz możliwość edycji. Po utworzeniu konta opiekuna, na jego adres e-mail przychodzi link aktywacyjny, w który należy wejść aby ustawić hasło. Opiekun może edytować dane i kontrolować przebieg EKG każdego użytkownika, który został do niego przypisany przez jego managera.



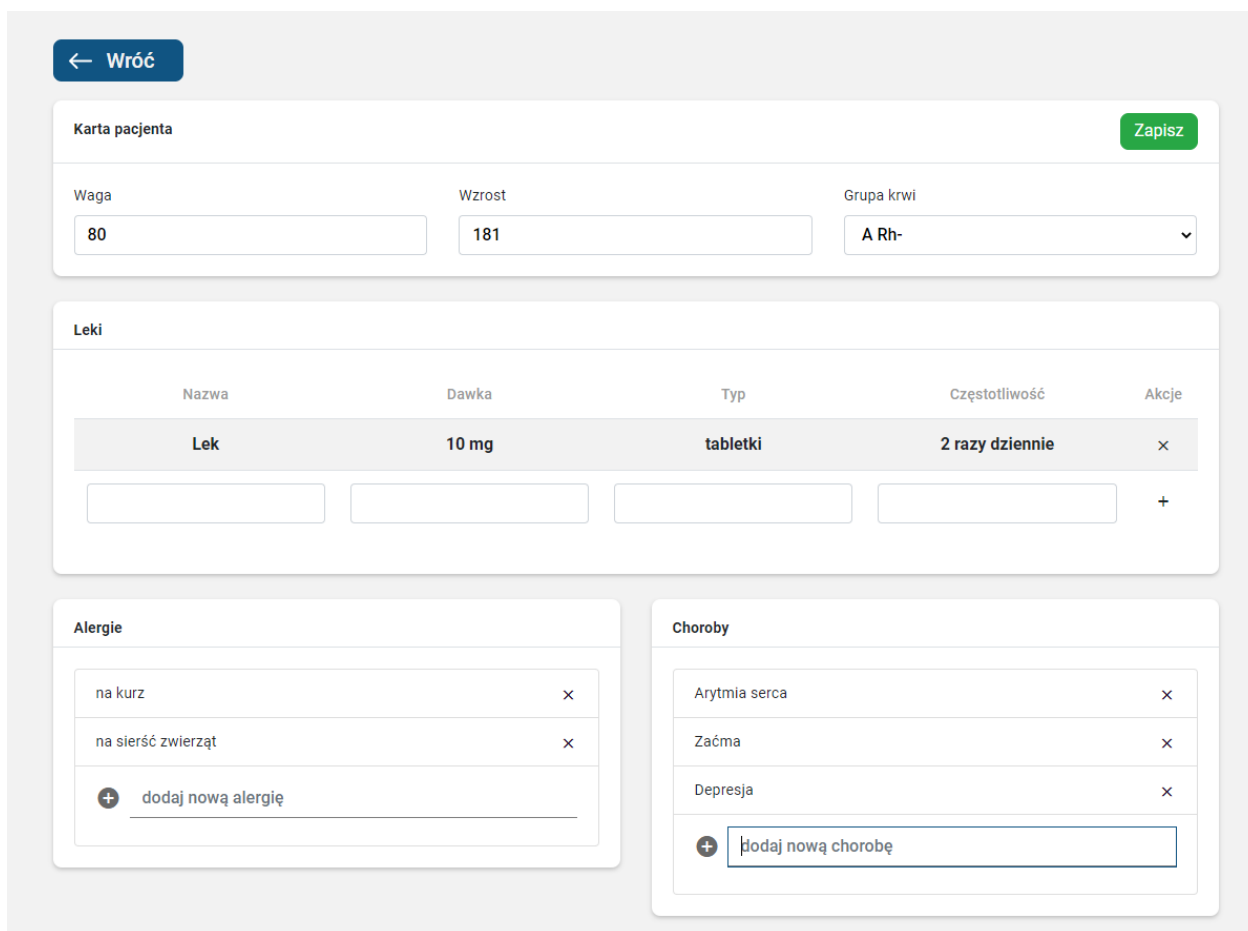
Rys. 29. Widok zakładki "Opiekunowie" z konta managera.



Rys. 30. Widok zakładki "Użytkownicy" z konta managera.

1.3.3. Dane użytkownika w panelu bocznym

Po wejściu w szczegóły użytkownika, z lewej strony ekranu wyświetlają się tabelki wraz z avatarą. W karcie pacjenta znajdują się takie dane jak wzrost, waga i grupa krwi. Po kliknięciu “Zobacz więcej” pojawiają się dodatkowe informacje: leki, alergia i choroby. Wszystkie dane może dodawać i edytować manager oraz opiekun.



Rys. 31 .Widok karty pacjenta

Poniżej widoczne są dane pacjenta (wiek, IMEI opaski) oraz trzecia tabelka z danymi kontaktowymi (telefon, adres i opis). Wszystkie te informacje zaciągane są z okna edycji użytkownika, dlatego można je edytować tylko po wejściu w kafelek “Edytuj użytkownika”.

W czwartej tabelce opiekun lub manager ma możliwość dodania użytkownikowi kontakty do osób ICE (*ang. In case of emergency, pol. w nagłym wypadku*), czyli numery, do których ratownicy powinni zadzwonić w razie nagłego wypadku. Ratownik kontaktuje się z osobą ICE, powiadamia ją o zdarzeniu i w niektórych przypadkach uzyskuje informacje na temat poszkodowanego (jakie leki przyjmuje, alergie, choroby itd.).

Dodając kontakt na platformie należy podać imię, pokrewieństwo, przedrostek oraz numer telefonu.

Dodaj kolejny numer ICE



IMEI

123456789012345

Imię

Pokrewieństwo

Pole jest obowiązkowe.

Pole jest obowiązkowe.

Przedrostek

Numer telefonu

Pole jest obowiązkowe.

Pole jest obowiązkowe.

Anuluj

Dodaj

Rys. 32 .Okno dodawania kontaktu ICE

W ostatniej tabelce manager lub opiekun ma możliwość dodawania dowolnych notatek na temat pacjenta. Przy notatce widoczny jest avatar, imię oraz nazwisko osoby, która ją dodała.

1.3.3. Wiadomości

Na platformie istnieje możliwość komunikacji pomiędzy managerem a opiekunem lub użytkownikiem oraz opiekunem a użytkownikiem, który jest do niego przypisany. Po wejściu w ikonkę wiadomości na środku ekranu pokazuje się czat, a z prawej strony lista rozmówców, z którymi dany manager lub opiekun może rozpocząć rozmowę. Przy każdym rozmówcy widoczna jest kropka oznaczająca status zalogowania rozmówcy (czerwona - wylogowany, zielona - zalogowany).

Przy każdej wysłanej wiadomości pojawia się godzina a większe bloki wiadomości podzielone są na sekcje dzienne. Wiadomości wysyłane są na bieżąco, w czasie rzeczywistym.

