

Politechnika Warszawska

W Y D Z I A Ł   M E C H A T R O N I K I



# Praca dyplomowa inżynierska

na kierunku Inżynieria Biomedyczna  
w specjalności Informatyka Biomedyczna

Aplikacja na urządzenia mobilne umożliwiająca pomiar częstotliwości  
tętna z użyciem wbudowanej kamery i oświetlenia

numer pracy według wydziałowej ewidencji prac: 114D-ISP-IB/297497/1284570

**Filip Aleksander Żarnowiec**

numer albumu 297497

promotor

dr hab. inż. Jakub Żmigrodzki

konsultacje

—

WARSZAWA 2023

## PRACA DYPLOMOWA inżynierska

Kierunek studiów: Inżynieria Biomedyczna

Specjalność: Informatyka Biomedyczna

Instytut prowadzący pracę: Instytut Metrologii i Inżynierii Biomedycznej

**Temat pracy: Aplikacja na urządzenia mobilne umożliwiająca pomiar częstotliwości tętna z użyciem wbudowanej kamery i oświetlenia.**

**Zakres pracy:**

Celem pracy jest opracowanie aplikacji działającej na urządzeniu mobilnym z systemem Android, która umożliwia pomiar częstotliwości pulsu za pomocą wbudowanej w urządzenie mobilne kamery fotograficznej oraz oświetlenia. Pomiar częstotliwości tętna będzie zrealizowany poprzez automatyczną analizę zmiany koloru i natężenia światła rozproszonego się w obszarze palca badanej osoby. Zmiany natężenia i koloru wynikają z różnicy między współczynnikami absorpcji światła dla  $HbO_2$  i  $HbCO_2$ . Opracowana aplikacja będzie: dokonywała parametryzacji zarejestrowanego obrazu oraz przetwarzała uzyskany sygnał w celu estymacji średniej wartości częstotliwości pulsu. Wynik pomiaru będzie prezentowany na wyświetlaczu urządzenia mobilnego. Użytkownik będzie mógł zapisać dane pomiarowe w pamięci urządzenia w postaci umożliwiającej ich późniejsze wykorzystanie. Dokładność działania oprogramowania zostanie zweryfikowana poprzez wykonanie referencyjnych pomiarów z użyciem np. elektrokardiografu lub pulsoksymetru.

**Podstawowe wymagania:**

Programowanie w języku Kotlin; Umiejętność wytwarzania oprogramowania; Znajomość technik przetwarzania obrazu; Znajomość technik analizy sygnału;

**Literatura:**

F.Lamonaca, Y. Kurylyak, G. Grimaldi, 2012, "Reliable Pulse Rate Evaluation by Smartphone"; N. V. Hoan, Jin-Hyeok Park, Suk-Hwan Lee, Ki-Ryong Kwon, 2017, "Real-time Heart Rate Measurement based on Photoplethysmography using Android Smartphone Camera";

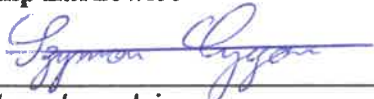
**Słowa kluczowe:** Kotlin, Android, pletyzmografia, estymacja pulsu, analiza sygnału

Praca dyplomowa jest realizowana we współpracy z przemysłem: ~~Tak~~/Nie \*

Nazwa firmy: .....

Imię i nazwisko dyplomanta:

Filip Żarnowiec



Imię i nazwisko promotora:

dr hab. inż. Jakub Żmigrodzki

Imię i nazwisko konsultanta: -

Temat wydano dnia:

23.02.2021

Termin ukończenia pracy:

30.01.2023

**Miejsce wykonywania praktyki przeddyplomowej:** Realeye

### Zatwierdzenie tematu



Opiekun specjalności

DYREKTOR  
INSTYTUTU METROLOGII I INŻYNIERII BIOMEDYCZNEJ  
POLITECHNIKA WARSZAWSKA



dr inż. Elżbieta Ślubska

Z-ca Dyrektora Instytutu

## Streszczenie

Niniejsza praca ma na celu opracowanie aplikacji na smartfona, której zadaniem będzie estymacja częstości skurczów serca użytkownika za pomocą wbudowanej kamery i oświetlenia. Na podstawie obrazów z kamery zostanie dokonana automatyczna analiza zmian intensywności światła rozproszonego w badanej tkance. Metoda użyta do realizacji pomiaru w aplikacji zostanie zweryfikowana w zakresie pomiarowym od 70 do 130 uderzeń serca na minutę. Wyniki pomiaru otrzymane przez aplikację zostaną porównane z wynikami odczytanymi z pulsoksymetru, który będzie urządzeniem referencyjnym.

Fotopletyzmografia jest to metoda pomiaru zmian objętości krwi w naczyniach krwionośnych z wykorzystaniem światła. Polega na pomiarze zmian natężenia emitowanej fali elektromagnetycznej, która ulega rozproszeniu podczas propagacji przez tkankę. W aplikacji rolę sensora pełni kamera w smartfonie, a wbudowane oświetlenie służy jako emiter fali elektromagnetycznej.

Na podstawie przeanalizowanej literatury, powstał wieloetapowy algorytm estymacji średniego tętna użytkownika. Pierwszym etapem jest kalibracja, podczas której określone jest czy palec znalazł się na obiektywie. Podczas 5 sekund rejestrowane przez kamerę obrazy zostają poddane walidacji, która polega na porównaniu średnich wartości intensywności pikseli z arbitralnie dobranymi progami. W następnym kroku przez 25 sekund rejestrowana jest średnia wartość intensywności pikseli dla kanału zielonego obrazu w modelu barw RGB. W ten sposób otrzymywany jest sygnał, który obrazuje wolumetryczną zmianę krwi w naczyniach krwionośnych. Następnie zostaje dokonana filtracja filtrem górnoprzepustowy, w celu dyskryminacji składowych, które nie należą do sygnału użytecznego. Jest to filtr Butterwortha, dwusetnego rzędu, o częstotliwości granicznej równej 1 Hz. W przetworzonym sygnale zostaje przeprowadzona detekcja szczytów. Interwały czasowe między kolejnymi szczytami odpowiadają momentom wyrzutu krwi utlenowanej z serca. Średnia wartość częstotliwości skurczów serca zostaje wyznaczona na podstawie mediany wykrytych interwałów.

Oprogramowanie powstało w języku programowania Kotlin, który jest preferowanym językiem do tworzenia aplikacji na urządzenia z systemem Android. Zgodnie z rekomendacjami zawartymi w dokumentacji Androida została wybrana architektura View-Model-ViewModel, która zakłada rozdzielenie interfejsu użytkownika od logiki działania aplikacji. Warstwa widoku umożliwia użytkownikowi wyświetlenie historycznych pomiarów oraz ich eksport. W modelu zawarta jest implementacja algorytmu oraz klasy wykorzystywane w implementacji. Model Widoku zarządza przepływem danych w aplikacji i asynchronicznie obsługuje zdarzenia w widoku niezależnie od działania głównego wątku.

Skuteczność zaproponowanego algorytmu zweryfikowana została podczas pomiarów porównawczym z pulsoksymetrem. Wykonane zostało 20 pomiarów, gdzie 10 było w stanie spoczynku, 5 po spacerze i 5 po lekkim wysiłku. Średnia różnica, między odczytami uzyskanymi z użyciem urządzenia referencyjnego i testowanego, wyniosła 0,5 bpm i 95% przedział zgodności na wykresie Blanda-Altmana się w zakresie od -6,4 do 5,4 bpm.

Obecna implementacja aplikacji realizuje założenia projektowe oraz umożliwia pomiar średniej wartości częstotliwości skurczów serca. Dokładność pomiaru została określona na podstawie porównania wyniku pomiaru dokonanego przez aplikację z wynikiem pomiaru pulsoksymetru. Aplikacja może znaleźć zastosowanie w sporcie oraz może służyć do użytku rekreacyjnego. Dalszym krokiem w rozwoju aplikacji byłoby umożliwienie pomiaru ciągłego.

**Słowa kluczowe:** Kotlin, Android, pletyzmografia, estymacja pulsu, analiza sygnału

## Abstract

The aim of this work is to develop a smartphone application that estimates the user's heart rate using the built-in camera and lighting. Input image analysis of changes in the intensity of scattered light in the tissue is performed. The measurement method used in the application will be verified in the range of 70 to 130 beats per minute, and the results will be compared with a pulseoximeter.

Photoplethysmography is a method of measuring volumetric changes in blood vessels using light. It involves measuring changes in the intensity of the emitted electromagnetic wave, which is scattered during its passage through the tissue. A camera in the smartphone serves as the sensor, and the built-in lighting serves as the electromagnetic wave emitter.

The developed algorithm for estimating the user's average heart rate consists of several stages. The first stage is calibration, during which it is determined if the finger is placed on the lens. During 5 seconds, a finger position is validated by comparing the average pixel intensity of the image with arbitrarily selected thresholds. In the next step, the average pixel intensity value for the green channel of the RGB colour model is recorded for 25 seconds. This yields a signal that reflects volumetric changes in blood vessels. The signal is then filtered using a high-pass Butterworth filter of the 200th order, with a cut-off frequency of 1 Hz, to discriminate frequency components outside the measurement range. Peak detection is performed on the processed signal, and the time intervals between consecutive heart contraction. The average heart rate is determined based on the median of detected intervals.

The software was developed in the Kotlin programming language, which is a preferred language for creating applications for Android devices. The View-Model-ViewModel architecture was selected in accordance with the documentation's recommendations, which assumes the separation of the user interface from the application's logic. The View layer allows the user to display historical measurements and export them. The Model layer contains the implementation of the algorithm and the classes used in the implementation. The View Model layer manages the flow of data in the application and asynchronously handles events in the view independent of the main thread's operation.

The effectiveness of the proposed algorithm was verified through comparative measurements with a pulse oximeter. 20 measurements were taken, including 10 at rest, 5 after a walk, and 5 after light exercise. The average difference between the readings obtained using the reference device and the tested device was 0.5 bpm and the 95% Bland-Altman agreement ranged from -6.4 to 5.4 bpm.

The current implementation of the application fulfils the project assumptions and allows for measuring the average value of heart rate frequency. The measurement accuracy was determined by comparing the results obtained from the application with those obtained from the pulse oximeter. The application can be used in sports or for recreational purposes. The next step in the development of the application would be to enable continuous measurement.

**Key words:** Kotlin, Android, photoplethysmography, pulse estimation, signal analysis

Wersja w języku polskim

**Politechnika Warszawska**

Warszawa, 6.03.23r  
miejscowość i data

Filip Zaruszaniec.....  
imię i nazwisko studenta

297497.....  
numer albumu

Wydział Mechatroniki, Inżynieria biomedyczna  
wydział kierunek studiów

### OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Filip Zaruszaniec.....  
(podpis studenta)

Wersja w języku polskim

**Politechnika Warszawska**

Warszawa, 6.03.23 r.  
miejscowość i data

Filip Zarwonec  
imię i nazwisko studenta

297497  
numer albumu

Wydział Mechatroniki, Inżynieria biomedyczna  
wydział i kierunek studiów

Oświadczenie studenta w przedmiocie udzielenia licencji  
Politechnice Warszawskiej

Oświadczam, że jako autor/~~współautor~~\* pracy dyplomowej udzielam/~~nie udzielam~~\* Politechnice Warszawskiej nieodpłatnej licencji na niewyłączne, nieograniczone w czasie, umieszczenie pracy dyplomowej w elektronicznych bazach danych oraz udostępnianie pracy dyplomowej w zamkniętym systemie bibliotecznym Politechniki Warszawskiej osobom zainteresowanym.

Licencja na udostępnienie pracy dyplomowej nie obejmuje wyrażenia zgody na wykorzystywanie pracy dyplomowej na żadnym innym polu eksploatacji, w szczególności kopiowania pracy dyplomowej w całości lub w części, utrwalania w innej formie czy zwielokrotniania.

Filip Zarwonec  
(podpis studenta)

\* niepotrzebne skreślić

1. Cel .....	3
2. Zakres pracy i wymagania .....	3
3. Wstęp teoretyczny .....	3
3.1. Fotopletyzmografia .....	3
3.2. Fizyczne podstawy pomiaru.....	4
4. Estymacja wartości średniej częstotliwości skróczów serca .....	5
4.1. Opis algorytmu.....	5
4.2. Implementacja .....	6
4.2.1. Przetwarzanie obrazu.....	6
4.2.2. Przetwarzanie sygnału .....	9
5. Aplikacja .....	11
5.1. Wybrany system operacyjny .....	11
5.2. Architektura.....	12
5.2.1. Single-Activity.....	12
5.2.2. Jednokierunkowy przepływ danych w aplikacji .....	12
5.2.3. Model-View-Viewmodel.....	12
5.3. Interfejs użytkownika.....	16
5.3.1. Ekran początkowy .....	16
5.3.2. Ekran główny.....	17
5.3.3. Ekran historii wyników .....	19
6. Weryfikacja .....	20
6.1. Oprogramowania .....	20
6.2. Działania algorytmu .....	21
6.3. Poprawności szacowania tętna .....	22
6.3.1. Wyniki .....	22
6.3.2. Dyskusja .....	23
7. Podsumowanie .....	25
8. Bibliografia .....	25
9. Wykaz symboli i skrótów .....	27
10. Spis rysunków.....	27
11. Spis tabel.....	28
12. Spis równań.....	28
13. Spis listingów .....	28
14. Załączniki.....	29
14.1. Kod źródłowy.....	29
14.1.1. MierzoPulsApp.kt .....	29

14.1.2.	MainActivity.kt .....	29
14.1.3.	AlgState.kt.....	29
14.1.4.	ImageProcessing.kt.....	29
14.1.5.	SignalProcessing.kt.....	31
14.1.6.	StudyManager.kt .....	31
14.1.7.	Study.kt .....	33
14.1.8.	StudyRepository.kt .....	33
14.1.9.	AppSettings.kt .....	34
14.1.10.	Camera.kt .....	34
14.1.11.	Event.kt .....	34
14.1.12.	HomeViewModel.kt .....	35
14.1.13.	Home.kt.....	36
14.1.14.	History.kt .....	37
14.1.15.	PulseBtn.kt.....	38
14.1.16.	StudyChart.kt .....	39
14.1.17.	InstructionDialog.kt.....	40
14.1.18.	LogoPW.kt.....	40
14.1.19.	ArrowIndicator.kt .....	40
14.1.20.	Checkbox.kt .....	41
14.1.21.	AndroidManifest.xml .....	41



## 1. Cel

Celem pracy jest opracowanie aplikacji na urządzenie mobilne typu smartfon, która umożliwia estymację średniej częstości skurczów serca użytkownika. Pomiar zostanie przeprowadzony z wykorzystaniem wybudowanej kamery i oświetlenia. Pozyskany zbiór obrazów, zostanie automatycznie przeanalizowany w celu określenia zmian natężenia światła, rozproszonego w obszarze palca badanej osoby. Zmiany te są wynikiem rozszerzania i zwężania się naczyń krwionośnych na wskutek zmian ciśnienia krwi. Aplikacja dokona parametryzacji zarejestrowanych obrazów w czasie, a następnie przetworzy uzyskany sygnał w celu wyznaczenia średniego pulsu podczas trwania badania. Wynik pomiaru zostanie zaprezentowany na wyświetlaczu urządzenia mobilnego.

## 2. Zakres pracy i wymagania

Zakres pracy:

- Opracowanie aplikacji mobilnej na system Android
- Implementacja algorytmu obliczającego średnią częstotliwość bicia serca
- Weryfikacja poprawności działania oprogramowania

Wymagania

- Aplikacja wykorzysta wbudowaną w urządzenie kamerę i oświetlenie
- Zostanie opracowany interfejs graficzny umożliwiający przeprowadzenie pomiaru
- Obszarem pomiarowym jest palec w warunkach statycznych
- Zakres pomiarowy urządzenia wynosi od 70 do 130 uderzeń serca na minutę
- Wynik działania algorytmu obliczania tętna zostanie porównany z pulsoksymetrem

## 3. Wstęp teoretyczny

### 3.1. Fotopletyzmografia

Fotopletyzmografia jest to metoda badania przepływu krwi w naczyniach krwionośnych za pomocą światła. Polega na pomiarze zmian natężenia światła, które jest absorbowane lub rozpraszane przechodząc przez skórę i tkanki. W zastosowaniach medycznych wykorzystuje się fale elektromagnetyczne o długościach z zakresu czerwieni i bliskiej podczerwieni. Zmiana natężenia światła przechodzącego przez badany obszar związana jest pulsacyjnym przepływem krwi przez tkankę.

Dzięki analizie sygnału fotopletyzmograficznego (PPG), który obrazuje zmiany rejestrowanego natężenia światła w czasie, można opisać parametry dotyczące przepływu krwi w badanym obszarze, takie jak tętno. W medycynie tą metodę wykorzystuje się w diagnostyce i leczeniu chorób naczyniowych, takich jak miażdżyca, nadciśnienie tętnicze czy choroby zakrzepowo-zatorowe. Poza zastosowaniami medycznymi czujniki fotopletyzmograficzne często stosowane są w rozwiązaniach takich jak opaski sportowe i inteligentne zegarki. Głównymi zaletami tej metody pomiaru jest niski koszt sensorów oraz nieinwazyjny sposób badania.

Przykładem urządzenia medycznego wykorzystującego fotopletyzmografię jest pulsometr. Służy on do pomiaru tętna oraz częściowej saturacji, najczęściej zakładane na palec. Składa się z emiterów światła o odpowiednio dobranych długościach fal oraz fotodetektorów, które mierzą spadek natężenia światła. Z zmian w natężeniu światła powstaje krzywa fotopletyzmograficzna, z której analizy obliczone zostaje tętno. Jest to pomiar ciągły, a obserwacją trendu częstości akcji serca pozwala na określenie parametrów życiowych pacjenta np. podczas operacji.

### 3.2. Fizyczne podstawy pomiaru

Fotopletyzmografia mierzy zmiany objętości krwi w naczyniach krwionośnych w badanym obszarze. Sensor składa się z fotodetektora i emitera światła, o odpowiedniej długości fali, który służy do oświetlania tkanek obwodowych. Pochłanianie światła, przechodzącego przez ośrodek częściowo absorbujący i rozpraszający, mierzonego w fotodetektorze, opisuje prawo Lamberta-Beera (Równanie 1). Prawo to, w ogólnym przypadku, głosi, że absorbancja jest wprost proporcjonalna do stężenia substancji w tkance i do grubości tkanki przez którą przechodzi promieniowanie [8].

$$A = kcl$$

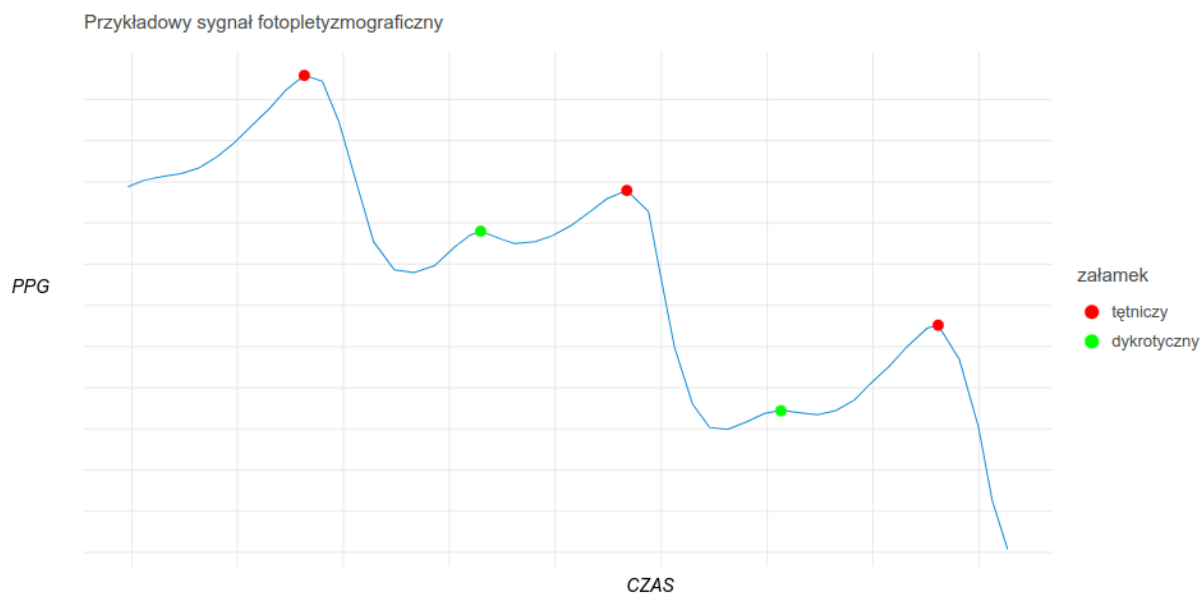
Równanie 1. Prawo Lamberta-Beera

gdzie:

- $A$  – absorbancja
- $k$  – współczynnik pochłaniania promieniowania substancji absorbującej
- $c$  – stężenie substancji absorbującej w warstwie, przez którą przechodzi fala elektromagnetyczna (warstwa absorbująca)
- $l$  – grubość warstwy absorbującej

Sygnał fotopletyzmograficzny jest sygnałem quasi-periodycznym, którego średnia częstotliwość, w warunkach statycznych, fluktuuje w pewnym zakresie. Wahania zależą od czynników, takich jak czynność oddechowa człowieka, działanie układu nerwowego (np. stres) lub aktywność fizyczna.

Składowa pulsacyjna wysokoczęstotliwościowa (powyżej 1 Hz) sygnału fotopletyzmograficznego jest wynikiem zmian objętości przepływ krwi utlenowanej w tętnicach. Mierzona objętość w czasie uzależniona jest od cyklu pracy serca. Charakterystyczny dla tej składowej sygnału jest załamek tętniczy i dykrotyczny [8]. Pierwszy związany jest z wyrzutem krwi utlenowanej do aorty, a drugi z zamknięciem zastawki aortalnej. Na rysunku 1., przedstawiony został przykładowy sygnał PPG zarejestrowany na palcu u człowieka z zaznaczonymi załamekami.



Rysunek 1. Przykładowy sygnał PPG rejestrowany, przez aplikację, na palcu człowieka

Sygnal fotopletyzmograficzny zawiera też składowe wolnozmiennie (poniżej 1 Hz). Składowe spowodowane są wazodylatacją i wazokonstrykcją naczyń, czyli rozkurczu i skurczu mięśni gładkich w ścianie naczyń krwionośnych oraz przepływem krwi żyłnej. W aplikacji, do estymacji średniego tętna wykorzystana zostanie składowa sygnału powyżej 1 Hz, ponieważ związana jest z cyklem pracy serca.

## **4. Estymacja wartości średniej częstotliwości skróczów serca**

### **4.1. Opis algorytmu**

Główną funkcją aplikacji jest estymacja średniej częstotliwości bicia serca użytkownika. W celu realizacji tego zadania został opracowany algorytm inspirowany rozwiązaniem przedstawionym w artykule [4]. Na wejściu algorytm przyjmuje obraz w modelu barw YUV, w którym Y odpowiada za luminancję, a UV kodują barwę. Jest to domyślny format obrazu w systemie operacyjnym Android, w którym pozyskiwane są obrazy z kamery w smartfonie. W powyższym artykule autor zaproponował algorytm bazujący na analizie obrazu w modelu barw RGB, w którym piksele obrazu opisane są za pomocą trzech wartości odpowiadającym trzem kanałom obrazu (czerwonego, zielonego, niebieskiego). W związku z tym przed przetworzeniem obrazu jest on konwertowany do odpowiedniego modelu barw.

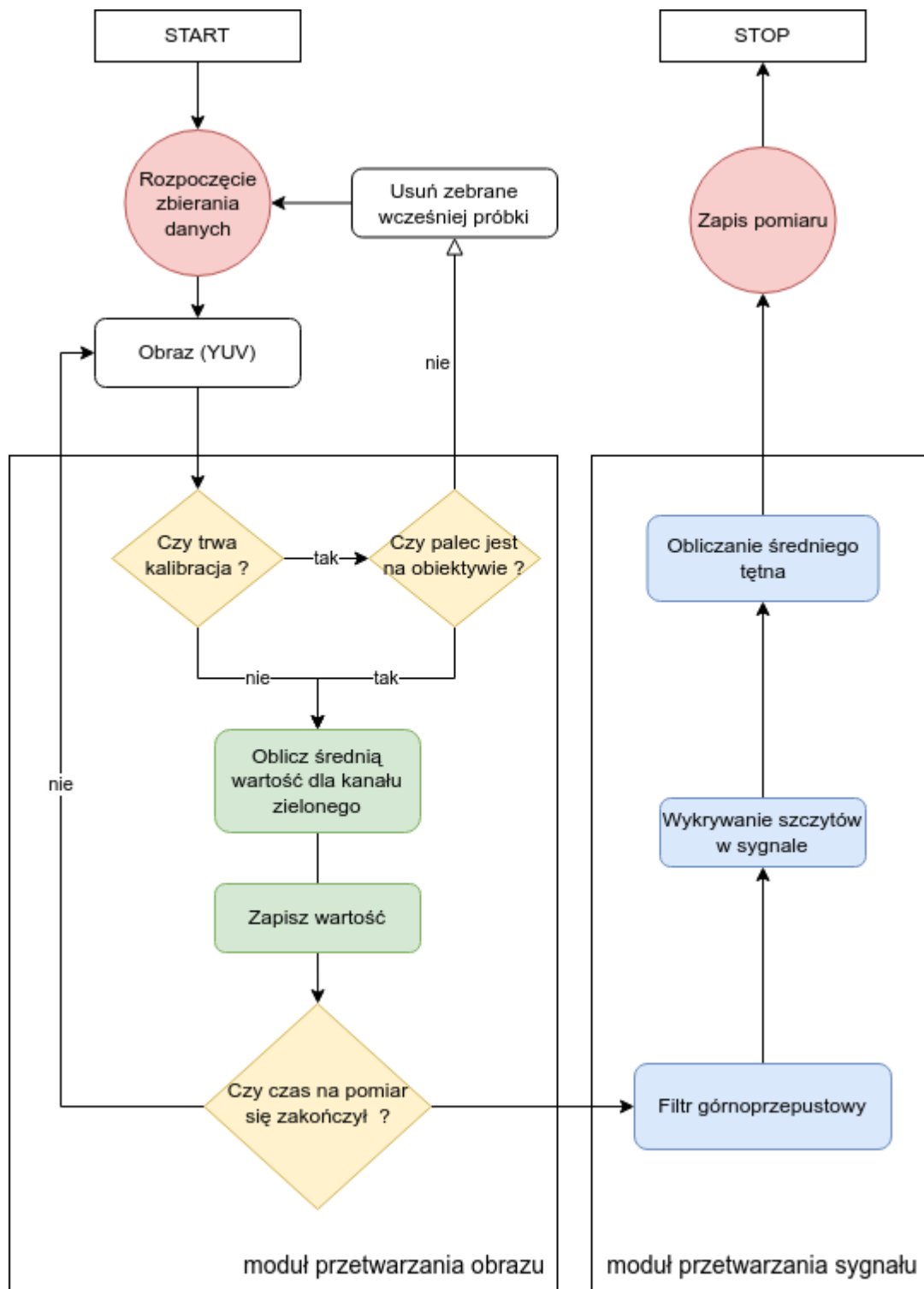
Algorytm jest wieloetapowy i w zależności od etapu, w którym jest algorytm pozyskany obraz zostaje przetworzony w odpowiedni sposób. Możemy wyróżnić etapy:

1. Kalibracja
2. Rejestracja sygnału
3. Obliczanie średniego tętna

Przez pierwsze 5 sekund aplikacja upewnia się, czy palec znajduje się na obiektywie. Jeżeli przez ten czas obszar pomiarowy zostaje pomyślnie walidowany następuje etap rejestracji sygnału. podczas trwania Właściwy pomiaru trwa 25 sekund, podczas których wynik przetworzenia pozyskanego obrazu zostaje zapisywany.

Następnie pozyskany sygnał zostaje podany na wejście filtra górnoprzepustowego, w celu dyskryminacji składowych wolnozmiennych w sygnale. W sposób automatyczny przeprowadzona zostanie detekcja potencjalnych szczytów w sygnale. Momentom wyrzutu krwi utlenowanej z serca odpowiadają interwały czasowe między kolejnymi szczytami. Wynik pomiaru, czyli średnia częstotliwość tętna, zostaje wyznaczona na podstawie analizy statystycznej wykrytych interwałów.

Na schemacie blokowym przedstawiona jest struktura algorytmu (rysunek nr 2, strona 6). Zaznaczony został podział algorytmu na dwa moduły ze względu na funkcję jakie pełnią. Moduł przetwarzania obrazów przyjmuje na wejście obraz, który następnie transformuje na próbkę sygnału. Używany jest podczas etapu kalibracji i rejestracji. Otrzymany sygnał podawany jest na moduł przetwarzania sygnału a następnie zapisywany w pamięci urządzenia. Podział ten ułatwił weryfikację działania poszczególnych modułów oraz samą implementację algorytmu. Omówienie struktury i działania modułów jest opisane w kolejnych rozdziałach.



Rysunek 2 Schemat blokowy działania algorytmu obliczania średniego tętna. Czarnym prostokątem zaznaczony został podział implementacji algorytmu na dwa moduły (przetwarzania obrazów i przetwarzania sygnału).

## 4.2. Implementacja

### 4.2.1. Przetwarzanie obrazu

Moduł przetwarzania obrazów jest odpowiedzialny za transformację obrazu z kamery smartfonu na próbkę sygnału fotopletyzmaograficznego. Podany na wejście modułu obraz zostaje przekonwertowany z formatu YUV na RGB. Następnie obliczona zostaje średnia

wartość pikseli na obrazie dla poszczególnych kanałów. Implementacja opisanych kroków została przedstawiona na listingu 1.

```
fun processImage(image: Image): List<Double> {  
    val mean = Core.mean(image.yuvToRgb())  
    return listOf(  
        mean.`val`[0],  
        mean.`val`[1],  
        mean.`val`[2],  
    )  
}
```

*Listing 1. Funkcja processImage, wejście (zmienna "image") - obraz w formacie YUV, wyjście – lista zawierająca średnie wartości pikseli dla kanałów R, G i B*

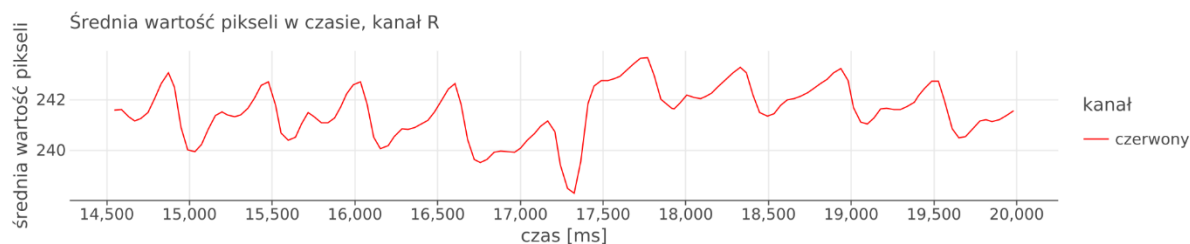
Funkcja *processImage* przyjmuje argument wejściowy w postaci zmiennej *Image*, reprezentującej obraz pozyskany z kamery smartfonu. Po konwersji obrazu (funkcja rozszerzająca *yuvToRgb*), następuje obliczenie średniej wartości pikseli dla poszczególnych kanałów. W tym celu wykorzystana została funkcja z biblioteki OpenCV *Core.mean*, widoczną na listingu 1. Wynik jej działania zostaje opakowywany w listę liczb zmiennoprzecinkowych o długości 3, z których pierwsza jest średnią wartość pikseli dla kanału czerwonego, druga dla kanału zielonego, trzecia dla kanału niebieskiego. Aplikacja podczas rejestracji sygnału wykorzystywana jest jedynie średnią dla kanału zielonego. Podczas kalibracji do oceny czy palec znalazł się na obiektywie wykorzystywane są wszystkie wartości zwracane przez funkcję *processImage*.

### Kalibracja

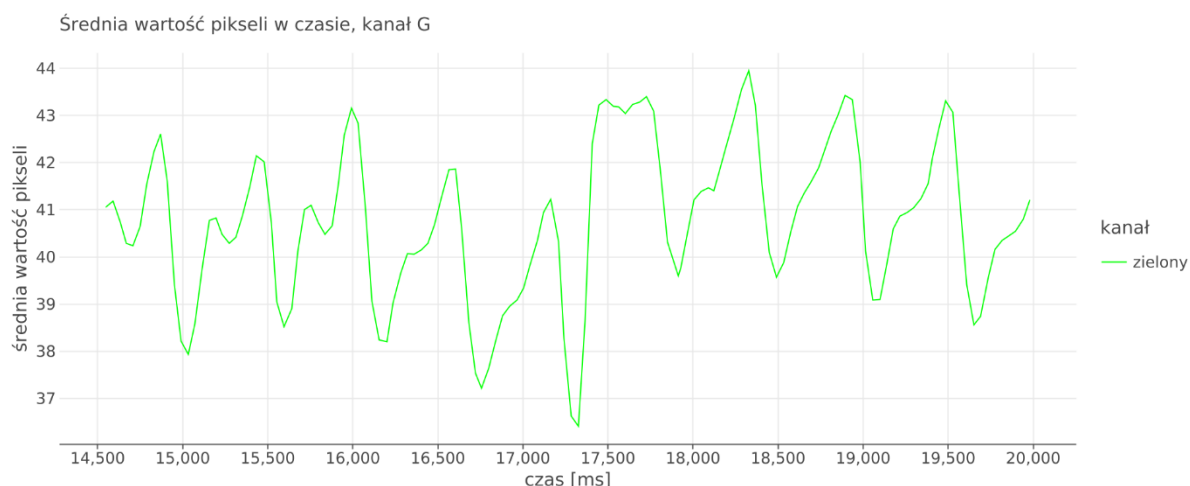
Po rozpoczęciu badania użytkownik proszony jest o położenie palca w taki sposób, żeby zakrywał jednocześnie obiektyw kamery oraz wbudowane oświetlenie. Na etapie kalibracji oprogramowanie weryfikuje, czy palec znalazł się w właściwej pozycji. Ocena odbywa się poprzez sprawdzenie, czy obliczone średnie wartości pikseli dla obrazu zawierają się w arbitralnie dobranych granicach. Z obserwacji wynika, że średnia wartość dla kanału czerwonego powinna być powyżej 170 pikseli, a dla kanałów zielonego i niebieskiego poniżej 100 pikseli. Podobne rozwiązanie zastosowano w pracy [1]. W razie przekroczenia założonych progów kalibracja rozpoczyna się od nowa. Porównanie odbywa się dla każdego obrazu przez 5 sekund. Po tym czasie algorytm zostaje skalibrowany i przechodzi do etapu rejestracji sygnału.