Jakub Uchman-Lach

2021-01-27



Jakub

Cześć Berta

12:15

Bert



13:06

hej

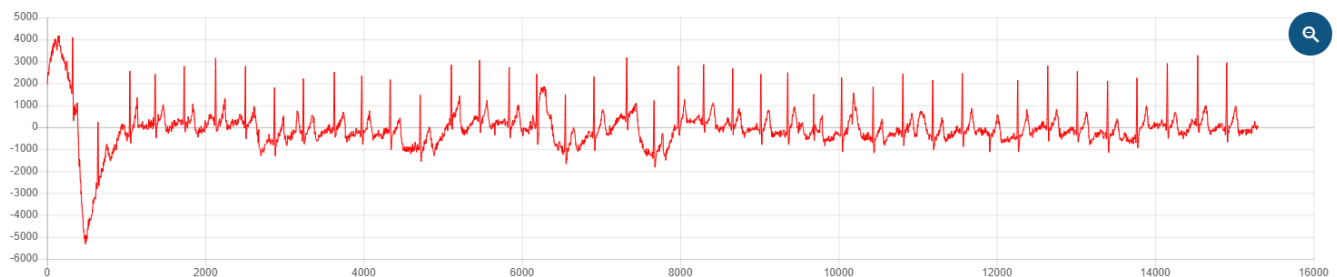
Rys. 33 .Okno konwersacji na platformie

1.3.4. Pomiary EKG

Informacje na temat przebiegu EKG są widoczne po przejściu w szczegóły danego użytkownika. Wszystkie pomiary podane są w tabeli, wraz z takimi informacjami jak: data, godzina, minimalny puls, średni puls, maksymalny puls, status oraz wykres, który wyświetla się po wejściu w jego szczegóły. Istnieje możliwość eksportu każdego wykresu do formatu .json.

Szczegóły pomiaru

Export



Rys. 34. Przykładowy wykres EKG

Pod wykresem znajdują się notatki, które może dodać dany opiekun lub manager oraz wykryte choroby przez opaskę na podstawie przebiegu.

2. OPASKA POMIAROWA

2.1. Architektura oprogramowania

2.1.1. Wstęp

Celem niniejszego etapu było opracowanie i implementacja oprogramowania opaski SiDLY Care EKG pracującej w architekturze systemu operacyjnego czasu rzeczywistego. Głównym zadaniem opracowywanego urządzenia było dokonywanie pomiaru sygnału EKG ze zintegrowanych elektrod umieszczonych na nadgarstku osoby noszącej opaskę, zapis zarejestrowanych danych do pamięci nieulotnej urządzenia oraz transmisja danych protokołem Bluetooth do aplikacji mobilnej z zainstalowaną aplikacją SiDLY Care EKG. Dodatkowymi funkcjonalnościami jakie przewidziano w ramach prac było rozgłaszanie aktualnego poziomu naładowania baterii oraz automatyczne wykrywanie położenia i zdjęcia palca z elektrody EKG.

2.1.2. Architektura

Podczas projektowania oprogramowania urządzenia, w ramach zadania należało zdecydować czy urządzenie będzie pracować w oparciu o system operacyjny czasu rzeczywistego, czy w oparciu o klasyczny kod. W tym celu wykonano porównanie zaprezentowano w poniższej tabeli:

Cecha	RTOS	Klasyczny Bare Metal
Możliwość implementacji w systemach wbudowanych	TAK	TAK
Zdolność do realizacji zadań wielowątkowych	TAK	NIE
Przewidywalny czas działania określonego zadania	TAK	NIE
Efektywne zarządzanie pamięcią	TAK	TAK

Skalowalność	TAK	NIE
PODSUMOWANIE	5/5	2/5

Najważniejszymi elementami biorącymi udział w decyzji o wyborze architektury systemu była zdolność do realizacji zadań wielowątkowych ze względu na jednoczesną potrzebę pomiaru i transmisji sygnału EKG oraz możliwość reagowania na polecenia aplikacji mobilnej. Ponadto bardzo istotna była skalowalność oraz łatwość potencjalnego przenoszenia systemu na inne urządzenia wbudowane dlatego wybrano architekturę opartą o RTOS.

Opracowane oprogramowanie działa w architekturze systemu operacyjnego czasu rzeczywistego RTOS (ang. Real Time Operating System) napisanego w języku C++. Jest to system operacyjny, który został opracowany tak, by spełnić wymagania narzucone na czas wykonywania żądanych operacji. Systemy takie stosuje się jako elementy komputerowych systemów sterowania pracujących w reżimie czasu rzeczywistego. Problemem w tego typu systemach operacyjnych jest algorytm szeregowania oraz podziału czasu. W systemie operacyjnym czasu rzeczywistego trzeba określić, któremu z procesów należy przydzielić procesor oraz na jak długi czas, aby wszystkie wykonywane procesy spełniały zdefiniowane dla nich ograniczenia czasowe. Podobnie jak większość systemów operacyjnych opracowany przez Sidly system jest systemem wielozadaniowym, tzn. umożliwia uruchomienie i wykonywanie wielu zadań w tym samym czasie. W omawianym projekcie – ze względu na niskie zużycie energii wybrano procesor jednordzeniowy, przez co w rzeczywistości, procesor w jednej chwili wykonuje tylko jedno zadanie, ale działa z bardzo dużą szybkością i przełącza się pomiędzy kolejnymi zadaniami na tyle sprawnie, że odnosimy wrażenie jakby wszystko wykonywało się równocześnie. Za przydzielanie czasu procesora poszczególnym zadaniom oraz ich przełączanie odpowiedzialna jest część systemu nazywana planistą (ang. scheduler) – algorytm szeregowania, określający, które zadanie będzie wykonywane jako kolejne. Przełączanie zadań nazywane jest przełączaniem kontekstu (ang. context switching). Opracowany system jest systemem z wywłaszczaniem (ang. preemptive kernel). Oznacza to, że w wyniku wystąpienia jakiegoś zdarzenia, aktualnie realizowane zadanie może zostać wstrzymane, po czym kontrolę przejmuje scheduler i wybiera kolejne.

Komunikacja systemu operacyjnego ze światem zewnętrznym (aplikacją mobilną) zestawiona jest za pomocą komunikacji Bluetooth poprzez moduł RF Bluetooth z serii PAN1326B/1316B firmy Panasonic, który oferuje zarówno łączność Bluetooth Low Energy, jak i Bluetooth. Moduł jest oparty na kontrolerze CC2564B firmy Texas Instruments z certyfikatami Bluetooth, FCC, IC i CE. Standard Bluetooth Low Energy (BLE) ma na celu zmniejszenie zużycia energii poprzez nawiązywanie bardzo szybkich połączeń (kilka ms) i przesyłanie niewielkich ilości danych.

Dzięki tym technikom zużycie energii zmniejsza się do jednej dziesiątej wartości urządzenia Classic Bluetooth. Urządzenia obsługujące technologię Bluetooth Smart mogą komunikować się zarówno z urządzeniami Bluetooth Classic, jak i Bluetooth Low Energy. Urządzenia Smart Ready wykorzystują szybkość transmisji danych Bluetooth Classic (3Mb/s) i ultraszybki czas połączenia Bluetooth Low Energy (3mS). Kryteria wyboru, oraz porównanie ze sobą modułów komunikacyjnych zestawiono w poniższej tabelce.

Cecha	PAN1326B	nRF51822
Moc nadawania	10 dBm	4 dBm
Czułość	-93 dBm	-85 dBm
Częstotliwość taktowania	26 MHz	16 MHz
Zintegrowana antena	TAK	NIE
Zużycie prądu w trybie niskiego zużycia energii	1 μ A	2,6 μ A
PODSUMOWANIE	5/5	2/5

Najważniejszymi parametrami branyymi pod uwagę podczas wyboru modułu było posiadanie zintegrowanej anteny, moc nadawania oraz z racji zasilania baterijnego – pobór prądu. Powyższe kryteria zdecydowały o wyborze układu PAN1326B.