### Wybór źródła sygnału pletyzmograficznego

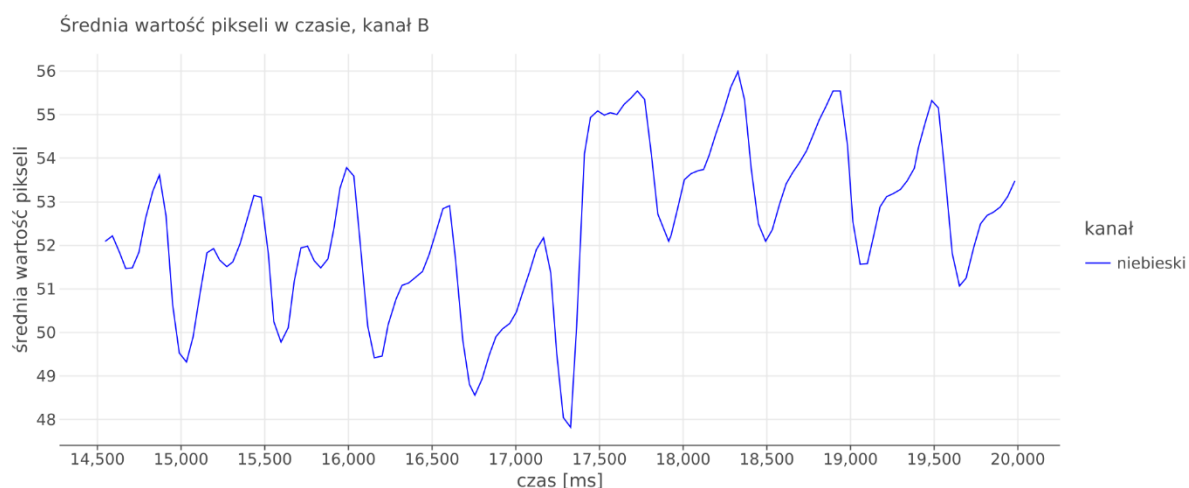
Przeanalizowana literatura nie była zgodna co do wyboru kanału obrazu RGB jako źródło sygnału. W artykułach [3] i [10] pod uwagę podczas rejestracji sygnału był brany tylko kanał czerwony. Natomiast w artykułach [1] i [6] określono kanał zielony jako najsilniejsze źródło sygnału. W związku z tym została przeprowadzona analiza jakościowa mocy sygnału w poszczególnych kanałach. Na rysunkach 3, 4, 5 przedstawione zostały wartości średnie dla kanałów czerwonego, zielonego i niebieskiego w czasie 5 s.



Rysunek 3. Średnia wartość pikseli w czasie dla kanału czerwonego. Na osi X oznaczony jest czas w milisekundach. Na osi y obliczona średnia wartość pikseli w kanale.



Rysunek 4. Średnia wartość pikseli w czasie dla kanału zielonego. Na osi X oznaczony jest czas w milisekundach. Na osi y obliczona średnia wartość pikseli w kanale.



Rysunek 5. Średnia wartość pikseli w czasie dla kanału niebieskiego. Na osi X oznaczony jest czas w milisekundach. Na osi y obliczona średnia wartość pikseli w kanale.

Można zauważyć okresowe wahania wartości średnich, co związane jest z pulsacyjnym przepływem krwi. Zakres wahań wartości średniej dla kanału czerwonego, na analizowanym odcinku czasowym, wynosi 6 pikseli. Dla kanału zielonego i niebieskiego zakres jest szerszy i wynosi ponad 8 pikseli. Na podstawie analizy innych danych literaturowych [8] oraz przeprowadzonych eksperymentów jako źródło sygnału pletyzmograficznego wybrano średnią wartość intensywności pikseli w kanale zielonym przetworzonych obrazów RGB.

#### 4.2.2. Przetwarzanie sygnału

Algorytm kończy rejestrację po 25 sekundach. Zapisane próbki, czyli wartości średniej intensywności pikseli w kanale zielonym pozyskanego zbioru obrazów, podaje się na wejście filtru górnoprzepustowego o częstotliwości granicznej 1Hz. W ten sposób zdyskredytowane są składowe wolnoczęstotliwościowe. Na tak przefiltrowanym sygnale, dokonana zostaje automatyczna detekcja szczytów, szerzej opisana w dalszej części rozdziału. Na podstawie interwałów czasowych między wystąpieniami szczytów w sygnale zostaje obliczona mediana. Do estymacji średniej częstotliwości bicia serca (HR) w bpm został użyty wzór 2. [4], gdzie interwały czasowe (T) podane są w sekundach.

$$HR = \frac{60}{Q_2(T)}$$

Równanie 2. Wzór na średnie tętno [4]

gdzie:

- $HR$  – średnie tętno [bpm]
- $Q_2(T)$  – drugi kwantyl (mediana) obliczonych interwałów między szczytami [s]

#### Filtracja

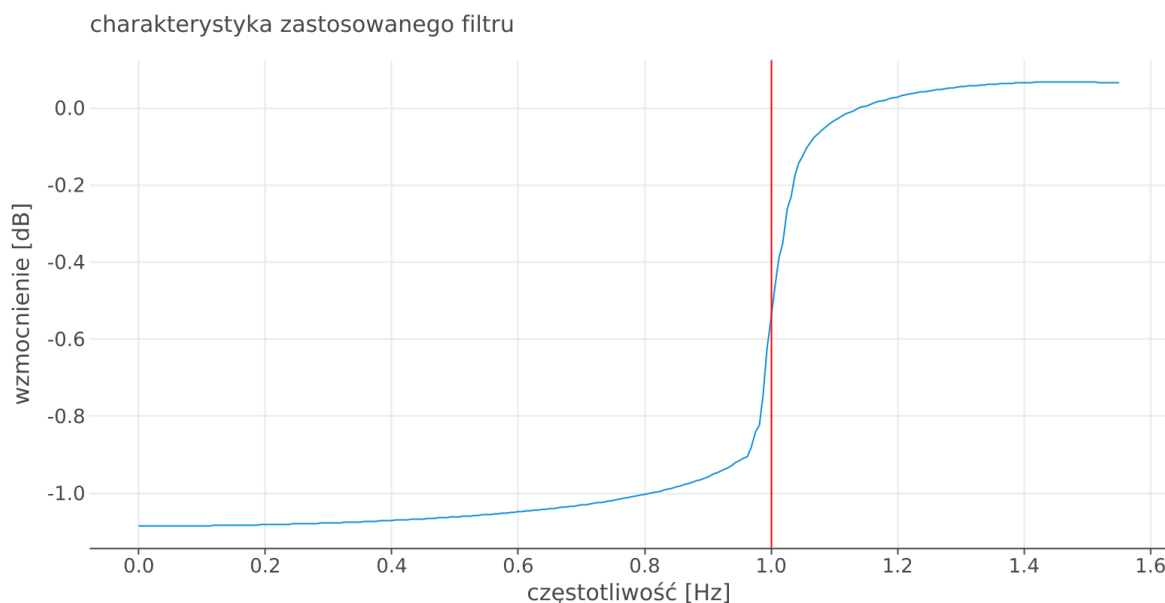
Eksperymentalnie został wytypowany filtr górnoprzepustowy o nieskończonej odpowiedzi impulsowej Butterwortha. Ze względu na to, że filtr jest cyfrowy został zastosowany rząd filtra wynoszący 200. Częstotliwości granicznej ustawiona na 1 Hz, w oparciu o artykuł [10]. Głównym zadaniem tego filtra jest odcięcie składowej stałej oraz składowych niskoczęstotliwościowych, dzięki czemu sygnał po przetworzeniu zawiera tylko te składowe, które zawierają informację o HR. Odpowiednia funkcja realizująca filtrację pokazana została na listingu 3. W celu implementacji filtra została użyta odpowiednia funkcja z biblioteki Java Digital Signal Processing [36] (JDSP) Butterworth.highPassFilter, która jako argumenty wejściowe przyjmuje:

- $fs$  – średnia częstotliwość próbkowania w Hz
- $order$  – rząd filtra
- częstotliwość graniczna

```
fun highpassFilter(  
    rawSignal: DoubleArray, // px  
    times: DoubleArray // s  
): DoubleArray {  
    val timeStart: Double = times.first()  
    val timeEnd: Double = times.last()  
  
    val fs: Double = times.size / (timeEnd - timeStart) // Hz  
    val order = 200  
    val cutOff = 1.0 // Hz  
  
    return Butterworth(fs).highPassFilter(rawSignal, order, cutOff)  
}
```

Listing 2. Funkcja `highpassFilter` realizująca filtrację; przyjmuje na wejście surowy sygnał zapisany z znacznikami czasowymi, na wyjściu zwracany jest sygnał po filtracji.

Znormalizowana charakterystyka częstotliwościowa zastosowanego filtra przedstawiona została na rysunku 6. Zgodnie z założeniami częstotliwości poniżej 1 Hz są tłumione. Filtr Butterwortha charakteryzuje się płaską charakterystyką amplitudową w paśmie przenoszenia. Duże nachylenie charakterystyki wokół częstotliwości granicznej (zaznaczona kolorem czerwonym) jest wynikiem zastosowania wysokiego rzędu filtra.



Rysunek 6. Znormalizowana charakterystyka częstotliwościowa zastosowanego filtru.

Niektóre rozwiązania (np. [8]) estymację częstotliwości tętna bazowały na analizie sygnału w domenie częstotliwości. Z wykorzystaniem o szybkiej transformaty Fouriera określana zostaje częstotliwość, dla której moc sygnału jest największa. Na podstawie tej częstotliwości określony zostaje średni interwał między załamkami tętnicznymi, a następnie estymowana zostaje średnia częstości skurczów serca użytkownika. Innym rozwiązaniem jest wykrycie szczytów w przefiltrowanym sygnale. Mediana z interwałów między załamkami tętnicznymi zostaje użyta do estymacji pulsu za pomocą równania 2. Po porównaniu metod wyniki okazały się bardzo zbliżone. Ostatecznie została wybrana druga metoda, ponieważ okazała się bardziej odporna na ruchy palca podczas badania [5].

### Automatyczna detekcja szczytów

Po filtracji sygnału następuje detekcja szczytów, której implementacja pokazana została na listingu 4. Funkcja przyjmuje dwa argumenty wejściowe. Pierwszy w postaci tablicy wartości typu Double, reprezentujący przefiltrowany sygnał oraz drugi będący częstotliwością próbkowania sygnału.

Do wykrycia próbki sygnału, w których potencjalnie występuje interesujący nas szczyt została użyta dedykowana funkcja z biblioteki JDSP [36] *detectPeaks*. Wykorzystuje ona algorytm znajdowania lokalnych maksimów. Polega on na porównaniu każdego punktu z sąsiadującymi z nim punktami. Jeżeli poprzednia wartość i następna wartość są mniejsze niż wartość rozpatrywanego punktu zostaje on zaklasyfikowany jako szczyt. Wytypowany zbiór punktów następnie zostaje filtrowany pod względem odległości między punktami.



```

fun peakFinder(
    signal: DoubleArray,
    fs: Double // Hz
): List<Int> {
    return FindPeak(signal)
        .detectPeaks() Peak!
        .filterByWidth(
            lower_threshold: fs * 60 / 130 ,
            upper_threshold: fs * 60 / 70
        ) IntArray!
        .toList()
}

```

Listing 3. Funkcja *peakFinder*, znajdująca indeksy wystąpienia szczytów w sygnale.

Na podstawie artykułu [4] warunkiem zaklasyfikowania punktu jako szukanego załamka tętniczego, jest to, żeby odstęp między punktami znajdował się w przedziale opisanym wzorami 3 i 4, przy założeniu, że szczyty mogą występować w zakresie od 70 do 130 bpm. Indeksy znalezionych kandydatów na załamek tętnicze zostają zwrócone na wyjściu funkcji *peakFinder*. Następnie wykorzystywane są do obliczenia średniej częstotliwości bicia serca użytkownika za pomocą równania 2.

$$d_{min} = 60 \cdot \frac{fs}{130}$$

Równanie 3. Wzór, za pomocą którego został oszacowany minimalny możliwy odstęp między wykrytymi szczytami. *F<sub>s</sub>* - częstotliwość próbkowania w przypadku aplikacji 25 klatek na sekundę.

$$d_{max} = 60 \cdot \frac{fs}{70}$$

Równanie 4. Wzór, za pomocą którego został oszacowany maksymalny możliwy odstęp między wykrytymi szczytami. *F<sub>s</sub>* - częstotliwość próbkowania w przypadku aplikacji 25 klatek na sekundę.

## 5. Aplikacja

### 5.1. Wybrany system operacyjny

Aplikacja została zaprojektowana na urządzenia mobilne z system operacyjnym Android. Jest to system operacyjny o otwartym kodzie źródłowym podlegającym wolnej licencji Apache License 2.0. Główną zaletą otwartych licencji jest transparentność i możliwość zweryfikowania działania poszczególnych elementów, co przekłada się na większe bezpieczeństwo i niezawodność działania systemu. Od 2005 roku prężnym rozwojem platformy zajmuje się firma Google. Udostępnia ona narzędzia programistyczne (SDK) oraz zintegrowane środowisko developerskie Android Studio. Wybory architektoniczne w aplikacji podyktowane zostały aktualnymi rekomendacjami zawartymi w dokumentacji Androida [12].

W 2017 roku Google ogłosiło język programowania Kotlin oficjalnym językiem Androida, a od 2019 jego użycie jest rekomendowane w przypadku tworzenia aplikacji na ten system operacyjny. Kotlin jest językiem wieloparadygmatowym (obiektowy, strukturalny, imperatywny oraz posiada elementy programowania funkcyjnego). Powstał w 2011 roku jako alternatywny język działający na wirtualnej maszynie Javy i jest całkowicie interoperatywny z językiem Java.

Duża popularność Kotlina najprawdopodobniej wynika z zaawansowanych funkcji języka takich jak np. funkcje rozszerzeń. Składnia Kotlina jest czytelna oraz charakteryzuje się większą ekspresywnością w stosunku do kodu Javy (mniej redundantnego kodu, określanego po ang. boilerplate code).

## **5.2. Architektura**

### **5.2.1. Single-Activity**

Punktem wejściowym każdej aplikacji napisanej w Androidzie jest klasa „Aktywność” (Activity). Implementuje ona interfejsu dostępu do ekranu użytkownika oraz przechwytuje akcje użytkownika. Posiada ona swój ekran, cykl życia oraz interfejsy potrzebne do sterowania narzędziami jakim jest wbudowana kamera lub oświetleniem. Implementując aplikację posłużyłem się wzorcem pojedynczej Aktywności (z ang. Single-Activity), który polega na stworzeniu aplikacji z wykorzystaniem jednej instancji klasy Aktywności. W alternatywnym podejściu (wielu aktywności) każda Aktywność posiada swój własny widok. Tworzy to pewną redundantność, ponieważ widoki mogą dzielić niektóre elementy. Zaletą wzorca Single-Activity jest również łatwiejsze zarządzanie stanem aplikacji i jej cyklem życia.

### **5.2.2. Jednokierunkowy przepływ danych w aplikacji**

Założeniem architektury jednokierunkowej jest przepływ danych tylko w jednym kierunku przez aplikację. Oznacza to, że dane są przekazywane od korzenia aplikacji do liści, a każdy komponent jest odpowiedzialny tylko za przetwarzanie danych, które otrzymuje. Taki sposób programowania pozwala na lepszą przewidywalność. Interfejs użytkownika, który składa się z mniejszych elementów UI, zwanych dalej komponentami, zgodnie z nomenklaturą stosowaną w użytej bibliotece Jetpack Compose [55]. Umożliwia ona tworzenie elementów interfejsu graficznego jako odpowiednio zanotowanej funkcji, której argumentami są dane, które potrzebne do wyświetlenia interfejsu graficznego. Biblioteka na podstawie zaimplementowanych funkcji tworzy kompozycję, której argumentem wejściowymi jest stan aplikacji. Jetpack Compose zajmuje się obserwacją argumentów funkcji (komponentów) i odpowiednią aktualizacją interfejsu. Stan aplikacji jest propagowany do podrzędnych komponentów zgodnie z dobrą praktyką "pojedynczego źródła wiedzy" (z ang. Single source of truth).

### **5.2.3. Model-View-Viewmodel**

W aplikacji został wykorzystany wzorec projektowy Model-View-Viewmodel (MVVM). Jest to rekomendowana architektura [12] dla aplikacji mobilnych na Androida. Aplikacja podzielona jest na trzy warstwy i każda z nich spełnia określone zadania co zgodne jest z zasadą projektową podziału odpowiedzialności. Zadaniem warstwy widoku jest wyświetlanie i zarządzanie elementami interfejsu użytkownika. Ta warstwa odpowiada jest za interakcje użytkownika z aplikacją. Model zawiera kod implementujący logikę działania algorytmu, klasy i funkcje wykorzystywane podczas jego działania. Warstwa modelu widoku (View-model) zawiera stan aplikacji (taki jak stan algorytmu, zapisane pomiary), który obserwowany jest przez widok. Stan może być modyfikowany zgodnie z logiką zawartą w modelu. Klasa HomeViewModel (załącznik 1.12. HomeViewModel.kt) jest implementacją warstwy pośredniej – modelu widoku, która zarządza przepływem danych w aplikacji.

#### **Widok**

W tworzeniu interfejsu użytkownika został użyty zestaw narzędzi do tworzenia nowoczesnego interfejsu użytkownika - Jetpack Compose. Twórcą biblioteki jest firma Google, która wydała pierwszą wersję w 2019 i od tego czasu zbiór bibliotek ulega ciągłemu rozwojowi. Charakteryzuje się funkcyjną implementacją, polegającą na opisie widoku jako funkcji stanu aplikacji. Inspirowana javascriptową biblioteką React, wydaną i utrzymywaną przez firmę Facebook. Od 2011 roku zdobywa rosnącą popularność jako nowoczesne podejście do budowania reaktywnych interfejsów użytkownika. Główną zaletą używania Jetpack Compose jest szybkie prototypowanie i automatyczna aktualizacja stanu komponentów interfejsu

użytkownika. Założeniem architektury jednokierunkowej jest unikanie stanu wewnętrznego i koncentracja na przepływie danych w jednym kierunku. Programowanie funkcyjne skupia się na definiowaniu funkcji jako czystych transformacji danych bez stanu wewnętrznego, dlatego dobrze wpasowuje się w założenia architektoniczne aplikacji.

Wykorzystane zostały gotowe komponenty (takie jak przyciski, dolna zakładka czy okna modalne) z biblioteki Material UI [22]. Jest to system gotowych komponentów, narzędzi oraz wskazówek dotyczących tworzenia interfejsu użytkownika. Implementuje ona specyfikację projektową Material Design firmy Google. Biblioteka ta zapewnia gotowe do użycia komponenty, których wygląd jest łatwy do dostosowania. Dzięki jej zastosowaniu można odseparować część wyglądu aplikacji od samych komponentów. Ustawiając odpowiednie kolory przewodnie, domyślnie każdy komponent domyślnie będzie ich używał, co znacznie przyspiesza implementację, jak również spełnia obecne dobre praktyki dotyczące tworzenia interfejsu użytkownika.

## Model

Do warstwy modelu należy logika działania aplikacji, w tym implementacja opracowanego algorytmu obliczania pulsu. Model składa się również z klas, które są odpowiedzialne za pobieranie, przetwarzanie i przechowywanie danych. Na listingu 5. Przedstawiona została klasa *Study*, która przechowuje dane dotyczące przeprowadzonego pomiaru. Do każdego pomiaru w momencie zapisu zostaje przypisany losowy UUID, czyli unikatowy identyfikator, za pomocą statycznej funkcji klasy UUID z paczki `java.util`. Ułatwia on rozróżnienie poszczególnych pomiarów oraz służy jako nazwa pliku zapisywanego do systemu plików aplikacji.

```
data class Study(  
    val id: String = UUID.randomUUID().toString(),  
    val date: String,  
    val pulse: Int,  
    val times: List<Int> = listOf(),  
    val raw: List<Double> = listOf(),  
    val filtered: List<Double> = listOf(),  
    val peaks: List<Int> = listOf()  
)
```

Listing 4. Klasa *Study*, która zawiera dane dotyczące przeprowadzonego pomiaru.

Publiczne pola klasy:

- **id** – zmienna typu `String` zawierająca losowy, unikatowy identyfikator
- **date** – zmienna typu `String`, zawierająca datę w formacie: `dd/mm/yyyy gg:mm`
- **pulse** – zmienna typu `Int`, określająca obliczony przez aplikację puls
- **times** – lista wartości typu `Int`, zawierająca wskaźniki czasu dla kolejnych próbek zarejestrowanego sygnału
- **raw** – lista wartości typu `Double`, zawierająca kolejne wartości próbek surowego sygnału
- **filtered** – lista wartości typu `Double`, zawierająca kolejne wartości próbek sygnału po filtracji
- **peaks** – lista indeksów (liczby całkowite), na których wykryte zostały szczyty.

Po każdym pomiarze zostaje tworzony obiekt klasy *Study*, który następnie zapisywany do pliku w postaci tekstowej w przestrzeni aplikacji. W celu serializacji i deserializacji obiektów klasy *Study* wykorzystana została biblioteka *Gson* [34]. Funkcje zawarte w bibliotece dokonują

konwersji kotlinowego obiektu na format tekstowy JSON. Podczas eksportu pomiaru z aplikacji również w tym formacie zostaje zapisany plik w pamięci urządzenia mobilnego. Zaletą formatu JSON jest to, że jest dobrze czytelny dla człowieka.

```
{
  "id": "11c9d8c2-0de3-4c48-bf2b-9bcfb5c2cc5d",
  "date": "22/11/2022 11:29",
  "raw": [37.53772786458334, 39.01825846354167, 41.64952799479167, 42.59622721354,
  "times": [0, 171, 206, 246, 293, 327, 368, 410, 453, 490, 528, 568, 607, 655, 691, 730, 775,
  "filtered": [0.5227599502660496, 0.47953014843085384, 0.5296516724698677, 0.572
  "peaks": [3, 8, 28, 49, 67, 71, 93, 114, 134, 153, 172, 194, 212, 218, 235, 240, 258, 263, 282
  "pulse": 86
}
```

Rysunek 7. Przykładowa zawartość wyeksportowanego pliku z pomiarem (JSON).

Funkcje rozszerzające (z ang. extension function) związane z klasą *Study*:

```
fun Study.toJson(): String = Gson().toJson( src: this)
```

Listing 5. Funkcja rozszerzająca działanie klasy *Study*. Zwraca obiekt tej klasy w formacie tekstowym JSON.

```
fun String.toStudy(): Study =
    Gson().fromJson( json: this, Study::class.java)
```

Listing 6. Funkcja rozszerzająca działanie klasy *String*. Funkcja zajmuje się deserializacją zmiennej tekstowej na obiekt klasy *Study*.

```
fun Study.fps(): Int {
    val frames = this.times.size
    val periodMs = (this.times.last() - this.times.first()).toDouble()
    return (1000.0 * frames / periodMs).toInt()
}
```

Listing 7. Funkcja rozszerzająca działanie klasy *Study*. Na podstawie zawartości zmiennej *times* (zawierającą wskaźniki czasowe) obliczona zostaje średnia częstotliwość próbkowania w *fps*.

Na listingu 8. została pokazana abstrakcyjna klasa *AlgState*, która reprezentuje etap, w którym obecnie znajduje się algorytm. Klasa *AlgState* jest typu zapieczętowanego (z ang. sealed class), która służy ograniczaniu hierarchii klas do specyficznych typów pochodnych.

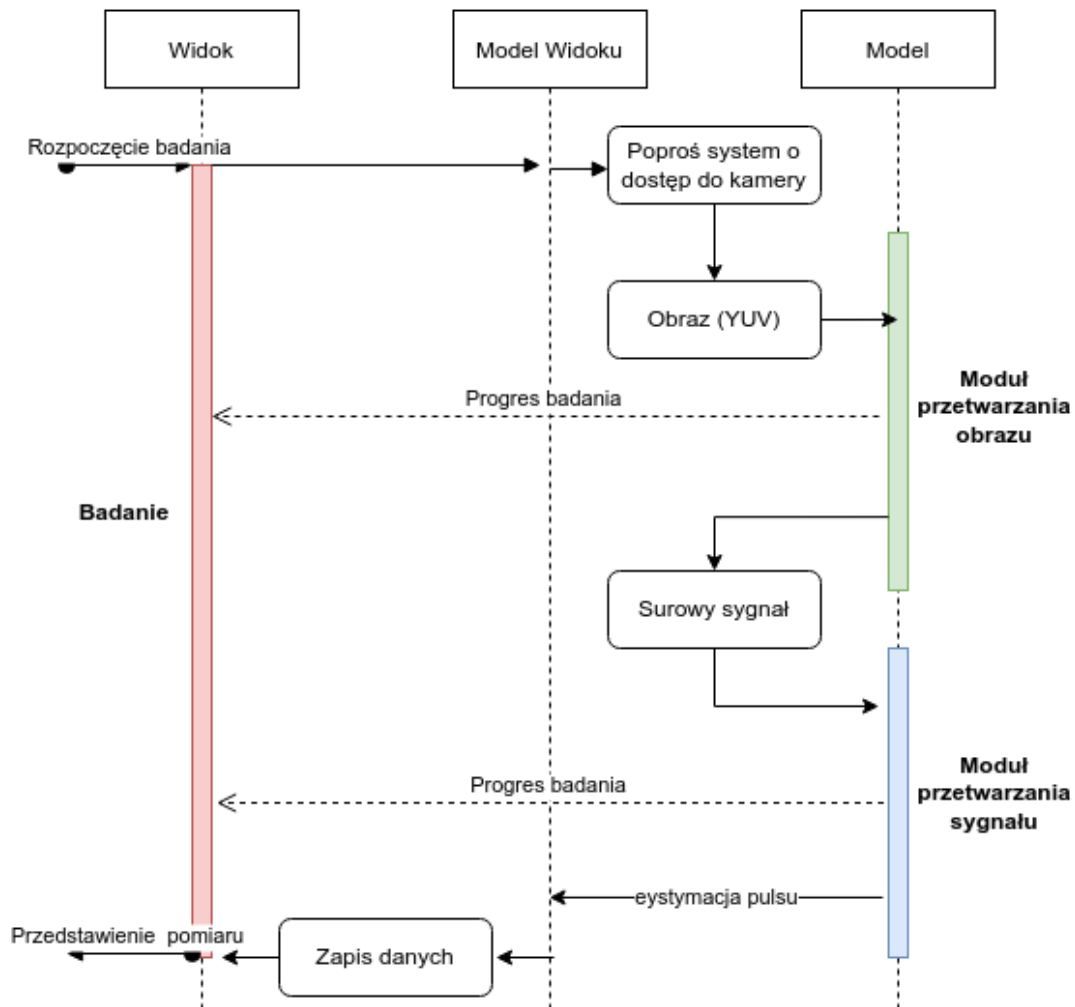
Na początku algorytm jest w stanie gotowości, który reprezentuje klasa *NONE*. Klasa *Calibration* związana jest z etapem kalibracji. Posiada pole typu boolean określające poprawność weryfikacji, czy palec jest w odpowiednim miejscu. Następnym etapom algorytmu, czyli rejestracji sygnału i obliczaniu średniego tętna odpowiadają klasy *Register* i *Finished*. Wynik działania algorytmu przechowywany jest w polu klasy *Result*, w celu wyświetlenia go użytkownikowi.

```
sealed class AlgState {
    object NONE : AlgState()
    class Calibration(var isFingerInPlace: Boolean = false) : AlgState()
    object Register : AlgState()
    object Finished : AlgState()
    class Result(val pulse: Int): AlgState()
}
```

Listing 8. Klasa AlgState. Reprezentuje stan, w którym obecnie znajduje się algorytm.

## Model widoku

Model widoku (z ang viewmodel) to warstwa zarządzająca przepływem danych w aplikacji w architekturze MVVM (Model-View-ViewModel). Jest to warstwa pośrednicząca pomiędzy modelem a widokiem. Zapewnia interfejsowi graficznemu dostęp do danych, zawartych w warstwie modelu, potrzebnych do poprawnego wyświetlenia widoku. Przechwytuje zdarzenia powstające w aplikacji i odpowiednio uaktualnia stan komponentów ekranu.



Rysunek 8. Schemat procesu pomiaru

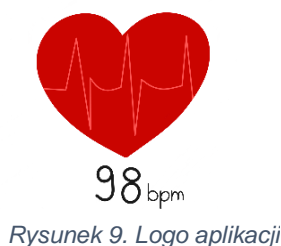
Na rysunku 8. (strona 15) przedstawiony został schemat procesu przeprowadzania pomiaru. Pokazany został przepływ danych w aplikacji poprzez poszczególne warstwy.

Architektura MVVM dzieli implementację odpowiadającą za interfejs użytkownika od kodu odpowiadającego za logikę domenową. Ta separacja obowiązków sprzyja tworzeniu czystego kodu. Dzięki temu również łatwiej zautomatyzować testowanie programu za pomocą testów jednostkowych, których zadaniem jest przetestowanie w izolacji danych funkcji programu, żeby zweryfikować poprawność działania całego systemu.

Użytkownik inicjalizuje proces badania (zaznaczony na schemacie kropką z strzałką). Następnie w odpowiedzi na to wydarzenie, model widoku sprawdza czy aplikacja ma dostęp do systemowej kamery i prosi o dostęp do niej. W przypadku niepowodzenia użytkownik zostanie poinformowany o tym komunikatem systemowym. W przypadku gdy aplikacja uzyska dostęp do aparatu, podaje napływające obrazy na wejście modułu przetwarzania obrazu. Model widoku przechowuje informację o czasie, który upłynął od rozpoczęcia badania i na jej podstawie udostępnia warstwie widoku daną opisującą progres pomiaru. Po upływie 25s przeznaczonego na badanie, uzyskany sygnał ulega przetworzeniu w module zaznaczonym kolorem niebieskim. Wynik działania algorytmu zostaje zapisany w pamięci wewnętrznej aplikacji, a następnie następuje jego prezentacja użytkownikowi.

### 5.3. Interfejs użytkownika

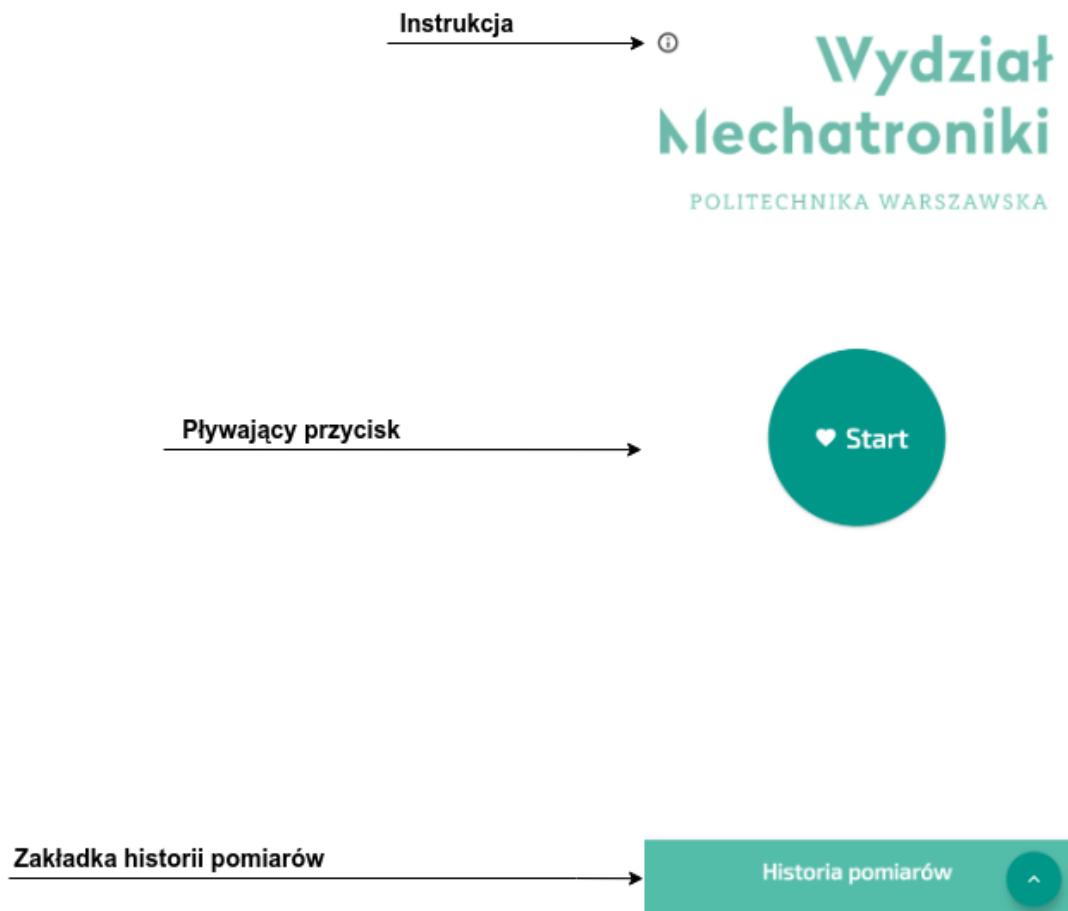
#### 5.3.1. Ekran początkowy



*Rysunek 9. Logo aplikacji*

Po uruchomieniu aplikacji, w oknie wyświetla się ekran ładowania (z ang. splash screen) z wykonanym logo aplikacji (rysunek 9). Logo pojawia się również na ikonce aplikacji, która roboczo przyjęła nazwę Mierzo puls. W czasie wyświetlania ekranu, system uruchamia aplikację i ładuje odpowiednie dane np. historyczne pomiary zapisane w przestrzeni plików aplikacji. Po dwóch 2 sekundach aplikacja automatycznie przenosi użytkownika na ekran główny. Jeżeli użytkownik po raz pierwszy uruchamia aplikację, powita go ona instrukcją z informacjami dotyczącymi sposobu i formy przeprowadzania badania.

### 5.3.2. Ekran główny



Rysunek 10. Ekran główny.

Jest to główny ekran aplikacji, który zawiera komponenty oznaczone na rysunku 10. Na środku ekranu znajduje się pływający przycisk sterujący stanem algorytmu. Poniżej jest wysuwana zakładka z ekranem historii pomiarów, który można aktywować przesuwając palcem zakładkę ku górze ekranu lub naciskając przycisk na zakładce. Po udanej kalibracji telefon zawibruje jeden raz, a drugi raz na zakończenie pomiaru. Dzięki tej funkcji pomiar można również przeprowadzić w pozycji, w której użytkownik nie widzi wyświetlacza telefonu. Wzbogaca to również komfort użytkowania.

#### Zakładka historii pomiarów

Zakładka dolna to wbudowany komponent z biblioteki Material UI. Gotowy obiekt ma zaimplementowane funkcje takie jak animację przesuwania zakładki oraz udostępnia interfejs do wyświetlenia ekranu, który ukazuje się po przeciągnięciu zakładki do góry. W aplikacji służy do wyświetlania zawartości ekranu historii pomiarów.



## Pływający przycisk



Rysunek 11 . Stany pływającego przycisku.

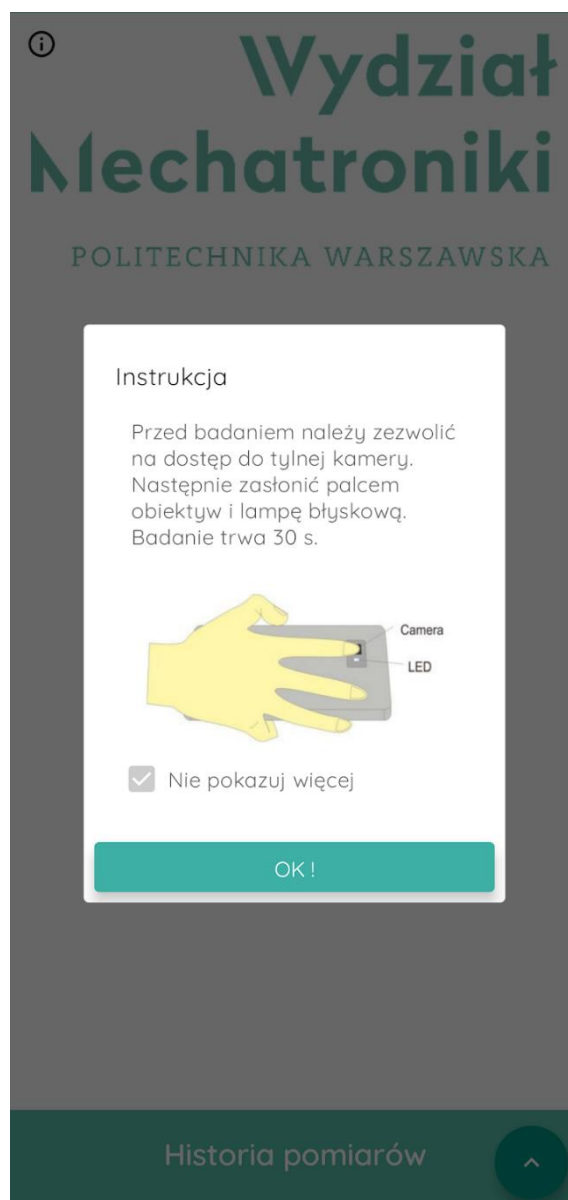
Element ten jest zależny od parametrów stanu aplikacji takich jak stan pomiaru reprezentowany przez klasę AlgState i progres badania, który jest zmienną zmiennoprzecinkową określającą progres badania od 0 do 1. Po naciśnięciu przycisku, tekst na przycisku zmienia się z “Start” na “Przyciśnij palec”, a stan pomiaru zmienia się z neutralnego na kalibrację. W tym samym czasie wokół przycisku rysuje się obręcz, której okrażenie przycisku sygnalizuje koniec badania. Po udanej kalibracji napis zmienia się na “Mierzę puls” oraz rozpoczyna się właściwa rejestracja. Gdyby użytkownika z jakiegoś powodu chciał przerwać badanie to należy nacisnąć przycisk podczas trwania badania. Po udanym pomiarze na przycisku zostanie wydrukowane obliczone średnie tętno. Po ponownym przyciśnięciu przycisk wraca do stanu początkowego.