2.1.3. Toolchain

Do opracowania niniejszego projektu wykorzystano toolchain (zintegrowany zestaw narzędzi) GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc) który zawiera zintegrowane i

sprawdzone pakiety zawierające kompilator GCC, biblioteki i inne narzędzia niezbędne do tworzenia oprogramowania bare-metal. Toolchain jest przeznaczony dla urządzeń opartych na 32-bitowych procesorach Arm Cortex-A, Cortex-R i Cortex-M. Zestawy narzędzi są dostępne do kompilacji krzyżowej w systemach operacyjnych Microsoft Windows (x86 32/64bit), Linux (x86_64 i 64-bit Arm) oraz Mac OS X.

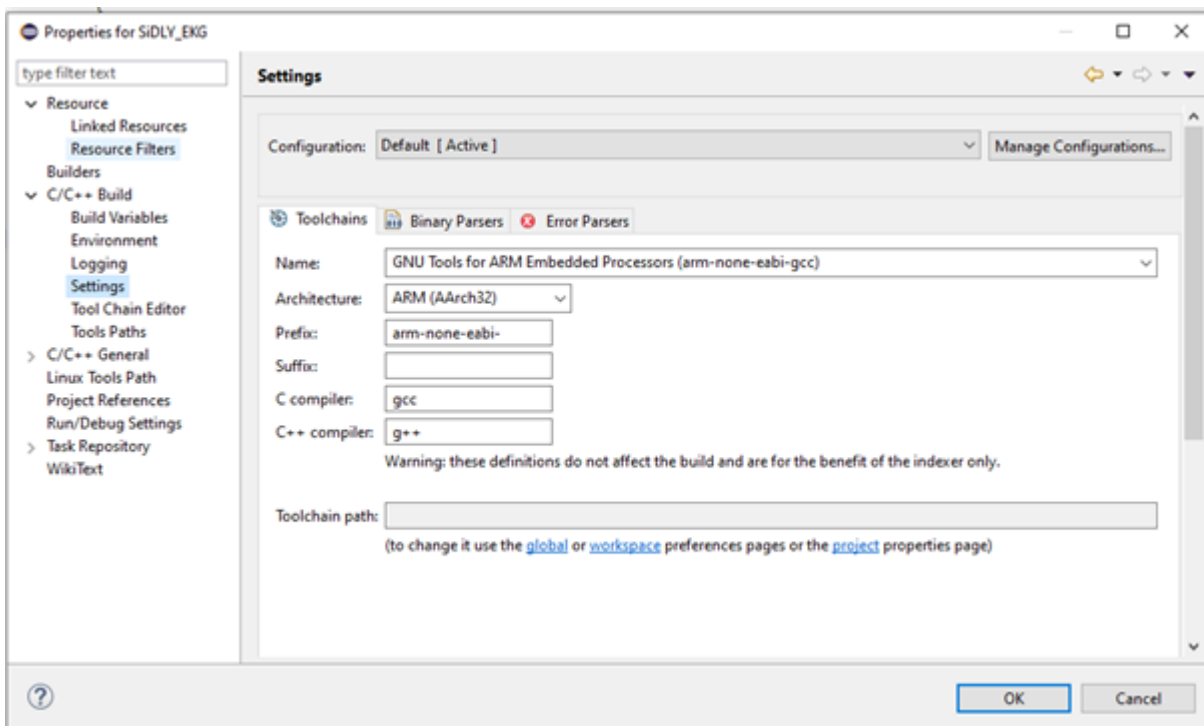
Zastosowany kompilator GCC (ang. GNU Compiler Collection) to tak naprawdę zestaw kompilatorów o otwartym kodzie źródłowym rozwijany w ramach Projektu GNU. Rozpowszechniany jest na licencji GPL oraz LGPL. GCC jest podstawowym kompilatorem w systemach uniksopodobnych, przy czym szczególnie ważną rolę odgrywa w procesie budowy jądra Linuksa. Program gcc (wywoływany podczas kompilacji np. z linii poleceń) odpowiada za przetworzenie argumentów, uruchomienie odpowiedniego kompilatora właściwego dla języka programowania w jakim zakodowano plik z kodem źródłowym, wykonanie programu asemblera dla tak otrzymanego wyniku oraz uruchomienie konsolidatora (linkera) w celu uzyskania pliku wykonywalnego. Przykładowo dla pliku napisanego w C zostaną wykonane następujące programy: preprocesor cpp, kompilator cc1, asembler as oraz konsolidator collect2 (dostępny zazwyczaj jako program ld). Należy przy tym zwrócić uwagę, iż program as wchodzi w skład pakietu oprogramowania binutils. Również pliki nagłówkowe biblioteki standardowej języka C nie są częścią GCC. Kompilator GCC składa się z 3 głównych części: front endu, middle endu oraz back endu. W skład GCC wchodzi kompilatory następujących języków programowania:

- C – gcc
- C++ – g++
- Objective-C – gobjc
- Fortran – g77 oraz nowa implementacja Fortranu 95 o nazwie GFortran
- Java – gcj
- Ada – gnat

a także eksperymentalnie

- Pascal – gpc
- Mercury
- VHDL
- PL/I

Na rys. 2.1.1 pokazano konfigurację toolchain'a w środowisku Eclipse dla niniejszego projektu:



Rys. 2.1.1 Konfiguracja toolchain'a w środowisku Eclipse

2.1.4. Oprogramowanie modułu EKG

Moduł EKG zastosowany w projekcie to MAX30001 – jest to kompletne, biopotencjałowe i bioimpedancyjne (BioZ), analogowe rozwiązanie front-end (AFE) do zastosowań wearable. Oferuje wysoką wydajność w zastosowaniach klinicznych i fitness, przy bardzo niskim poborze mocy, co zapewnia długi czas pracy na bateriach. MAX30001 to układ z pojedynczym kanałem biopotencjalnym zapewniającym odczyt krzywej elektrokardiogramu (EKG) oraz wykrywanie częstotści akcji serca. Konfiguracja urządzenia odbywa się za pomocą zapisu odpowiednich wartości w rejestrach układu. Zapis rejestru odbywa się za pomocą wywołania funkcji przedstawionej na rys. 2.1.2.

```
int MAX30001::max30001_reg_write(MAX30001_REG_map_t addr, uint32_t data) {

    uint8_t result[4];
    uint8_t data_array[4];
    int32_t success = 0;

    data_array[0] = (addr << 1) & 0xff;

    data_array[3] = data & 0xff;
    data_array[2] = (data >> 8) & 0xff;
    data_array[1] = (data >> 16) & 0xff;

    success = SPI_Transmit(&data_array[0], 4, &result[0], 4);

    if (success != 0) {
        return -1;
    } else {
        return 0;
    }
}
```

Rys. 2.1.2 Funkcja zapisująca wartość rejestru do układu MAX30001

Odpowiednią konfigurację układu EKG uzyskuje się poprzez zapis następujących wartości rejestrów:

Lp.	Adres rejestru	Wartość rejestru
1.	0x10	0x000810C4
2.	0x12	0x00004000
3.	0x14	0x00800000
4.	0x15	0x00035000
5.	0x1D	0x0035A300

Wszystkie rejestry układu EKG są 24 bitowe. Dla przykładu zaprezentowano strukturę rejestru o adresie 0x15 (CNFG-ECG) na rys. 2.1.3:

REG	NAME	R/W	23/15/7	22/14/6	21/13/5	20/12/4	19/11/3	18/10/2	17/9/1	16/8/0
0x15	CNFG_ECG	R/W	ECG_RATE[1:0]		x	x	x	x	ECG_GAIN[1:0]	
			x	ECG_DHPF	ECG_DLPF[1:0]		x	x	x	x
			x	x	x	x	x	x	x	x

Rys. 2.1.3 Rejestr 0x15 CNFG_ECG

Wpisanie wartości 0x00035000 skutkuje ustawieniem następujących bitów:

- ECG_RATE – 0x00 (b00) – Częstotliwość próbkowania sygnału EKG – 512 próbek na sekundę (sps);
- ECG_GAIN – 0x03 (b11) – Wzmocnienie kanału EKG 160 V/V;
- ECG_DHPF – 0x01 (b1) – Cyfrowy filtr górnoprzepustowy 0,5 Hz;
- ECG_DLPF – 0x01 (b01) – Cyfrowy filtr dolnoprzepustowy 40 Hz.