Na rysunku 11. Kolejno od lewego przycisku, na górze, pokazany jest wygląd: stanu neutralnego przycisku, przycisku w trakcie kalibracji algorytmu. Na dole po lewej stronie zaprezentowany jest przycisk w trakcie trwania pomiaru, a po prawej prezentacja wyniku działania algorytmu.



## Instrukcja

Gdy po raz pierwszy zainstalujemy aplikację po jej otwarciu pokaże się instrukcja, zawierająca wskazówki i informacje odnośnie do przeprowadzanych pomiarów. Użytkownik może określić, czy instrukcja ma się wyświetlać przy uruchomieniu aplikacji odpowiednim przyciskiem. Otworzyć instrukcję można również manualnie przyciskiem Instrukcja na ekranie głównym.

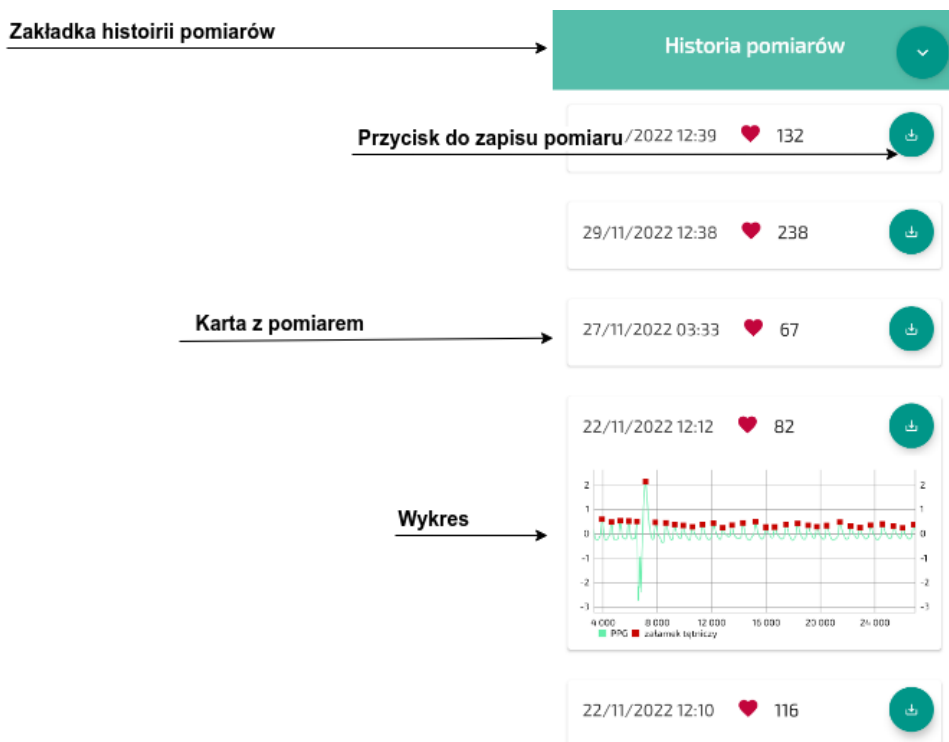


Rysunek 12. Okno modalne z instrukcją

### 5.3.3. Ekran historii wyników

Interfejs użytkownika, który ma za zadanie umożliwić użytkownikowi interakcję z zapisanymi badaniami. Widok zaimplementowany jest jako lista wierszy, z których każdy wyświetla informacje dotyczące historycznego pomiaru, takie jak data pomiaru oraz wyznaczona średnia częstotliwość bicia serca. Z prawej strony każdego wiersza znajduje się przycisk umożliwiający pobranie zapisu z badania w formie tekstowej w pliku z rozszerzeniem json. Dodatkowym elementem interfejsu jest możliwość podejrzenia wykresu uzyskanego sygnału fotopletyzmoграфicznego po filtracji z zaznaczonymi znalezionymi załamkami tętniczymi. W

celu wyświetlenia wykresu należy nacisnąć dany wiersz. Podobną operację należy wykonać w celu minimalizacji wiersza.



Rysunek 13. Ekran historii pomiarów

## Karta z pomiarem

Najbardziej podstawowym komponentem kontenerowym w bibliotece Material Design jest karta. Każdy wiersz listy pomiarów jest białą kartą, na której wydrukowane zostały odpowiednie informacje dotyczące odbytych pomiarów. Dzięki ich wykorzystaniu biblioteka automatycznie maluje odpowiednie cienie, co wzbogaca wizualnie interfejs użytkownika.

## 6. Weryfikacja

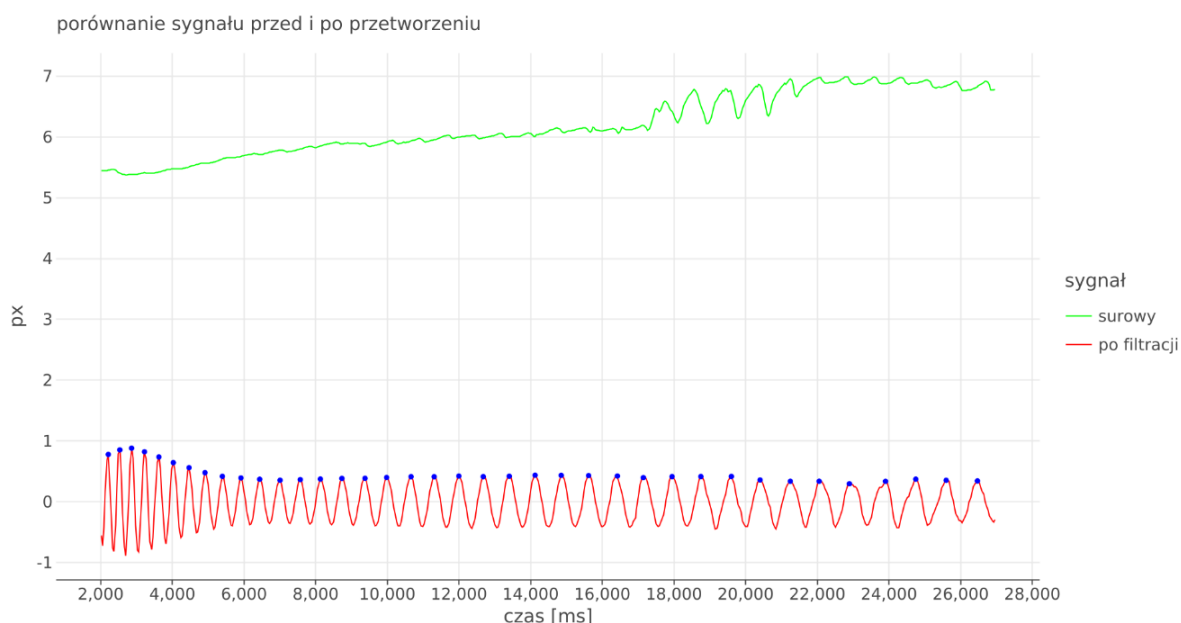
### 6.1. Oprogramowania

Android jest systemem operacyjnym, który może działać na urządzeniach różnych marek smartfonów, posiadających różne podzespoły. Ważne jest by mieć pewność, że zaimplementowana aplikacja działa poprawnie niezależnie od urządzenia.

Najbardziej podstawowym sposobem testowania oprogramowania są testy manualne. Wykorzystane zostały narzędzia z platformy Firebase. Firebase Crashlytics [24], służy do automatycznego zgłaszania awarii aplikacji. W raporcie otrzymujemy informację o modelu urządzenia, wersji Androida oraz o błędzie, który wystąpił. Użyty również został Firebase Analytics [23], do zbierania informacji o średniej ilości klatek na sekundę podczas badania. Aplikacja została przetestowana na telefonach Motorola g(60)s, Samsung Galaxy A52 oraz Xiaomi Redmi Note 8T. Smartfony posiadały wersję 11 Androida. Oprogramowanie działało płynnie oraz wszystkie zgłoszone usterki zostały naprawione. Wszystkie badane telefony miały stałą wartość próbkowania równą 25 Hz.

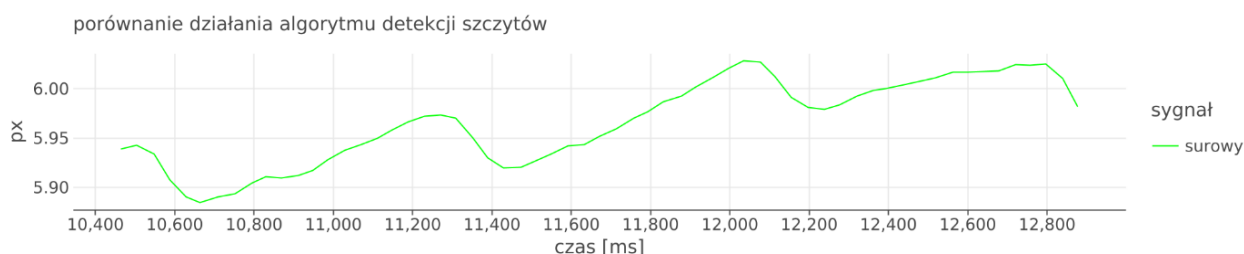
## 6.2. Działania algorytmu

Żeby zweryfikować poprawność działania zaimplementowanych modułów w separacji, zostało przygotowane przykładowe nagranie palca przyłożonego do obiektywu z włączoną wbudowaną lampą. Następnie uzyskane w ten sposób obrazy zostały podane na wejście modułu przetwarzania obrazu. Wynik działania przedstawiony zielonym kolorem na rysunku 14. Uzyskany surowy sygnał następnie został poddany filtracji (opisanej w wcześniejszych rozdziałach), której rezultat zaznaczony kolorem zielony (rysunek 14). Wykryte przez algorytm szczyty zaznaczone są kolorem niebieskim

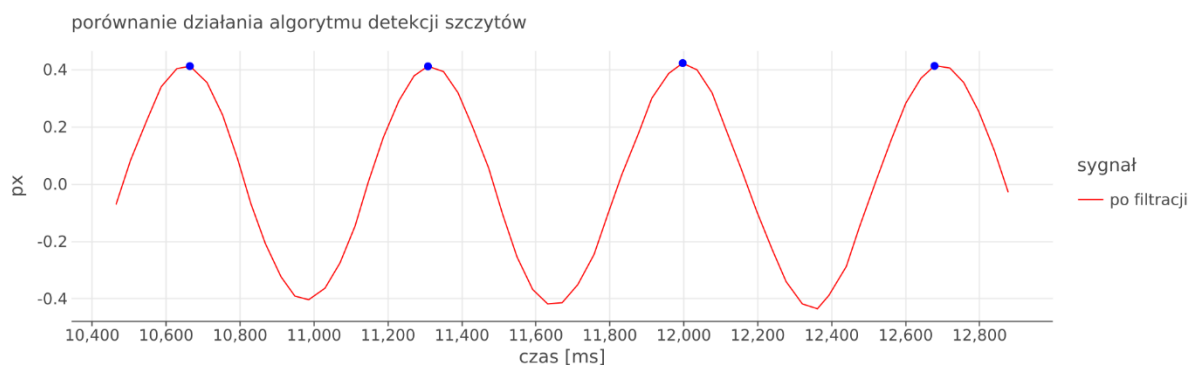


Rysunek 14. Wykres przedstawia surowy sygnał uzyskany po przetworzeniu obrazów oraz efekt filtracji modułu przetwarzania sygnału.

Na powyższym wykresie da się zauważyć, że zastosowana filtracja górnoprzepustowa dobrze wytłumiła składowe poniżej 1 Hz. W ten sposób uwidocznione zostały subtelne zmiany w średniej wartości piksela związane z przepływem krwi tętniczej. Na rysunku 15. został przedstawiony przebieg surowego sygnału w dwu sekundowym przedziale czasu. Dopiero na tym wykresie da się zauważyć (w porównaniu z rysunkiem 14.) składową pulsacyjną. Kształt sygnału zgadza się z przewidywaniami z artykułu [1].



Rysunek 15. Wynik działania modułu przetwarzania obrazu dla czasu od 10 do 12 s.



Rysunek 16. Wynik działania modułu przetwarzania sygnału dla czasu od 10 do 12 s.

Na rysunku 16. Pokazany został przefiltrowany sygnał wraz z wykrytymi szczytami w tym samym przedziale czasowym. Na podstawie porównania rysunków 15 i 16 można ocenić, że algorytm poprawnie określa momenty zmiany średniej wartości natężenia pikseli w analizowanym nagraniu.

### 6.3. Poprawności szacowania tętna

Zgodnie z założeniami projektowymi działanie algorytmu zostało zweryfikowane poprzez referencyjny pomiar za pomocą pulsoksymetru Nellcor Ultra Cap N-6000B. W aplikacji jest liczona średnia częstotliwość bicia serca z 25 s, a użyty pulsoksymetr aktualizował wartości w sposób beat-to-beat, dlatego należało oszacować średnie wskazanie, podczas trwania pomiaru na smartfonie. Odbyło się to poprzez obserwację odczytów z pulsoksymetru i na tej podstawie została określona wartość najczęściej występująca. Protokół badania zakładał przeprowadzenie pomiarów porównawczych podczas spoczynku, lekkiej aktywności fizycznej (spacer), oraz po serii ćwiczeń. Zgodność została oceniona na podstawie wskaźnika jakim jest wykres Blanda-Altmana. Wskaźnik najczęściej wykorzystuje się w przypadku porównania nowej metody pomiaru z obowiązującym złotym standardem, którym w tym eksperymencie był pulsoksymetr.

#### 6.3.1. Wyniki

Zostało zebranych 20 pomiarów, 10 dla stanu w spoczynku i po 5 dla prób po określonej aktywności fizycznej. Pomiary w uderzeniach na minutę zawarte w tabelach 1, 2, 3.

Tabela 1. Pomiary w spoczynku

Pomiar w spoczynku	
Pulsoksymetr [bpm]	Badana aplikacja [bpm]
79	80
73	72
82	79
71	69
75	73
72	72
72	80
74	78
70	70
75	75

Tabela 2. Pomiary po spacerze

Pomiar po spacerze	
Pulsoksymetr [bpm]	Badana aplikacja [bpm]
90	91
102	103
91	94
110	116
85	82

Tabela 3. Pomiary po ćwiczeniach

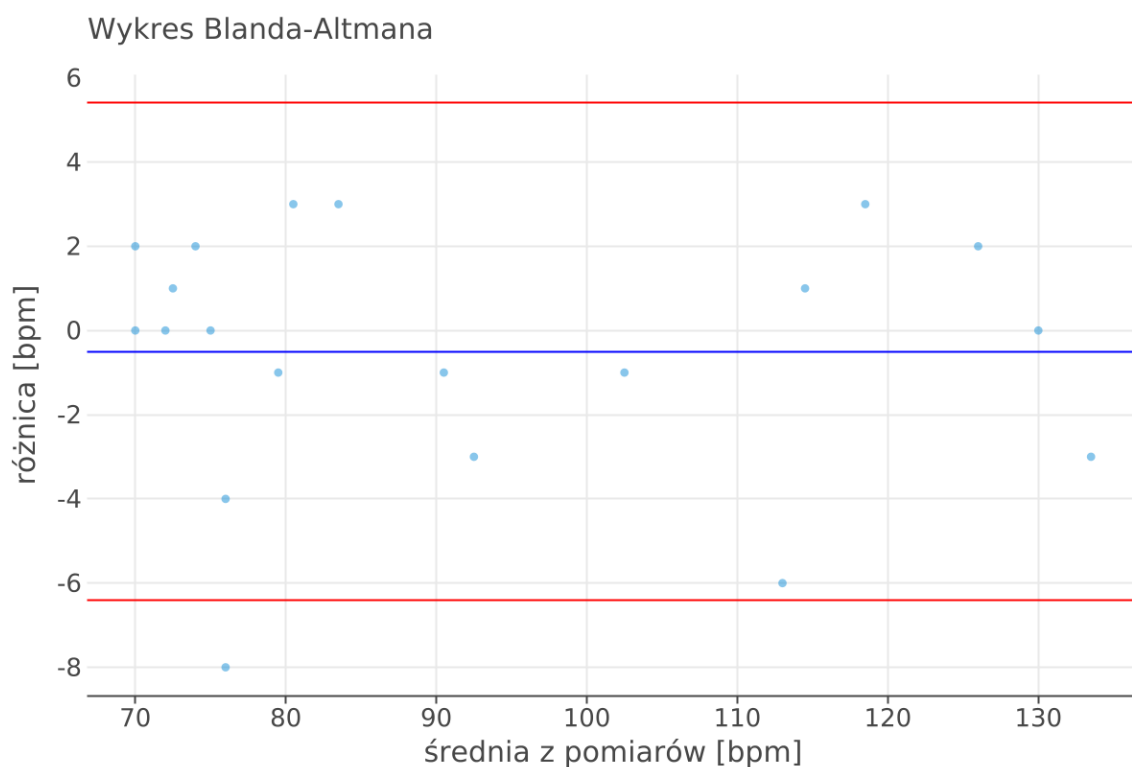
Pomiar po ćwiczeniach	
Pulsoksymetr [bpm]	Badana aplikacja [bpm]
115	114
120	117
130	130
127	125
132	135

### 6.3.2. Dyskusja

Rysunek 17. obrazuje wyniki pomiarów na wykresie Blanda-Altmana. Poziomą niebieską linią oznaczona została średnia różnica, natomiast czerwone linie wskazują 95% przedział zgodności przy założeniu, że różnice podlegają rozkładowi normalnemu. Obliczona średnia różnica wniosła -0,5 bpm, świadczy to o tym, że średni błąd pomiaru jest mały.