Po wyzwoleniu pomiaru EKG, w zależności od trybu pracy przechowywane w pamięci urządzenia bądź wysyłane do aplikacji mobilnej protokołem Bluetooth. Konfiguracja modułu EKG jest możliwa z poziomu aplikacji mobilnej dzięki zaprojektowaniu interpretera poleceń tekstowych. Interpreter odpowiednio analizuje otrzymywane polecenia i wykonuje żądane operacje. Wpisanie odpowiedniej wartości danego rejestru modułu EKG wykonuje się za pomocą przesłania polecenia np.: „set_reg ecg 15 00035000”. Pierwszy fragment polecenia (set_reg) wskazuje interpreterowi, że polecenie zawiera instrukcję ustawiającą odpowiedni rejestr określonego układu. Drugi fragment (ecg) wskazuje układ docelowy, w tym przypadku układ EKG czyli MAX30001. Następnym elementem jest szesnastkowo zapisany adres rejestru, a kolejnym szesnastkowo zapisana wartość rejestru. Pełna konfiguracja opaski otrzymywana przez aplikację mobilną w celu wyzwolenia pomiaru EKG została przedstawiona poniżej: Zastosowanie interpretera poleceń tekstowych pozwala na zachowanie elastyczności i skalowalności rozwiązania. Oprogramowanie opaski nie musi być aktualizowane w przypadku konieczności zmiany konfiguracji urządzenia. Wystarczy, że aplikacja mobilna prześle inną konfigurację, a opaska SiDLY Care EKG dostosuje się do wymagań.

2.1.5. Protokół komunikacyjny

W celu realizacji niniejszego zadania opracowano dedykowany protokół komunikacyjny który zapewnia maksymalną przepustowość danych przy jednoczesnym przesłaniu niezbędnej ilości informacji. Każda ramka danych zawiera elementy wskazane na rys. 2.1.4.

```
typedef struct __attribute__((packed)) {
    uint32_t start_byte      :8;
    uint32_t sample_count    :8;
    uint32_t rtor            :16;    //R to R data - modified to 16 bits
    uint32_t rtor_bpm        :8;
    uint32_t ecg             :24;    //ECG ADC data
    uint32_t ecg_2           :24;    //ECG ADC data 2
    uint32_t ecg_3           :24;    //ECG ADC data 2
    uint32_t ecg_4           :24;    //ECG ADC data 2
    uint8_t  crc8            :8;
} ecg_comm_packet_ble;
```

Rys. 2.1.4 Struktura ramki danych pakietu BLE

Każda ramka składa się z bajtu startu który zawsze przyjmuje wartość 0xAA. Następnym elementem jest bajt zawierający numer kolejnej ramki. Kolejnym elementem są 2 bajty danych zawierające wyznaczony odstęp mierzony w ilości próbek pomiędzy charakterystycznymi załamkami R. W kolejnym bajcie znajduje się przeliczona wartość odstępu na tętno w uderzeniach na minutę (bpm). Następnie znajdują się 4 kolejne 3 bajtowe wartości mierzonego sygnału EKG. Ramkę kończy bajt zawierający sumę kontrolną liczoną algorytmem CRC8. Na rys. 2.1.5. zaprezentowano fragment zadania realizowanego w trakcie wysyłania danych EKG czyli kompletowanie ramki danych.

```
ret = ((MAX30001_Helper*)sensor)->get_sensor_report(sensor_report);
ret |= ((MAX30001_Helper*)sensor)->get_sensor_report(sensor_report_2);
ret |= ((MAX30001_Helper*)sensor)->get_sensor_report(sensor_report_3);
ret |= ((MAX30001_Helper*)sensor)->get_sensor_report(sensor_report_4);

if (ret < 0)
    return 0;
data_packet_ble = (ecg_comm_packet_ble*)buf;
data_packet_ble->start_byte = 0xAA;
data_packet_ble->sample_count = sample_count;
sample_count += m_ecg_ble_packet_count;

data_packet_ble->ecg = sensor_report.ecg;
data_packet_ble->ecg_2 = sensor_report_2.ecg;
data_packet_ble->ecg_3 = sensor_report_3.ecg;
data_packet_ble->ecg_4 = sensor_report_4.ecg;

data_packet_ble->rtor = (sensor_report.rtor | sensor_report_2.rtor | sensor_report_3.rtor | sensor_report_4.rtor);
data_packet_ble->rtor_bpm = (sensor_report.rtor_bpm | sensor_report_2.rtor_bpm | sensor_report_3.rtor_bpm | sensor_report_4.rtor_bpm);
data_packet_ble->crc8 = crc8((uint8_t*)data_packet_ble, sizeof("data_packet_ble") - sizeof(uint8_t));
data_len = sizeof("data_packet_ble");
```

Rys. 2.1.5 Kompletowanie ramki danych kolejnymi próbkami sygnału EKG

Ponadto w celu zabezpieczenia ramki przed modyfikacją w trakcie transmisji zastosowano liczenie sumy kontrolnej algorytmem CRC8, funkcję realizującą ten fragment zaprezentowano na rys. 2.1.6.

```
uint8_t crc8(uint8_t *data, uint32_t length)
{
    uint8_t crc = 0x00;
    uint32_t msg_idx = 0;

    if (length == 0xFFFFFFFF)
        return 0;

    init_crc8();

    while (msg_idx < length) {
        crc = crc8_table[crc ^ data[msg_idx++]];
    }

    return crc;
}
```

Rys. 2.1.6 Funkcja realizująca wyznaczenie sumy kontrolnej CRC8

2.1.6. Warstwa komunikacyjna

Bluetooth Low Energy (Bluetooth LE, potocznie BLE, wcześniej znany jako Bluetooth Smart) to technologia bezprzewodowej sieci osobistej zaprojektowana i sprzedawana przez Bluetooth Special Interest Group (Bluetooth SIG) mająca na celu nowatorskie zastosowania w opiece zdrowotnej, fitnessie, branżach bezpieczeństwa i rozrywki domowej. BLE implementuje w swojej strukturze usługi, cechy i deskryptory które są zbiorczo określane jako atrybuty i identyfikowane przez identyfikatory UUID. Każdy producent może wybrać losowy lub pseudolosowy UUID do użytku zastrzeżonego, ale Bluetooth SIG zarezerwowała szereg UUID (w postaci xxxxxxxx-0000-1000-8000-00805F9B34FB) dla standardowych atrybutów. W celu zapewnienia wydajności identyfikatory te są reprezentowane w protokole jako wartości 16-bitowe lub 32-bitowe, a nie 128-bitowe wymagane dla pełnego UUID. Na przykład usługa przechowująca Informacje o urządzeniu ma krótki kod 0x180A, a nie 0000180A-0000-1000-... . Pełna lista znajduje się w internetowym dokumencie Bluetooth Assigned Numbers.

Opaska SiDLY EKG implementuje standardowe UUID między innymi dla informacji o stanie naładowania baterii, dzięki temu aplikacje mobilne pracujące w oparciu o standard BLE mogą odczytać poziom naładowania baterii urządzenia. Fragment kodu odpowiedzialny za inicjalizację owej charakterystyki został zaprezentowany na rys. 2.1.7:

```
/**
 * Instantiate a battery service.
 *
 * The construction of a BatteryService adds a GATT battery service in @p
 * _ble GattServer and sets the initial charge level of the battery to @p
 * level.
 *
 * @param[in] _ble BLE device which will host the battery service.
 * @param[in] level Initial charge level of the battery. It is a percentage
 * where 0% means that the battery is fully discharged and 100% means that
 * the battery is fully charged.
 */
BatteryService(BLE &_ble, uint8_t level = 100) :
    ble(_ble),
    batteryLevel(level),
    batteryLevelCharacteristic(
        GattCharacteristic::UUID_BATTERY_LEVEL_CHAR,
        &batteryLevel,
        GattCharacteristic::BLE_GATT_CHAR_PROPERTIES_NOTIFY
    )
{
    MBED_ASSERT(level <= 100);
    GattCharacteristic *charTable[] = { &batteryLevelCharacteristic };
    GattService batteryService(
        GattService::UUID_BATTERY_SERVICE,
        charTable,
        sizeof(charTable) / sizeof(GattCharacteristic *)
    );
    ble.gattServer().addService(batteryService);
}
```

Rys. 2.1.7 Fragment kodu odpowiedzialny za inicjalizację domyślnej charakterystyki rozgłaszającej informacje o baterii

Dodatkowo w celu realizacji zadania utworzono dwie dodatkowe, niestandardowe charakterystyki do rozgłaszania informacji o statusie położenia palca na elektrodzie EKG oraz do wysyłania danych EKG. Nie zastosowano domyślnych numerów charakterystyk ze względu na brak przewidzianych takowych w standardzie BLE.

Wysyłanie ramek komunikacyjnych EKG odbywa się za pomocą charakterystyki *CustomServiceUUID->notifyCharUUID*, a fragment kodu realizujący wysyłanie danych zaprezentowano na rys. 2.1.9.

```
int BLE_Icarus_TransferData(uint8_t data_transfer[20]){
    int ret;
    ret = BLE::Instance(BLE::DEFAULT_INSTANCE).gattServer().write(notifyChar.getValueHandle(), data_transfer, 20);
    return ret;
}
```

Rys. 2.1.9 Wysyłanie danych EKG

2.2. Architektura sprzętowa

2.2.1. Wstęp

Głównym celem zespołu SiDLY było opracowanie nowatorskiego urządzenia zdolnego do pomiaru jedno odprowadzeniowego sygnału EKG oraz transmisja tego sygnału do aplikacji mobilnej. Urządzenie musiało spełniać wiele rygorystycznych wymogów, dlatego projekt był niezmiernie trudny i wymagający. Przede wszystkim należało zapewnić minimalny pobór energii przez urządzenie, dlatego wybrano układy charakteryzujące się niskim poborem prądu. Ponadto urządzenie musiało być wygodne w noszeniu, dlatego zdecydowano się na formę popularnego urządzenia wearable czyli zegarka zakładanego na nadgarstek. Urządzenie ma być wygodne w codziennym użytkowaniu dlatego zdecydowano się zapewnić wodoszczelność i pyłoszczelność co dodatkowo narzuciło wymóg zastosowania ładowania indukcyjnego, co również pozytywnie wpływa na wygodę użytkownika

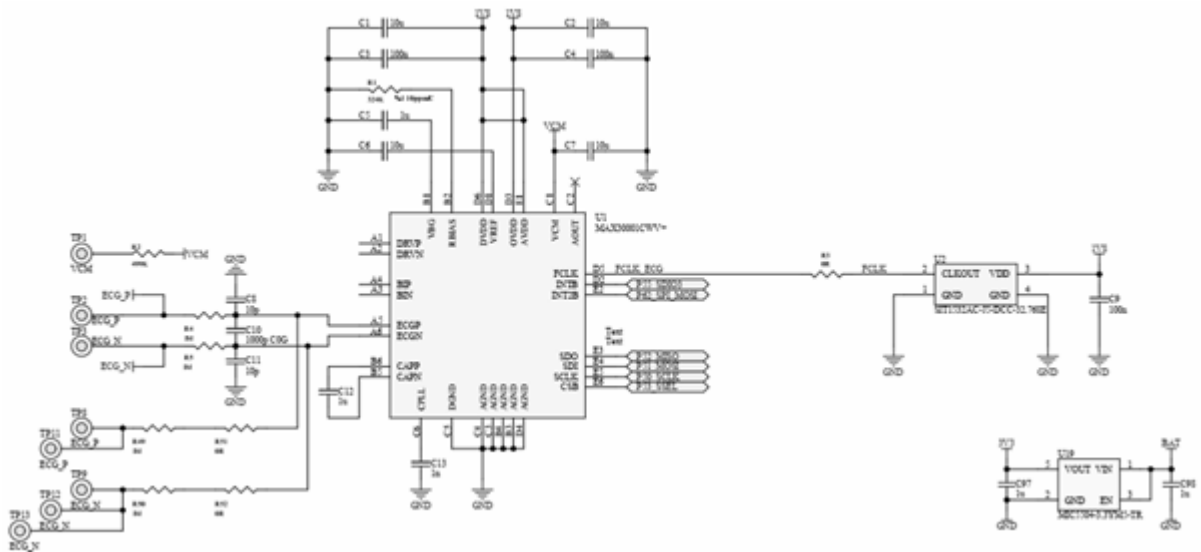
2.2.2. Analog Front-End EKG (AFE)

Moduł do pomiaru EKG został wybrany w oparciu o kilka zasadniczych kryteriów. Docelowo wybranym układem został MAX30001, jest to kompletne, biopotencjałowe i bioimpedancyjne (BioZ), analogowe rozwiązanie front-end (AFE) do zastosowań wearable. Oferuje wysoką wydajność w zastosowaniach klinicznych i fitness, przy bardzo niskim poborze mocy, co zapewnia długi czas pracy na bateriach. MAX30001 to układ z pojedynczym kanałem biopotencjalnym zapewniającym odczyt krzywej elektrokardiogramu (EKG) oraz wykrywanie częstości akcji serca. Wybór docelowego układu został wybrany w oparciu o kryteria zaprezentowane w poniższej tabeli:

Cecha	MAX30001	ADS1292
Ilość kanałów	1	2
Rozdzielczość przetwornika A/C	24	24

Impedancja wejściowa	$> 1 \text{ G}\Omega$	$500 \text{ M}\Omega$
Zużycie energii	$85 \mu\text{W}$	$335 \mu\text{W}/\text{kanał}$
Zużycie prądu w trybie uśpania	$0,6 \mu\text{A}$	$20 \mu\text{A}$
PODSUMOWANIE	5/5	0/5

Konkurencyjnym układem dla MAX30001 jest układ ADS1292 którego producentem jest Texas Instruments. Powyższe zestawienie wskazuje, dlaczego do niniejszego projektu wybrano układ MAX30001, przede wszystkim ze względu na wyższą impedancję wejściową co jest niezmiernie ważne pod kątem bezpieczeństwa osoby noszącej opaskę, oraz ze względu na niższy pobór prądu w trybie uśpienie, co bezpośrednio przedkłada się na dłuższy czas pracy urządzenia na baterii. Na rys. 2.2.1 zaprezentowano schemat podłączenia układu MAX30001:

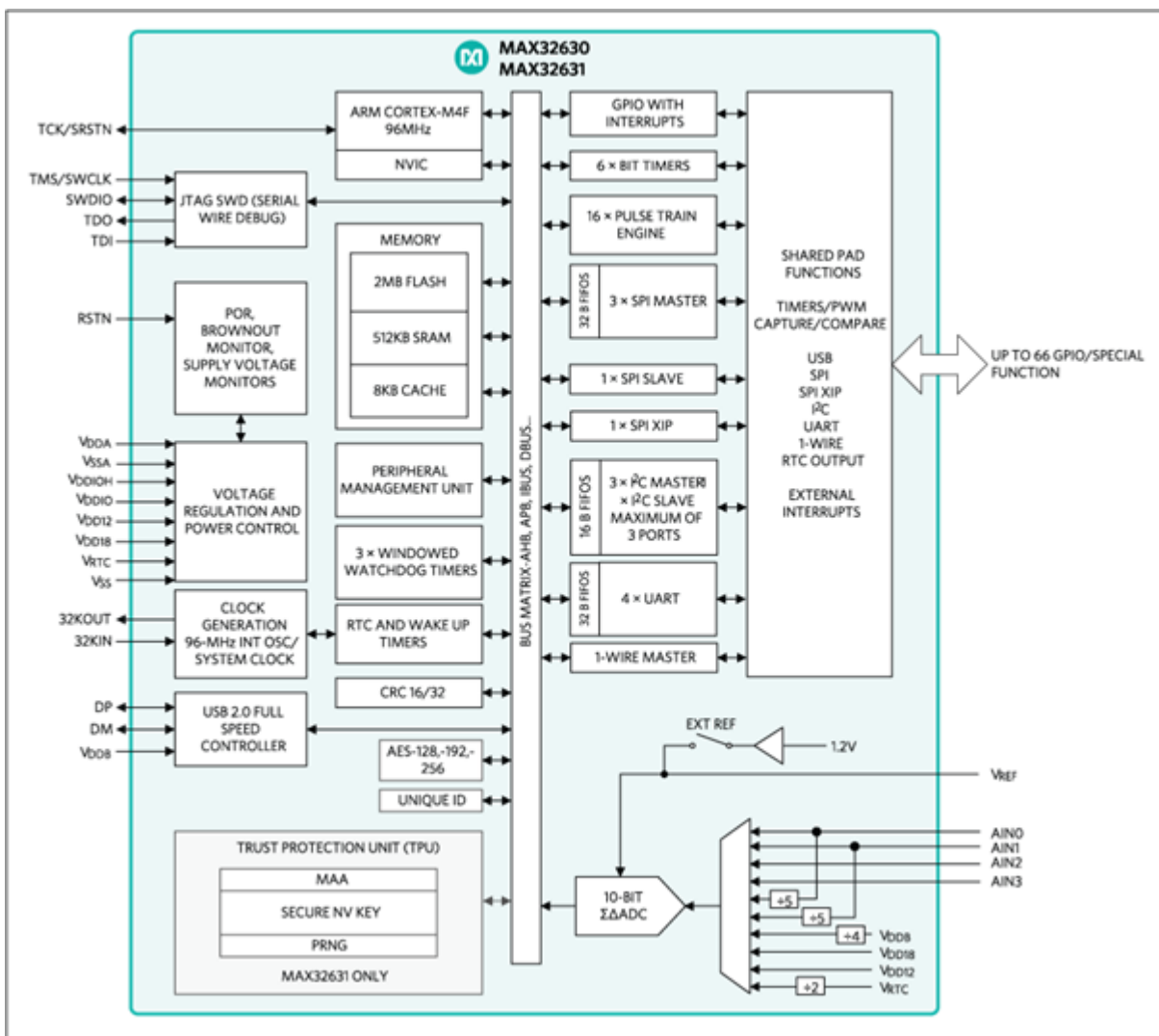


Rys. 2.2.1 Schemat podłączenia układu MAX30001

Na powyższym rysunku pokazano schemat podłączenia układu AFE EKG. Można zauważyć, że układ wymaga niewielkiej ilości elementów dodatkowych w celu jego poprawnego podłączenia, a szybka magistrala SPI pozwala na bezbłędną transmisję dużej ilości danych.

2.2.3. Procesor sygnałowy

Działaniem opaski kieruje główny procesor sygnałowy, którego zadaniem jest komunikacja z układami peryferyjnymi oraz modułami komunikacyjnymi. Procesor sygnałowy odbierając polecenia z aplikacji mobilnej za pośrednictwem modułu komunikacyjnego Bluetooth, steruje diodą LED informując o swoim statusie użytkownika, oraz steruje układem EKG. Sterowanie układem EKG obejmuje jego konfigurację, zapis danych do pamięci, odbiór próbek EKG, ewentualną filtrację danych, oraz wysłanie danych w ustalonej ramce komunikacyjnej. Dodatkowo procesor główny odczytuje poziom naładowania baterii i rozgłasza go za pośrednictwem Bluetooth. Zastosowany procesor sygnałowy to MAX32630. Mikrokontrolery tej generacji są idealne do urządzeń wearable i aplikacji IoT, które nie mogą sobie pozwolić na kompromisy w zakresie mocy lub wydajności. MAX32630 jest wyposażony w procesor Arm® Cortex®-M4 z procesorem FPU, który zapewnia ultraniską moc i wysoką wydajność przetwarzania sygnału przy znacznie zmniejszonym zużyciu energii i łatwości użytkowania. To właśnie jego wysoka wydajność i niskie zużycie energii zdecydowały o jego wyborze. Na rysunku 2.2.2 zaprezentowano schemat blokowy procesora.



Rys. 2.2.2 Schemat blokowy procesora MAX32630

Jak wynika ze schematu blokowego, procesor ten ma szereg układów peryferyjnych do komunikacji z innymi układami, tak jak SPI, I2C, UART, 1-WIRE, USB itp. Rozważaną konkurencją dla tego procesora był układ STM32F401CB. Zestawienie porównawcze zaprezentowano w poniższej tabeli:

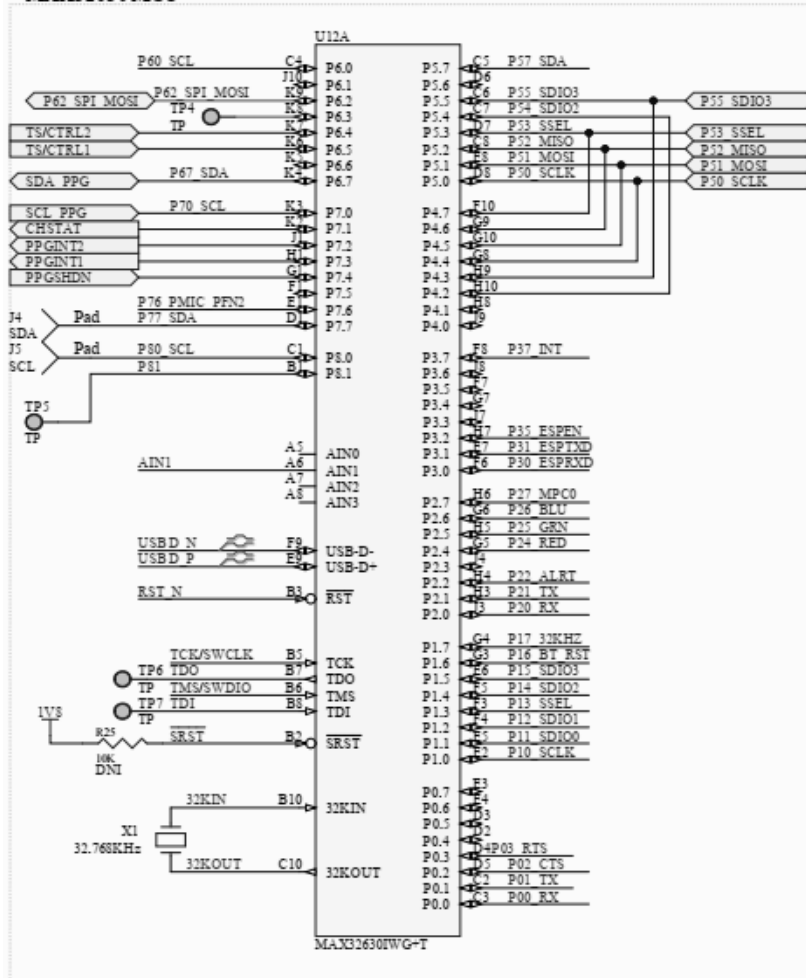
Cecha	MAX32630	STM32F401CB
Pamięć RAM	512 kB	64 kB