Przedział zgodności znalazł się w zakresie od -6,4 do 5.4 bpm i prawie wszystkie punkty zawierały się w nim. Z danych wynika, że badana wartość badanego parametru ma rozrzut 11 bpm.

Biorąc pod uwagę założenia konstrukcyjne, aplikację można użyć w celach rekreacyjnych lub sportowych. Dostępne urządzenia sportowe często wykorzystują informację o pulsie w celu określenia strefy tętna, w której znajduje się użytkownik. Parametr potrzebny w do oceny strefy tętna to tętno maksymalne (HRmax), które jest zależne od stanu fizycznego użytkownika. W przypadku badanego obiektu parametr ten wynosi 190 bpm. Według producenta opasek sportowych Polar [13] można wyróżnić 5 stref tętna, których zakresy zostały przedstawione w tabeli 4. Dodatkowo w trzeciej kolumnie zostały obliczone przedziały dla badanego.



Rysunek 17. Porównanie wyników z aplikacji z pulsoksymetrem.

Po podstawie wartości w tabelach 1,2,3,4 można orzec, że pomiary przeprowadzone przez aplikację we prawie wszystkich (poza jednym przypadkiem) przypadkach zaklasyfikowałyby badanego do właściwej strefy tętna. Pomiary w spoczynku dla obu metod wahały się w przedziale od 70 – 82 bpm, co nie klasyfikuje się jako trening, co zgadza się z założeniem tej części pomiarów. Wyniki w tabeli dotyczącej badania po spacerze zawierały się w przedziale 82-116 bpm, w którym w całości zawiera się przedział lekkiej intensywności treningu. Wartości dla pomiarów po lekkim treningu (114 -135 bpm) zaklasyfikowałyby trening jako „lekki”. Przeprowadzone badania wskazują, iż błąd pomiarowy aplikacji jest na tyle mały, że umożliwia określenie strefy tętna.

Tabela 4. Strefy tętna [10]

Strefy tętna		
Intensywność treningu	Stosunek tętna do HRmax	Dla HRmax = 190 bpm
Bardzo lekka	50 – 60 %	95 – 114 bpm
Lekka	60 – 70 %	114 – 133 bpm
Średnia	70 – 80 %	133 – 152 bpm
Wysoka	80 – 90 %	152 – 171 bpm
Maksymalna	90 – 100 %	171 – 190 bpm

## 7. Podsumowanie

Przedstawione rozwiązanie implementuje wszystkie wymagania projektowe. Aplikacja umożliwia interakcję użytkownika z jej funkcjami, którą główną z nich jest kalkulacja tętna użytkownika za pomocą wbudowanej kamery i oświetlenia. Poprawność działania została oceniona na podstawie przeprowadzonych badań porównawczych z pulsoksymetrem. W stosunku do fluktuacji badanego parametru algorytm z akceptowalnym dla planowanych zastosowań określa tętno użytkownika.

Głównym ograniczeniem aplikacji jest brak odporności na przemieszczenie palca podczas pomiaru oraz wymaganie statyczności podczas badania. W następnych iteracjach implementacji algorytmu, mógłby zostać dodany moduł odpowiedzialny usuwania artefaktów

Na ten moment oprogramowanie bada średnią częstotliwość bicia serca w określonym przedziale czasowym. Rozwiązanie było inspirowane działaniem podobnych aplikacji w sklepie Play (market z aplikacjami dla urządzeń z systemem Android). Potencjalnym kierunkiem rozwoju aplikacji jest wprowadzenie pomiaru ciągłego. Ułatwiłoby to porównanie z pulsoksymetrem, który mierzy tętno w takim trybie. Znajac momenty wystąpienia załamków tętnicznych można obliczyć parametr zmienności rytmu zatokowego (HRV). Opisuje on wariancję w czasie między uderzeniami serca. Wyznaczenie tego i innych potencjalnych parametrów również mogłoby rozszerzyć funkcje aplikacji.

## 8. Bibliografia

- [1] N. V. Hoan, J.-H. Park, S.-H. Lee, and K.-R. Kwon, 'Real-time Heart Rate Measurement based on Photoplethysmography using Android Smartphone Camera', *Journal of Korea Multimedia Society*, vol. 20, pp. 234–243, 2 2017.
- [2] S. Kwon, H. Kim, and K. S. Park, 'Validation of heart rate extraction using video imaging on a built-in camera system of a smartphone', 2012, pp. 2174–2177.
- [3] R. B. Lagido, J. Lobo, S. Leite, C. Sousa, L. Ferreira, and J. Silva-Cardoso, 'Using the smartphone camera to monitor heart rate and rhythm in heart failure patients', 2014, pp. 556–559.
- [4] F. Lamonaca, Y. Kurylyak, D. Grimaldi, and V. Spagnuolo, 'Reliable pulse rate evaluation by smartphone', 2012, pp. 234–237.
- [5] J. Lázaro, Y. Nam, E. Gil, P. Laguna, and K. H. Chon, 'Smartphone-camera-acquired pulse photoplethysmographic signal for deriving respiratory rate', 2014 8th Conference of the European Study Group on Cardiovascular Oscillations, ESGCO 2014. IEEE Computer Society, pp. 121–122, 2014.
- [6] S. A. Siddiqui, Y. Zhang, Z. Feng, and A. Kos, 'A Pulse Rate Estimation Algorithm Using PPG and Smartphone Camera', *Journal of Medical Systems*, vol. 40, 5 2016.
- [7] P. Pelegris, K. Banitsas, T. Orbach, and K. Marias, 'A novel method to detect heart beat rate using a mobile phone', 2010, pp. 5488–5491.
- [8] E. Mejía-Mejía et al., 'Photoplethysmography Signal Processing and Synthesis', In *Photoplethysmography*, Kyriacou, P.A., Allen, J., Eds.; Elsevier, 2021.
- [9] K. K. Tremper and S. J. Barker, 'Pulse Oximetry', *Anesthesiology*, vol. 70, pp. 98–108, 1 1989.
- [10] A. Bánhalmi, J. Borbás, M. Fidrich, V. Bilicki, Z. Gingl, and L. Rudas, 'Analysis of a Pulse Rate Variability Measurement Using a Smartphone Camera', *Journal of Healthcare Engineering*, vol. 2018. Hindawi Limited, 2018
- [11] PQStat, dostęp 15.01.2023 r, <http://manuals.pqstat.pl/statpqpl:plotpl:bladpl>

- [12] Rekomendacje dotyczące architektury aplikacji w Androidzie, dostęp 15.01.2023 r, <https://developer.android.com/topic/architecture/recommendations>
- [13] Strefy tętna, dostęp 15.01.2023 r, <https://www.polar.com/blog/running-heart-rate-zones-basics/>
- [14] AndroidX Core (wersja 1.8.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.core>
- [15] AndroidX AppCompat (wersja 1.4.2), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.appcompat>
- [16] AndroidX Activity (wersja 1.5.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.activity/activity-compose>
- [17] AndroidX Work (wersja 2.7.1), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.work/work-runtime-ktx>
- [18] AndroidX Lifecycle (wersja 2.5.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.lifecycle/lifecycle-runtime>
- [19] AndroidX Compose UI (wersja 1.0.5), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.compose.ui/ui>
- [20] AndroidX Compose Material (wersja 1.0.5), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.compose.material/material>
- [21] AndroidX Compose Lifecycle Viewmodel (wersja 2.4.1), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.appcompat>
- [22] Android Material (wersja 1.6.1), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/com.google.android.material/material>
- [23] Firebase Analytics (wersja 30.2.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/com.google.firebase/firebase-analytics>
- [24] Firebase Crashlytics (wersja 30.2.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/com.google.firebase/firebase-crashlytics>
- [25] Camera Core (wersja 1.1.0-alpha11), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.camera/camera-core>
- [26] Camera Camera2 (wersja 1.1.0-alpha11), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.camera/camera-camera2>
- [27] Camera Lifecycle (wersja 1.1.0-alpha11), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.camera/camera-lifecycle>
- [28] Camera View (wersja 1.0.0-alpha31), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.camera/camera-view>
- [29] Camera Extensions (wersja 1.0.0-alpha31), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/androidx.camera/camera-extensions>
- [30] Koin Android (wersja 3.2.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/io.insert-koin/koin-android>
- [31] Koin AndroidCompat (wersja 3.2.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/io.insert-koin/koin-android-compat>
- [32] Koin AndroidX Workmanager (wersja 3.2.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/io.insert-koin/koin-androidx-workmanager>
- [33] Koin AndroidX Compose (wersja 3.2.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/io.insert-koin/koin-androidx-compose>
- [34] Gson (wersja 4.5.3.0), dostęp 13.09.22 r, <https://mvnrepository.com/artifact/com.google.code.gson/gson>
- [35] OpenCV Android (wersja 4.5.3.0), dostęp 13.09.22 r, <https://github.com/QuickBirdEng/opencv-android>



[36] Java Digital Signal Processing (wersja 2.0.0), dostęp 13.09.22 r,  
<https://github.com/psambit9791/jdsp>

[37] Android Chart (wersja 3.1.0), dostęp 13.09.22 r,  
<https://github.com/PhilJay/MPAndroidChart>

## 9. Wykaz symboli i skrótów

- bpm – uderzenia serca na minutę (z ang. beats per minute)
- dB – decybel
- fps – klatki na sekundę (z ang. frames per second)
- HbCO<sub>2</sub> – deoksyhemoglobina
- HbO<sub>2</sub> – oksyhemoglobina
- HRmax – tętno maksymalne
- HRV – zmienność rytmu zatokowego (z ang)
- Hz – Hertz
- JDSP – biblioteka Java Digital Signal Processing [36]
- JSON – format pliku tekstowego
- JVM – wirtualna maszyna Javy
- MVVM – wzorzec projektowy Model-View-ViewModel
- ms – milisekunda
- px – wartość piksela
- RGB – model barw RGB
- s – sekunda
- UUID – unikalny identyfikator
- View-model – model widoku (z ang. View-model)
- YUV – model barw YUV

## 10. Spis rysunków

Rysunek 1. Przykładowy sygnał PPG rejestrowany, przez aplikację, na palcu człowieka.....	4
Rysunek 2 Schemat blokowy działania algorytmu obliczania średniego tętna. Czarnym prostokątem zaznaczony został podział implementacji algorytmu na dwa moduły (przetwarzania obrazów i przetwarzania sygnału). .....	6
Rysunek 3. Średnia wartość pikseli w czasie dla kanału czerwonego. Na osi X oznaczony jest czas w milisekundach. Na osi y obliczona średnia wartość pikseli w kanale. ....	8
Rysunek 4. Średnia wartość pikseli w czasie dla kanału zielonego. Na osi X oznaczony jest czas w milisekundach. Na osi y obliczona średnia wartość pikseli w kanale. ....	8
Rysunek 5. Średnia wartość pikseli w czasie dla kanału niebieskiego. Na osi X oznaczony jest czas w milisekundach. Na osi y obliczona średnia wartość pikseli w kanale. ....	8
Rysunek 6. Znormalizowana charakterystyka częstotliwościowa zastosowanego filtru. ....	10
Rysunek 7. Przykładowa zawartość wyeksportowanego pliku z pomiarem (JSON). ....	14
Rysunek 8. Schemat procesu pomiaru.....	15
Rysunek 9. Logo aplikacji .....	16
Rysunek 10. Ekran główny.....	17
Rysunek 11 . Stany pływającego przycisku.....	18
Rysunek 12. Okno modalne z instrukcją .....	19
Rysunek 13. Ekran historii pomiarów .....	20

Rysunek 14. Wykres przedstawia surowy sygnał uzyskany po przetworzeniu obrazów oraz efekt filtracji modułu przetwarzania sygnału. ....	21
Rysunek 15. Wynik działania modułu przetwarzania obrazu dla czasu od 10 do 12 s. ....	21
Rysunek 16. Wynik działania modułu przetwarzania sygnału dla czasu od 10 do 12 s. ....	22
Rysunek 17. Porównanie wyników z aplikacji z pulsoksymetrem. ....	24

## 11. Spis tabel

Tabela 1. Pomiary w spoczynku.....	22
Tabela 2. Pomiary po spacerze.....	23
Tabela 3. Pomiary po ćwiczeniach.....	23
Tabela 4. Strefy tętna [10].....	24

## 12. Spis równań

Równanie 1. Prawo Lamberta-Beera .....	4
Równanie 2. Wzór na średnie tętno [4] .....	9
Równanie 3. Wzór, za pomocą którego został oszacowany minimalny możliwy odstęp między wykrytymi szczytami. $F_s$ - częstotliwość próbkowania w przypadku aplikacji 25 klatek na sekundę. ....	11
Równanie 4. Wzór, za pomocą którego został oszacowany maksymalny możliwy odstęp między wykrytymi szczytami. $F_s$ - częstotliwość próbkowania w przypadku aplikacji 25 klatek na sekundę. ....	11

## 13. Spis listingów

Listing 1. Klasa AlgState. Reprezentuje stan, w którym obecnie znajduje się algorytm. ....	15
Listing 2. Funkcja processImage, wejście - obraz w YUV, wyjście - średnie wartości pikseli dla kanałów RGB.....	7
Listing 3. Funkcja highpassFilter realizująca filtrację; przyjmuje na wejście surowy sygnał zapisany z znacznikami czasowymi, na wyjściu zwracany jest sygnał po filtracji.....	9
Listing 4. Funkcja peakFinder, znajdujący indeksy wystąpienia szczytów w sygnale. ....	11
Listing 5. Klasa Study, która zawiera dane dotyczące przeprowadzonego pomiaru. ....	13
Listing 6. Funkcja rozszerzająca działanie klasy Study. Zwraca obiekt tej klasy w formacie tekstowym JSON. ....	14
Listing 7. Funkcja rozszerzająca działanie klasy String. Funkcja zajmuje się deserializacją zmiennej tekstowej na obiekt klasy Study. ....	14
Listing 8. Funkcja rozszerzająca działanie klasy Study. Na podstawie zawartości zmiennej times (zawierającą wskaźniki czasowe) obliczona zostaje średnia częstotliwość próbkowania w fps. ....	14

## 14. Załączniki

### 14.1. Kod źródłowy

#### 14.1.1. MierzopulsApp.kt

```
package pl.pw.mierzopuls

import android.app.Application
import org.koin.android.ext.koin.androidApplication
import org.koin.android.ext.koin.androidContext
import org.koin.androidx.viewmodel.dsl.viewModel
import org.koin.core.context.GlobalContext
import org.koin.dsl.module
import pl.pw.mierzopuls.model.alg.ImageProcessing
import pl.pw.mierzopuls.model.alg.StudyManager
import pl.pw.mierzopuls.model.AppSetting
import pl.pw.mierzopuls.model.CameraLifecycle
import pl.pw.mierzopuls.model.StudyRepository
import pl.pw.mierzopuls.ui.HomeViewModel

class MierzopulsApp : Application() {

    private val utilModule = module {
        single { CameraLifecycle() }
        single { ImageProcessing }
    }

    private val repositoriesModule = module {
        single { StudyRepository() }
        single { AppSetting(androidContext()) }
    }

    private val appModule = module {
        single { StudyManager(get(), get()) }
        viewModel {
            HomeViewModel(androidApplication(), get(), get()) }
    }

    override fun onCreate() {
        super.onCreate()

        GlobalContext.startKoin {
            androidContext(applicationContext)
            modules(utilModule)
            modules(repositoriesModule)
            modules(appModule)
        }
    }
}
```

#### 14.1.2. MainActivity.kt

```
package pl.pw.mierzopuls

import android.os.Bundle
import android.view.WindowManager
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import
```

```
androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.core.splashscreen.SplashScreen.Companion.installSplashScreen
import org.koin.androidx.viewmodel.ext.android.viewModel
import pl.pw.mierzopuls.ui.Home
import pl.pw.mierzopuls.ui.HomeViewModel
import pl.pw.mierzopuls.ui.theme.MierzopulsTheme
```

```
class MainActivity : ComponentActivity() {
    @ExperimentalFoundationApi
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        window.addFlags(
            WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON
        )

        val vm: HomeViewModel = getViewModel()
        installSplashScreen().apply {
            this.setKeepOnScreenCondition {
                vm.isLoading
            }
        }
        setContent {
            MierzopulsTheme {
                Surface(color =
                    MaterialTheme.colors.background) {
                    Home()
                }
            }
        }
    }
}
```

#### 14.1.3. AlgState.kt

```
package pl.pw.mierzopuls.model.alg

sealed class AlgState {
    object NONE : AlgState()
    class Calibration(var isFingerInPlace: Boolean = false) : AlgState()
    object Register : AlgState()
    object Finished : AlgState()
    class Result(val pulse: Int): AlgState()
}
```

#### 14.1.4. ImageProcessing.kt

```
package pl.pw.mierzopuls.model.alg

import android.annotation.SuppressLint
import android.graphics.ImageFormat
import android.media.Image
import android.util.Log
import androidx.camera.core.ImageAnalysis
```

```

import org.opencv.android.OpenCVLoader
import org.opencv.core.*
import org.opencv.imgproc.Imgproc
import java.util.concurrent.Executors

object ImageProcessing {
    const val LOG_TAG = "ImgProc"
    init {
        if (!OpenCVLoader.initDebug()) {
            Log.e(LOG_TAG, "Unable to load
OpenCV! BE")
            throw InstantiationException(
                "OpenCV not loaded correctly!"
            )
        } else {
            Log.d(LOG_TAG, "OpenCV library loaded
correctly")
        }
    }

    fun processImage(image: Image): List<Double> {
        val mean = image.yuvToRgba().let {
            Core.mean(it)
        }
        return listOf(
            mean.`val`[0],
            mean.`val`[1],
            mean.`val`[2],
        )
    }

    fun isFingerInPlace(mean: List<Double>):
Boolean {
        return mean[1] < 100 && mean[2] < 100 &&
mean[0] > 170
    }

    @SuppressWarnings("UnsafeOptInUsageError")
    fun imageAnalysisUseCase(onImage: (Image) ->
Unit): ImageAnalysis {
        return ImageAnalysis.Builder()

        .setBackpressureStrategy(ImageAnalysis.STRATE
GY_KEEP_ONLY_LATEST)
        .build()
        .apply {

setAnalyzer(Executors.newSingleThreadExecutor())
{ imageProxy ->
            imageProxy.use {
                onImage(it.image!!)
            }
        }
    }

    /**
     * Converts YUV image to RGB matrix
     */
    private fun Image.yuvToRgba(): Mat {
        val rgbaMat = Mat()

        if (format == ImageFormat.YUV_420_888
            && planes.size == 3
        ) {
            val chromaPixelStride =
planes[1].pixelStride

```

```

        if (chromaPixelStride == 2) { // Chroma
channels are interleaved
            assert(planes[0].pixelStride == 1)
            assert(planes[2].pixelStride == 2)
            val yPlane = planes[0].buffer
            val uvPlane1 = planes[1].buffer
            val uvPlane2 = planes[2].buffer
            val yMat = Mat(height, width,
CvType.CV_8UC1, yPlane)
            val uvMat1 = Mat(height / 2, width / 2,
CvType.CV_8UC2, uvPlane1)
            val uvMat2 = Mat(height / 2, width / 2,
CvType.CV_8UC2, uvPlane2)
            val addrDiff = uvMat2.dataAddr() -
uvMat1.dataAddr()
            if (addrDiff > 0) {
                assert(addrDiff == 1L)
                Imgproc.cvtColorTwoPlane(yMat,
uvMat1, rgbaMat,
Imgproc.COLOR_YUV2RGBA_NV12)
            } else {
                assert(addrDiff == -1L)
                Imgproc.cvtColorTwoPlane(yMat,
uvMat2, rgbaMat,
Imgproc.COLOR_YUV2RGBA_NV21)
            }
        } else { // Chroma channels are not
interleaved
            val yuvBytes = ByteArray(width * (height
+ height / 2))
            val yPlane = planes[0].buffer
            val uPlane = planes[1].buffer
            val vPlane = planes[2].buffer

            yPlane.get(yuvBytes, 0, width * height)

            val chromaRowStride =
planes[1].rowStride
            val chromaRowPadding =
chromaRowStride - width / 2

            var offset = width * height
            if (chromaRowPadding == 0) {
                // When the row stride of the chroma
channels equals their width, we can copy
// the entire channels in one go
                uPlane.get(yuvBytes, offset, width *
height / 4)
                offset += width * height / 4
                vPlane.get(yuvBytes, offset, width *
height / 4)
            } else {
                // When not equal, we need to copy the
channels row by row
                for (i in 0 until height / 2) {
                    uPlane.get(yuvBytes, offset, width /
2)
                    offset += width / 2
                    if (i < height / 2 - 1) {
                        uPlane.position(uPlane.position()
+ chromaRowPadding)
                    }
                }
                for (i in 0 until height / 2) {
                    vPlane.get(yuvBytes, offset, width /
2)
                    offset += width / 2

```

```

        if (i < height / 2 - 1) {
            vPlane.position(vPlane.position()
+ chromaRowPadding)
        }
    }
}

val yuvMat = Mat(height + height / 2,
width, CvType.CV_8UC1)
yuvMat.put(0, 0, yuvBytes)
Imgproc.cvtColor(yuvMat, rgbaMat,
Imgproc.COLOR_YUV2RGBA_I420, 4)
}
}

return rgbaMat
}
}

```

#### 14.1.5. SignalProcessing.kt

```
package pl.pw.mierzopuls.model.alg
```

```

import
com.github.psambit9791.jdsp.filter.Butterworth
import
com.github.psambit9791.jdsp.signal.peaks.FindPea
k
import
com.github.psambit9791.jdsp.signal.peaks.Peak
import pl.pw.mierzopuls.model.Study
import pl.pw.mierzopuls.model.formatStudyDate
import java.util.*
import kotlin.math.roundToInt

```

```

fun processSignal(
    raw: List<Double>,
    times: List<Int>
): Study {
    val filtered: DoubleArray = highpassFilter(
        rawSignal = raw.toDoubleArray(),
        times = times.map { it.toDouble() / 1000
    }.toDoubleArray()
    )
    // drop first 50 samples due to filtering disruption
    val peaks: List<Int> = peakFinder(
        filtered.drop(50).toDoubleArray()
    )
    val pulse: Int = calculatePulse(
        times.drop(50),
        peaks
    ).roundToInt()

    return Study(
        date =
        Calendar.getInstance().formatStudyDate(),
        raw = raw,
        times = times.drop(50).map { it - times[0] },
        filtered = filtered.drop(50),
        peaks = peaks,
        pulse = pulse
    )
}

```

```

fun highpassFilter(
    rawSignal: DoubleArray, // px
    times: DoubleArray // s
): DoubleArray {

```

```

    val timeStart: Double = times.first()
    val timeEnd: Double = times.last()

    val fs: Double = times.size / (timeEnd - timeStart)
    // Hz
    val order = 200
    val cutOff = 1.0 // Hz

    return Butterworth(fs).highPassFilter(rawSignal,
    order, cutOff)
}

fun peakFinder(
    signal: DoubleArray
): List<Int> {
    val detectedPeaks: Peak = FindPeak(signal)
        .detectPeaks()
    val filtered: IntArray = detectedPeaks
        .filterByWidth(4.0, null)
    return filtered.toList()
}

```

```

fun calculatePulse(
    times: List<Int>,
    peaksIds: List<Int>
): Double {
    val peakTimestamps = times.filterIndexed { idx, _
    ->
        peaksIds.contains(idx)
    }
    val quantil2 = peakTimestamps.mapIndexed {
    idx, time ->
        if (idx + 1 >= peakTimestamps.lastIndex) 0.0
        else (peakTimestamps[idx + 1] - time) / 1000.0
    }.median()

    return 60.0 / quantil2
}

```

```

private fun List<Double>.median() =
this.sorted().let {
    if (it.size % 2 == 0)
        (it[it.size / 2] + it[(it.size - 1) / 2]) / 2
    else
        it[it.size / 2]
}

```

#### 14.1.6. StudyManager.kt

```
package pl.pw.mierzopuls.model.alg
```

```

import android.annotation.SuppressLint
import android.content.Context
import android.os.Build
import android.os.VibrationEffect
import android.os.Vibrator
import android.util.Log
import androidx.annotation.RequiresApi
import androidx.camera.core.CameraSelector
import androidx.camera.core.FocusMeteringAction
import androidx.camera.core.MeteringPointFactory
import
androidx.camera.core.SurfaceOrientedMeteringPoi
ntFactory
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import pl.pw.mierzopuls.model.CameraLifecycle

```

```
import pl.pw.mierzopuls.model.getCameraProvider
```

```
class StudyManager(
    private val cameraLifecycle: CameraLifecycle,
    private val imageProcessing:
    ImageProcessing,
) {
    companion object {
        const val CALIBRATION_TIME = 5000L
        const val REGISTRATION_TIME = 25000L
    }
    private val cameraSelector =
    CameraSelector.DEFAULT_BACK_CAMERA
    private val timeStamps =
    mutableListOf<Long>()
    private val values = mutableListOf<Double>()
    private var startTime = 0L
    private var onResult: (List<Long>,
    List<Double>) -> Unit = { _, _ -> }
    private var vibrator: Vibrator? = null

    var algState: AlgState by
    mutableStateOf<AlgState>(AlgState.NONE)
    var progress: Float by mutableStateOf(0.0f)

    @RequiresApi(Build.VERSION_CODES.O)
    @SuppressLint("UnsafeOptInUsageError")
    val imageAnalysisUseCase =
    ImageProcessing.imageAnalysisUseCase { image -
    >
        when (val state = algState) {
            is AlgState.NONE,
            is AlgState.Finished -> {
                Log.w(ImageProcessing.LOG_TAG, "Alg
                state = ${state.javaClass}")
            }
            is AlgState.Calibration -> {
                val mean =
                ImageProcessing.processImage(image)
                if
                (ImageProcessing.isFingerInPlace(mean)) {
                    values += mean[1]
                    timeStamps +=
                    System.currentTimeMillis()
                } else {
                    values.clear()
                    timeStamps.clear()
                    progress = 0f
                    startTime = System.currentTimeMillis()
                }
            }
            is AlgState.Register -> {
                values +=
                ImageProcessing.processImage(image)[1]
                timeStamps +=
                System.currentTimeMillis()
            }
        }

        if (algState is AlgState.Calibration || algState
        is AlgState.Register) {
            updateState(System.currentTimeMillis())
        }
    }

    @RequiresApi(Build.VERSION_CODES.O)
```

```
suspend fun beginStudy(context: Context,
    vibrator: Vibrator, onResult: (List<Long>,
    List<Double>) -> Unit) {
    this.onResult = onResult
    this.vibrator = vibrator
    startTime = System.currentTimeMillis()
    cameraLifecycle.doOnStart()
    prepareCamera(context)
    algState = AlgState.Calibration(false)
}
```

```
suspend fun dismissStudy(context: Context) {
    algState = AlgState.NONE
    cameraLifecycle.doOnDestroy()
    context.getCameraProvider().unbindAll()

    startTime = 0L
    progress = 0f
    timeStamps.clear()
    values.clear()
}
```

```
suspend fun finishStudy(context: Context) {
    progress = 1f
    algState = AlgState.Finished
    cameraLifecycle.doOnDestroy()
    context.getCameraProvider().unbindAll()
}
```

```
fun setResult(pulse: Int) {
    algState = AlgState.Result(pulse)
}
```

```
@RequiresApi(Build.VERSION_CODES.O)
private fun updateState(lastTime: Long) {
    (lastTime - startTime).let {
        if (algState is AlgState.Calibration && it >
        CALIBRATION_TIME) {
            algState = AlgState.Register
            values.clear()
            timeStamps.clear()
        }
```

```
vibrator?.vibrate(VibrationEffect.createOneShot(40
    0, 3))
    }
    if (algState is AlgState.Register && it >
    REGISTRATION_TIME + CALIBRATION_TIME) {
        this.onResult(timeStamps, values)
    }
    if (it > 200L) {
        progress = (it - 100L).toFloat() /
        (REGISTRATION_TIME + CALIBRATION_TIME -
        200L).toFloat()
    }
}
```

```
@RequiresApi(Build.VERSION_CODES.O)
private suspend fun prepareCamera(context:
    Context) {
    val cameraProvider =
    context.getCameraProvider()
    try {
        // Must unbind the use-cases before
        rebinding them.
        cameraProvider.unbindAll()
        val camera =
```

```

cameraProvider.bindToLifecycle(
    cameraLifecycle, cameraSelector,
    imageAnalysisUseCase
)
camera.cameraControl.enableTorch(true)
Log.d("CameraPrep", "torch enabled")

val factory: MeteringPointFactory =
    SurfaceOrientedMeteringPointFactory(
        10F, 10F
    )
val autoFocusPoint =
    factory.createPoint(1F, 1F)

camera.cameraControl.startFocusAndMetering(
    FocusMeteringAction.Builder(autoFocusPoint).disab
    leAutoCancel().build()
)
Log.d("CameraPrep", "autofocus
disabled")
} catch (ex: Exception) {
    Log.e("CameraCapture", "Failed to bind
camera use cases", ex)
}
}
}

```

#### 14.1.7. Study.kt

```
package pl.pw.mierzopuls.model
```

```

import com.google.gson.Gson
import pl.pw.mierzopuls.model.alg.AlgState
import java.util.*

```

```
typealias StudyDate = String
```

```

/**
 * @property id unique id
 * @property date unique id
 * @property pulse detected pulse value
 * @property times timestamps
 * @property raw raw data
 * @property filtered processed and filtered data
 * @property peaks indexes of detected peaks
 * @constructor creates an empty group.
 */

```

```

data class Study(
    val id: String = UUID.randomUUID().toString(),
    val date: String,
    val pulse: Int,
    val times: List<Int> = listOf(),
    val raw: List<Double> = listOf(),
    val filtered: List<Double> = listOf(),
    val peaks: List<Int> = listOf()
)

```

```

fun Study.fps(): Int {
    val frames = this.times.size.toDouble()
    val periodMs = (this.times.last() -
this.times.first()).toDouble()
    return (1000.0 * frames / periodMs).toInt()
}

```

```

operator fun Study.plus(list: List<Study>) =
    listOf(this, *list.toArray())

```

```

fun List<Study>.sortByDate(): List<Study> {
    val s = AlgState.Calibration(true)
    val t = s.isFingerInPlace
    return this.map {
        it to it.date.toMs()
    }.sortedBy {
        it.second
    }.map {
        it.first
    }.reversed()
}

```

```
fun Study.toJson(): String = Gson().toJson(this)
```

```

fun String.toStudy(): Study =
    Gson().fromJson(this, Study::class.java)

```

```

fun StudyDate.toDisplay(): String =
    this.replace("_", " ")

```

```

fun StudyDate.toMs(): Long {
    return this.slice(6..9).toLong() * 31556952000L
+ //year
    this.slice(3..4).toLong() * 2629800000L +
//month
    this.slice(0..1).toLong() * 86400000L + //day
    this.slice(11..12).toLong() * 3600000L +
//hour
    this.takeLast(2).toLong() * 60000L // min
}

```

```

fun Calendar.formatStudyDate(): StudyDate {
    val minutes = this[Calendar.MINUTE].let { if (it
<= 9) "0$it" else "$it" }
    val hours = this[Calendar.HOUR_OF_DAY].let {
if (it <= 9) "0$it" else "$it" }
    val month = this[Calendar.MONTH].let { if (it <=
8) "0${it + 1}" else "${it + 1}" }
    val day = this[Calendar.DAY_OF_MONTH].let {
if (it <= 9) "0$it" else "$it" }
    return "$day/$month/${this[Calendar.YEAR]}
$hours:$minutes"
}

```

#### 14.1.8. StudyRepository.kt

```
package pl.pw.mierzopuls.model
```

```

import android.content.Context
import android.net.Uri
import androidx.core.net.toFile
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import java.io.File

```

```

class StudyRepository {
    fun save(
        context: Context,
        study: Study,
        target: SaveTarget = SaveTarget.APP
    ) {
        when(target) {
            SaveTarget.APP -> {
                File(
                    context.applicationInfo.dataDir,
                    "${study.id}.json"
                ).writeText(
                    study.toJson()
                )
            }
        }
    }
}

```



```

    )
}
is SaveTarget.EXPORT -> {
    target.uri
        .ToFile()
        .writeText(study.toJson())
}
}
}

suspend fun readStudies(
    context: Context
): List<Study>? {
    return withContext(Dispatchers.IO) {
        context.applicationInfo.dataDir.let { dir ->
            File(dir).listFiles { _, s ->
                s.takeLast(5) == ".json"
            }?.map { file ->
                file.readText().toStudy()
            }
        }
    }
}

fun readStudy(context: Context, id: String): Study
{
    return context.applicationInfo.dataDir.let { dir -
>
        File(dir, "${id}.json").readText().toStudy()
    }
}
}

```

```

sealed class SaveTarget {
    object APP: SaveTarget()
    class EXPORT(val uri: Uri): SaveTarget()
}

```

#### 14.1.9. AppSettings.kt

```
package pl.pw.mierzopuls.model
```

```
import android.content.Context
import androidx.core.content.edit
```

```

class AppSetting(context: Context) {
    private val sharedPreferences =
        context.getSharedPreferences("settings",
            Context.MODE_PRIVATE)
    var showInstructionOnStart: Boolean
        get() =
            sharedPreferences.getBoolean("show_instructio
n_on_start", true)
        set(value) = sharedPreferences.edit {
            putBoolean("show_instruction_on_start",
                value)
            apply()
        }
}

```

#### 14.1.10. Camera.kt

```
package pl.pw.mierzopuls.model
```

```

import android.content.Context
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.core.content.ContextCompat
import androidx.lifecycle.Lifecycle

```

```

import androidx.lifecycle.LifecycleOwner
import androidx.lifecycle.LifecycleRegistry
import java.util.concurrent.Executor
import kotlin.coroutines.resume
import kotlin.coroutines.suspendCoroutine

```

```

suspend fun Context.getCameraProvider():
ProcessCameraProvider {
    return suspendCoroutine { continuation ->

```

```

        ProcessCameraProvider.getInstance(this).also {
            future ->
                future.addListener({
                    continuation.resume(future.get())
                }, executor)
        }
    }
}

```

```

val Context.executor: Executor
    get() = ContextCompat.getMainExecutor(this)

```

```

class CameraLifecycle : LifecycleOwner {
    private val lifecycleRegistry: LifecycleRegistry
    =
        LifecycleRegistry(this)

    init {
        lifecycleRegistry.currentState =
            Lifecycle.State.CREATED
    }

```

```

    fun doOnStart() {
        lifecycleRegistry.currentState =
            Lifecycle.State.STARTED
    }

```

```

    fun doOnResume() {
        lifecycleRegistry.currentState =
            Lifecycle.State.RESUMED
    }

```

```

    fun doOnDestroy() {
        lifecycleRegistry.currentState =
            Lifecycle.State.DESTROYED
    }

```

```

    override fun getLifecycle(): Lifecycle {
        return lifecycleRegistry
    }
}