Pamięć FLASH	2000 kB	256 kB
Maksymalna częstotliwość taktowania rdzenia	96 MHz	84 MHz
Zużycie energii	106 μ A/MHz	128 μ A/MHz
Zużycie prądu w trybie uśpienia	600 nA	12 μ A
PODSUMOWANIE	5/5	0/5

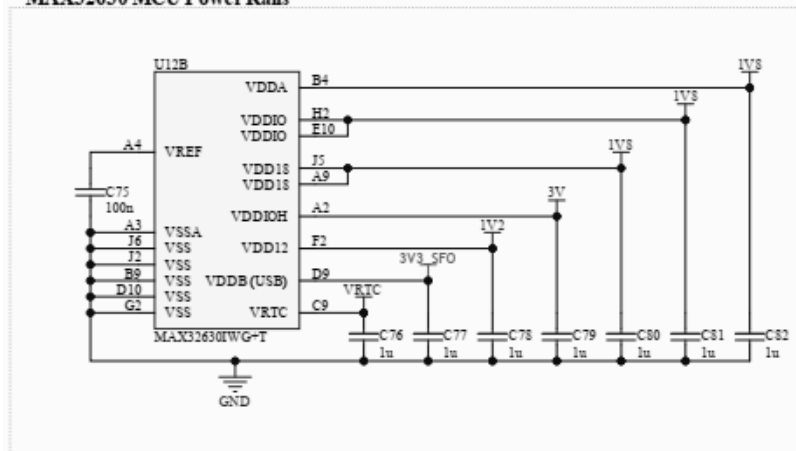
Kluczowe dla wyboru procesora było zużycie energii, z uwagi na stosowanie zasilania bateryjnego oraz wielkość pamięci RAM i FLASH. Na rysunku 2.2.3 zaprezentowano schemat podłączenia procesora sygnałowego.



MAX32630 MCU



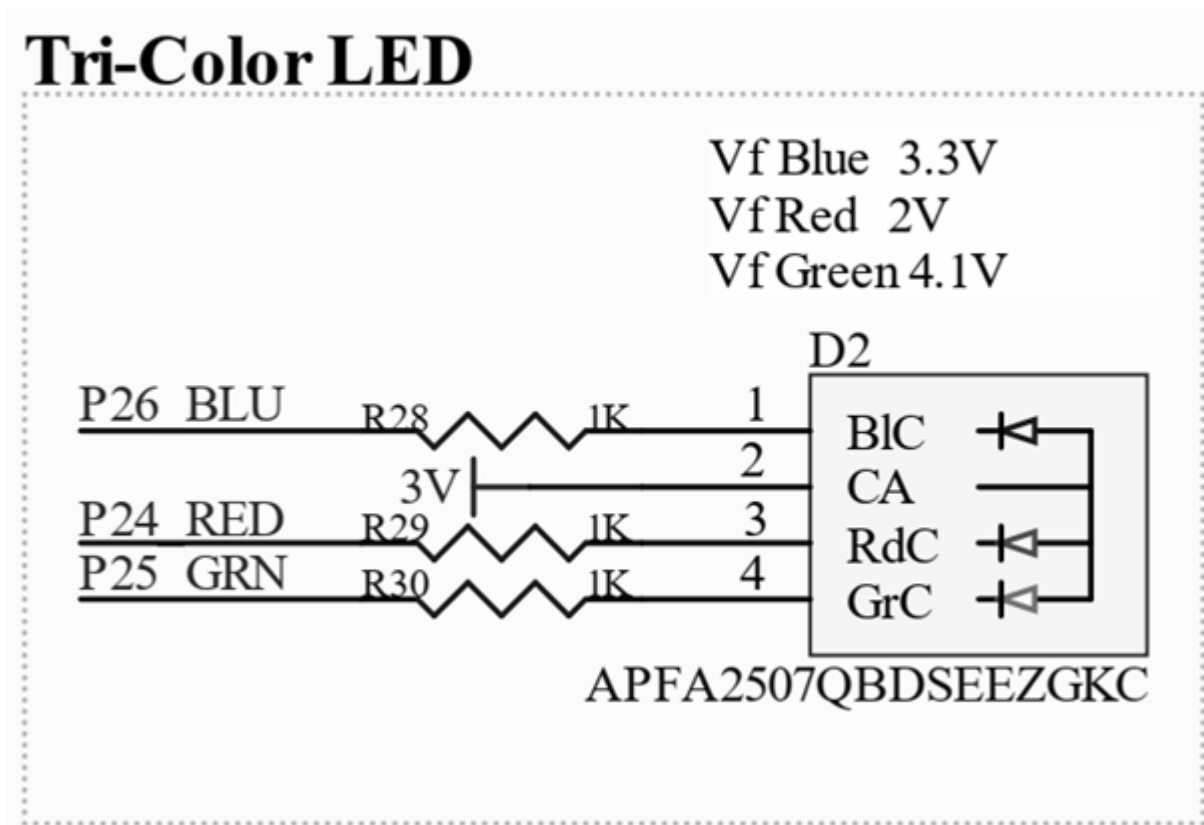
MAX32630 MCU Power Rails



Rys. 2.2.3 Schemat podłączenia procesora sygnałowego

2.2.4. Komunikacja z użytkownikiem

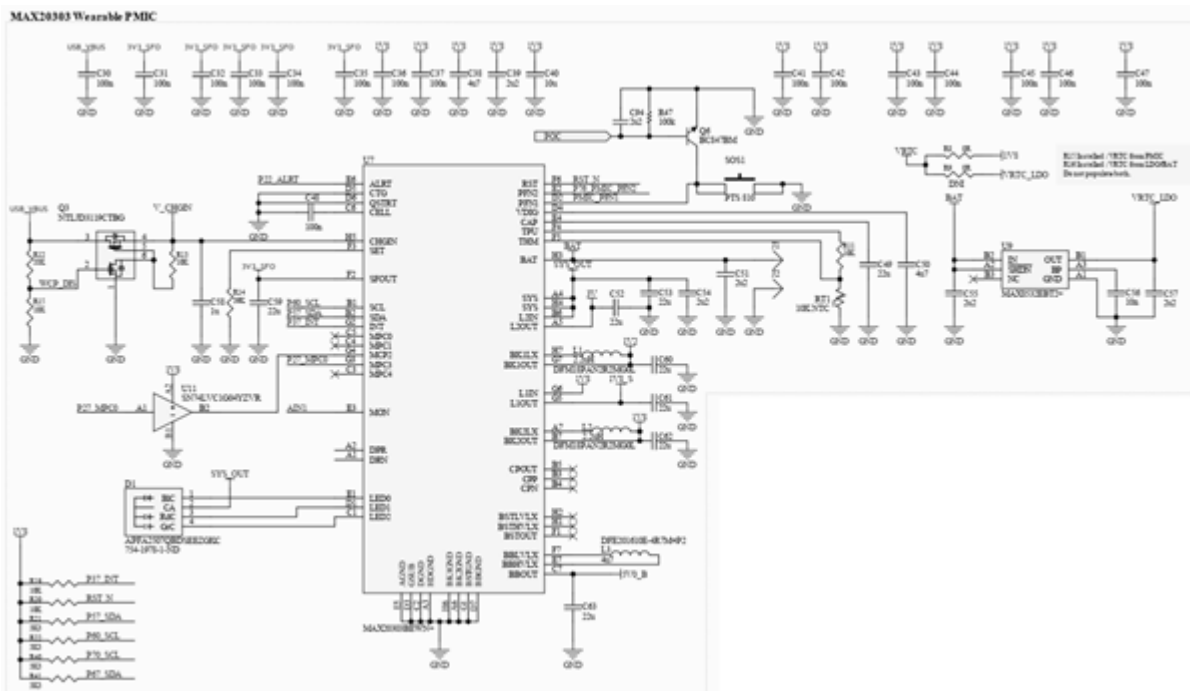
Do komunikacji z użytkownikiem wykorzystano trójkolorową diodę LED oraz przycisk. Dioda została wybrana ze względu na prostotę sterowania oraz montażu. Dzięki zastosowaniu diody RGB urządzenie może sygnalizować stan swojej pracy czy poziom naładowania baterii. Schemat podłączenia diody pokazano na rys. 2.2.4.



Rys. 2.2.4 Schemat podłączenia diody LED

2.2.5. Układ zasilania

Urządzenie jest zasilane z baterii znajdującej się w urządzeniu. Ładowanie jest możliwe za pomocą wodoszczelnego portu USB typu C, bądź za pomocą ładowarki indukcyjnej. Rolę układu ładowania indukcyjnego pełni BQ51050, jest to wysokowydajny, bezprzewodowy odbiornik zasilania zgodny ze standardem Qi ze zintegrowanym kontrolerem ładowania akumulatorów Li-Ion/Li-Pol do zastosowań wearable. Układ BQ51050 zapewnia wydajną konwersję zasilania AC-DC, integruje sterownik cyfrowy wymagany do zgodności z protokołem komunikacyjnym Qi v1.2 oraz zapewnia wszystkie niezbędne algorytmy sterowania potrzebne do wydajnego i bezpiecznego ładowania akumulatorów Li-Ion i Li-Pol. Razem z kontrolerem BQ500212A po stronie nadajnika, BQ51050 tworzy kompletny system bezprzewodowego przesyłu energii do bezpośrednich ładowarek akumulatorów. Wykorzystując indukcyjny transfer mocy w bliskim polu, cewka odbiorcza osadzona w urządzeniu przenośnym może odbierać moc przesyłaną przez

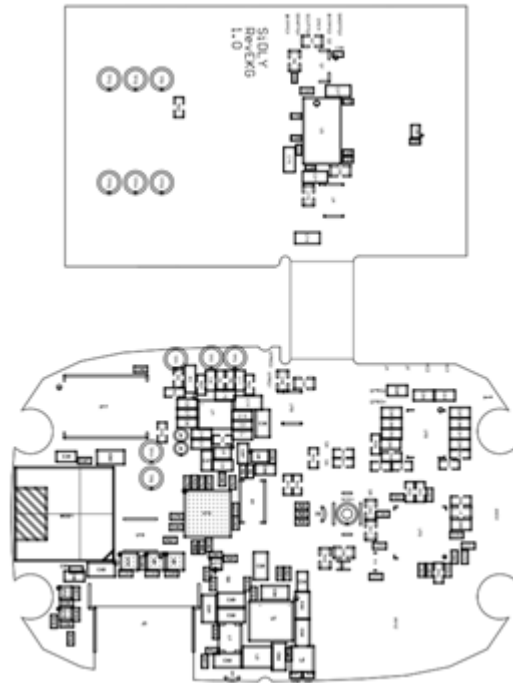
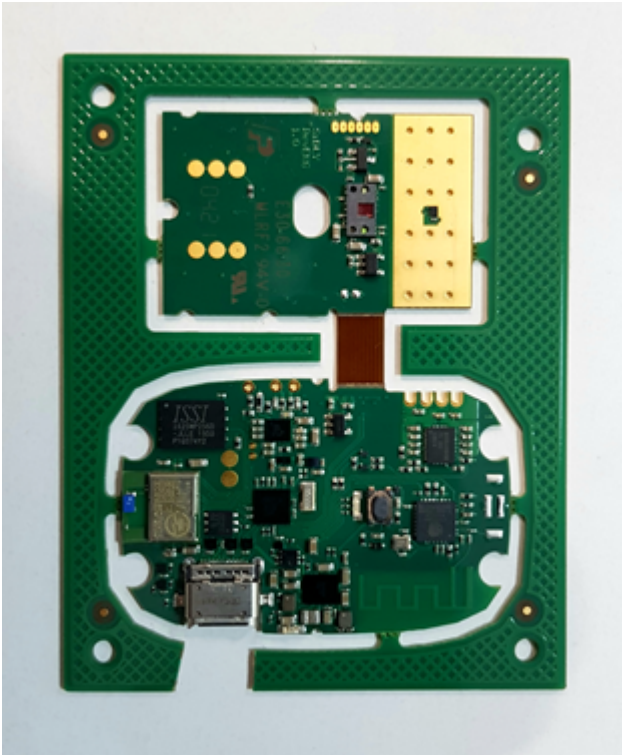


Rys. 2.2.6 Schemat układu zasilania

Za dystrybucję zasilania odpowiada układ MAX20303 jest to wysoce zintegrowane i programowalne rozwiązanie do zarządzania energią przeznaczone do zastosowań noszonych na ciele o bardzo niskim poborze mocy. Jest zoptymalizowany pod kątem rozmiaru i wydajności, aby zwiększyć wartość produktu końcowego poprzez wydłużenie żywotności baterii i zmniejszenie całkowitego rozmiaru rozwiązania. Elastyczny zestaw zoptymalizowanych pod względem mocy regulatorów napięcia, w tym regulatory bucks, boost, buck-boost i liniowe, zapewnia wysoki poziom integracji i możliwość stworzenia w pełni zoptymalizowanej architektury zasilania. Prąd spoczynkowy każdego regulatora jest specjalnie dostosowany do 1µA, aby przedłużyć żywotność baterii w aplikacjach, które są zawsze włączone. MAX20303 zawiera kompletne rozwiązanie do zarządzania akumulatorem z ładowarką, ścieżką zasilania i wskaźnikiem naładowania. W ładowarce wbudowane są zarówno zarządzanie termiczne, jak i ochrona wejścia.

2.2.6. Płytki PCB

Na poniższych rysunkach zaprezentowano zaprojektowany layout płytki, która została wyprodukowana w najnowszej technologii rigid flex.



Rys. 2.2.7 Zdjęcie i layout płytki PCB