```

#### 14.1.11. Event.kt

```
package pl.pw.mierzopuls.model
```

```

import android.content.Context
import android.os.Bundle
import android.util.Log
import com.google.firebase.analytics.FirebaseAnalytics

```

```
const val STUDY_EVENT = "event_study"
```

```

fun sendEvent(context: Context, study: Study) {
    FirebaseAnalytics.getInstance(context).logEvent(
        STUDY_EVENT, Bundle().apply {
            putInt("pulse", study.pulse)
        }
    )
}

```



```

        putString("date", study.date)
        putInt("fps", study.fps())
    }
)
Log.d(STUDY_EVENT, ""
    params:
    date = ${study.date}
    pulse = ${study.pulse}
    fps = ${study.fps()}
    """).trimIndent())
}

```

#### 14.1.12. HomeViewModel.kt

```
package pl.pw.mierzopuls.ui
```

```

import android.Manifest
import android.app.Application
import android.content.pm.PackageManager
import android.os.Vibrator
import androidx.activity.compose.ManagedActivityResultLa
    uncher
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.core.content.ContextCompat
import androidx.core.content.getSystemService
import androidx.lifecycle.ViewModelScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import org.koin.java.KoinJavaComponent.inject
import pl.pw.mierzopuls.model.alg.AlgState
import pl.pw.mierzopuls.model.alg.StudyManager
import pl.pw.mierzopuls.model.alg.processSignal
import pl.pw.mierzopuls.model.*

```

```

class HomeViewModel(
    application: Application,
    val appSetting: AppSetting,
    private val studyManager: StudyManager
) : AndroidViewModel(application) {
    private val studyRepository: StudyRepository
    by inject(StudyRepository::class.java)

```

```

    var studies: List<Study> by
mutableStateOf(listOf())
    val algState: AlgState
    get() = studyManager.algState
    val studyProgress
    get() = studyManager.progress

```

```

    var openInstruction by
mutableStateOf(appSetting.showInstructionOnSt
    art)
    var isLoading by mutableStateOf(true)

```

```

    init {
        viewModelScope.launch {
            val lastTime = System.currentTimeMillis()
            viewModelScope.launch {
                studies =
                studyRepository.readStudies(getApplication())?.so
                rtByDate() ?: listOf()
            }
            if (System.currentTimeMillis() - lastTime <
                300L) {

```

```

                delay(System.currentTimeMillis() -
                lastTime)
            }
            isLoading = false
        }
    }

    fun beginStudy(launcher:
        ManagedActivityResultLauncher<String, Boolean>)
    {
        if (!checkPermissions(launcher,
            Permissions.CAMERA)) return
        viewModelScope.launch {
            studyManager.beginStudy(
                getApplication(),

                getApplication<Application>().applicationContext.ge
                tSystemService()!!
            ) { timeStamps, values ->
                finishStudy(timeStamps, values)
            }
        }

        fun dismissStudy() {
            viewModelScope.launch {
                studyManager.dismissStudy(getApplication())
            }
        }

        fun readStudy(id: String) =
            studyRepository.readStudy(getApplication(), id)

        private fun finishStudy(timeStamps: List<Long>,
            values: List<Double>) {
            viewModelScope.launch {
                studyManager.finishStudy(getApplication())
                vibrate()
            }
            val timeStampsOffSetFromZero =
                timeStamps.map { (it - timeStamps[0]).toInt() }
                processSignal(values.toList(),
                timeStampsOffSetFromZero).let { study ->
                    studies = study + studies
                    studyRepository.save(getApplication(),
                    study)
                    studyManager.setResult(study.pulse)
                    sendEvent(getApplication(), study)
                }
        }

        private fun checkPermissions(
            launcher:
            ManagedActivityResultLauncher<String, Boolean>,
            permission: Permissions
        ): Boolean {
            val hasPermission: Boolean =
                ContextCompat.checkSelfPermission(
                    getApplication(),
                    permission.permissionString
                ) ==
                PackageManager.PERMISSION_GRANTED
            if (hasPermission) {
                return true
            } else {
                when(permission) {

```

```

        Permissions.CAMERA ->
launcher.launch(permission.permissionString)
    }
    }
    return false
}

private fun vibrate() {
    val v =
getApplication<Application>().applicationContext.ge
tSystemService<Vibrator>()
    v!!.vibrate(400)
}

enum class Permissions(val permissionString:
String) {
    CAMERA(Manifest.permission.CAMERA)
}
}

```

### 14.1.13. Home.kt

```

package pl.pw.mierzopuls.ui

import android.net.Uri
import android.util.Log
import androidx.activity.compose.rememberLauncherForA
ctivityResult
import androidx.activity.result.contract.ActivityResultContra
cts
import androidx.compose.foundation.ExperimentalFoundat
ionApi
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.Info
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import org.koin.androidx.compose.inject
import pl.pw.mierzopuls.model.alg.AlgState
import pl.pw.mierzopuls.model.Study
import pl.pw.mierzopuls.model.toJson
import pl.pw.mierzopuls.ui.components.InstructionDialog
import pl.pw.mierzopuls.ui.components.LogoPW
import pl.pw.mierzopuls.ui.components.PulseBtn

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun Home() {
    val viewModel: HomeViewModel by inject()
    val permissionLauncher =
rememberLauncherForActivityResult(
        ActivityResultContracts.RequestPermission()
    )
}

```

```

    var exportUri: Uri? by remember {
mutableStateOf(null)
    }
    var exportStudy: Study? by remember {
mutableStateOf(null)
    }
    val exportLauncher =
rememberLauncherForActivityResult(
        ActivityResultContracts.CreateDocument("**/*")
    ) { exportUri = it }
    val contentResolver =
LocalContext.current.contentResolver

    LaunchedEffect(exportUri) {
        Log.d("LaunchEff", "export uri: $exportUri")
        if (exportUri != null) {

contentResolver.openOutputStream(exportUri!!).us
e

it!!.write(exportStudy?.toJson()?.toByteArray())
        }
    }

    HistoryBottomSheet(
        studies = viewModel.studies,
        onSave = { study ->
            exportStudy = study
            exportLauncher.launch("${study.id}.json")
        }
    ) {
        Column(verticalArrangement
Arrangement.Center) {
            Box {
                LogoPW(modifier
Modifier.align(Alignment.TopCenter))
                IconButton(onClick =
viewModel.openInstruction = true) {
                    Icon(
                        modifier = Modifier.padding(8.dp),
                        imageVector = Icons.Outlined.Info,
                        contentDescription = ""
                    )
                }
            }
            PulseBtn(modifier =
Modifier
                .align(Alignment.CenterHorizontally)
                .fillMaxWidth()
                .aspectRatio(1.0f),
                algState = viewModel.algState,
                progress = viewModel.studyProgress,
                onClick = {
                    if (viewModel.algState is
AlgState.NONE) {

viewModel.beginStudy(permissionLauncher)
                } else {
                    viewModel.dismissStudy()
                }
            }
        )
    }
}

```

```

    }
}

if (viewModel.openInstruction) {
    var showAgain by remember {
        mutableStateOf(viewModel.appSetting.showInstructionOnStart)
    }
    AlertDialog(
        onDismiss = { viewModel.openInstruction = false },
        showAgain = showAgain,
        onCheckboxChange = {
            viewModel.appSetting.showInstructionOnStart = it
        }
    )
}
}

```

#### 14.1.14. History.kt

package pl.pw.mierzopuls.ui

```

import androidx.compose.animation.animateContentSize
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.launch
import pl.pw.mierzopuls.R
import pl.pw.mierzopuls.model.Study
import pl.pw.mierzopuls.model.toDisplay
import pl.pw.mierzopuls.ui.components.*
import pl.pw.mierzopuls.ui.theme.test

```

```

@OptIn(ExperimentalMaterialApi::class)
@ExperimentalFoundationApi
@Composable
fun HistoryBottomSheet(
    studies: List<Study>,
    onSave: (Study) -> Unit,
    homeContent: @Composable () -> Unit
) {
    val sheetState =

```

```

rememberBottomSheetState(initialValue =
    BottomSheetValue.Collapsed)
    val bottomSheetScaffoldState =
        rememberBottomSheetScaffoldState(
            bottomSheetState = sheetState
        )
    val coroutineScope =
        rememberCoroutineScope()
    BottomSheetScaffold(
        scaffoldState = bottomSheetScaffoldState,
        sheetPeekHeight = 80.dp,
        sheetContent = {
            LazyColumn(modifier = Modifier.fillMaxSize()) {
                stickyHeader {
                    Surface(color = test) {
                        Box(
                            Modifier
                                .fillMaxWidth()
                                .height(80.dp) {
                                    Text(
                                        modifier = Modifier
                                            .align(Alignment.Center)
                                            .fillMaxWidth()
                                            .height(80.dp)
                                            .padding(16.dp),
                                        text = stringResource(id =
                                            R.string.app_history),
                                        fontFamily = FontFamily.Default,
                                        fontWeight = FontWeight.Bold,
                                        fontSize = 20.sp,
                                        color = Color.White,
                                        textAlign = TextAlign.Center
                                    )
                                }
                            FloatingActionButton(
                                modifier = Modifier
                                    .padding(8.dp)
                                    .align(Alignment.CenterEnd),
                                backgroundColor =
                                    MaterialTheme.colors.primary,
                                onClick = {
                                    coroutineScope.launch {
                                        if (sheetState.isCollapsed) {
                                            sheetState.expand()
                                        } else {
                                            sheetState.collapse()
                                        }
                                    }
                                }
                            ) {
                                ArrowIndicator(isExpanded =
                                    bottomSheetScaffoldState.bottomSheetState.isCollapsed.not())
                            }
                        }
                    }
                }
            }
        }
        items(studies) { study ->
            StudyRow(study = study, onSave)
        }
    } {
        Box(
            Modifier
                .fillMaxSize()
                .padding(it)
                .animateContentSize()
        ) { homeContent() }
    }
}

```

```

}

@OptIn(ExperimentalMaterialApi::class)
@Composable
fun StudyRow(
    study: Study,
    onSave: (Study) -> Unit
) {
    var isExpanded by remember {
        mutableStateOf(false) }

    Card(modifier = Modifier
        .padding(16.dp)
        .fillMaxWidth()
        .height(if (isExpanded) 272.dp else 72.dp),
        elevation = 2.dp,
        onClick = {
            isExpanded = !isExpanded
        }) {
        Row(modifier = Modifier.fillMaxWidth()) {
            Row(modifier = Modifier.fillMaxWidth(),
                verticalAlignment =
                    Alignment.CenterVertically) {
                Column(horizontalAlignment =
                    Alignment.CenterHorizontally) {
                    Text(
                        modifier = Modifier.padding(16.dp),
                        text = study.date.toDisplay(),
                        color = Color.DarkGray
                    )
                }
                Row(Modifier.width(120.dp)) {
                    Icon(modifier = Modifier.padding(8.dp),
                        imageVector = Icons.Filled.Favorite,
                        tint = Color(195, 5, 60),
                        contentDescription = "")
                    Text(
                        fontSize = 17.sp,
                        modifier = Modifier.padding(8.dp),
                        text = study.pulse.let { if (it == 2) "
$it" else "$it" }
                    )
                }
                Column(modifier = Modifier.fillMaxWidth(),
                    horizontalAlignment = Alignment.End
                ) {
                    FloatingActionButton(
                        modifier = Modifier
                            .padding(8.dp)
                            .size(48.dp),
                        backgroundColor =
                            MaterialTheme.colors.primary,
                        onClick = { onSave(study) }
                    ) {
                        Icon(modifier =
                            Modifier.padding(16.dp),
                            painter = painterResource(id =
                                R.drawable.ic_save),
                            contentDescription = ""
                        )
                    }
                }
            }
        }
        if (isExpanded) {
            Row(
                modifier = Modifier

```

```

                .height(200.dp)
                .fillMaxWidth()
                .padding(8.dp),
                verticalAlignment = Alignment.Bottom
            ) {
                StudyChart(
                    modifier = Modifier
                        .height(200.dp)
                        .fillMaxWidth(),
                    study = study,
                )
            }
        }
    }
}

```

#### 14.1.15. PulseBtn.kt

```
package pl.pw.mierzopuls.ui.components
```

```

import
androidx.compose.animation.animateContentSize
import androidx.compose.foundation.layout.*
import
androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.Button
import
androidx.compose.material.CircularProgressIndicator
or
import androidx.compose.material.Icon
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import
androidx.compose.material.icons.outlined.Favorite
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.TextUnit
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import org.koin.androidx.compose.inject
import pl.pw.mierzopuls.R
import pl.pw.mierzopuls.model.alg.AlgState
import pl.pw.mierzopuls.ui.HomeViewModel
import pl.pw.mierzopuls.ui.theme.LightRose

```

```

@Composable
fun PulseBtn(modifier: Modifier = Modifier,
    algState: AlgState,
    progress: Float,
    onClick: () -> Unit
) {
    val viewModel: HomeViewModel by inject()
    Box(
        modifier = modifier.animateContentSize(),
    ) {
        if (algState != AlgState.NONE) {
            CircularProgressIndicator(
                progress,
                Modifier
                    .size(270.dp)
                    .align(Alignment.Center),
                color = LightRose,
                strokeWidth = 24.dp,
            )
        }
    }
}

```

```

    }
    Button(modifier = Modifier
        .align(Alignment.Center)
        .width(if (algState is AlgState.NONE)
180.dp else 240.dp)
        .aspectRatio(1f),
        shape = CircleShape,
        onClick = onClick
    ) {
        if (algState is AlgState.NONE || algState is
AlgState.Result) {
            Icon(
                modifier = Modifier
                    .padding(8.dp)
                    .align(Alignment.CenterVertically)
                    .size(if (algState is AlgState.Result)
24.dp else 36.dp)
                ,
                imageVector = Icons.Outlined.Favorite,
                contentDescription = ""
            )
        }
        Text(
            modifier =
Modifier.align(Alignment.CenterVertically),
            fontFamily = FontFamily.Default,
            fontWeight = FontWeight.Bold,
            fontSize = algState.buttonFontSize(),
            text = when(algState) {
                AlgState.NONE -> stringResource(id =
R.string.btn_pulse_alg_NONE)
                is AlgState.Finished,
                is AlgState.Register ->
stringResource(id =
R.string.btn_pulse_alg_REGISTRATION)
                is AlgState.Result -> stringResource(id
= R.string.btn_pulse_alg_RESULT, algState.pulse)
                is AlgState.Calibration ->
stringResource(id =
R.string.btn_pulse_alg_CALIBRATION)
            }
        )
    }
}

```

```

@Composable
fun AlgState.buttonFontSize(): TextUnit =
when(this) {
    is AlgState.Result -> 48.sp
    is AlgState.Calibration -> 24.sp
    else -> 24.sp
}

```

### 14.1.16. StudyChart.kt

```
package pl.pw.mierzopuls.ui.components
```

```

import android.graphics.Color
import
androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import
androidx.compose.ui.viewinterop.AndroidView

```

```

import
com.github.mikephil.charting.charts.CombinedChart
import
com.github.mikephil.charting.charts.CombinedChart
.DrawOrder
import
com.github.mikephil.charting.charts.ScatterChart
import
com.github.mikephil.charting.components.XAxis
import com.github.mikephil.charting.data.*
import pl.pw.mierzopuls.R
import pl.pw.mierzopuls.model.Study

```

```
@Composable
```

```

fun StudyChart(
    modifier: Modifier = Modifier,
    study: Study,
) {
    val line1 = study.filtered.zip(study.times).map {
        Entry(it.second.toFloat(), it.first.toFloat())
    }.drop(50)
    val line1Label = stringResource(id =
R.string.study_chart_line1)
    val points =
study.filtered.zip(study.times).filterIndexed { idx, _
->
        study.peaks.contains(idx) && idx > 50
    }.map {
        Entry(it.second.toFloat(), it.first.toFloat())
    }
    val pointsLabel = stringResource(id =
R.string.study_chart_points)

```

```

    AndroidView(
        modifier = modifier
            .height(250.dp)
            .fillMaxWidth(),
        factory = { context ->
            val chart = CombinedChart(context)

```

```

            chart.description.isEnabled = false
            chart.setBackgroundColor(Color.WHITE)
            chart.setDrawGridBackground(false)
            chart.setDrawBarShadow(false)
            chart.isHighlightFullBarEnabled = false
            chart.drawOrder = arrayOf(
                DrawOrder.LINE, DrawOrder.SCATTER
            )

```

```

            val lineData = LineData(
                LineDataSet(line1, line1Label).apply {
                    color = Color.rgb(100, 238, 170)
                    setDrawCircles(false)
                },
            )

```

```

            val pointsData = ScatterData(
                ScatterDataSet(points, pointsLabel).apply
{
                    color = Color.rgb(200, 4, 0)

```

```

            setScatterShape(ScatterChart.ScatterShape.SQUA
RE)
        }
    )
    val data = CombinedData()
    data.setData(lineData)

```



```

data.setData(pointsData)

chart.data = data

val xAxis: XAxis = chart.xAxis
xAxis.position =
XAxis.XAxisPosition.BOTTOM

chart.invalidate()
chart
})
}

```

### 14.1.17. AlertDialog.kt

**package** pl.pw.mierzopuls.ui.components

```

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.AlertDialog
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import pl.pw.mierzopuls.R

```

```

@Composable
fun AlertDialog(
    onDismiss: () -> Unit,
    showAgain: Boolean,
    onCheckboxChange: (Boolean) -> Unit
) {
    AlertDialog(
        title = { Text(text = stringResource(id =
R.string.instruction_title)) },
        text = {
            Column {
                InstructionContent()
                Row {
                    Checkbox(modifier =
Modifier.padding(horizontal = 8.dp),
checked = showAgain,
onCheckedChange = onCheckboxChange)
                    Text(stringResource(id =
R.string.instruction_show_again))
                }
            }
        },
        buttons = {
            Row(
                modifier = Modifier.padding(all = 8.dp),
                horizontalArrangement =
Arrangement.Center
            ) {
                Button(
                    modifier = Modifier.fillMaxWidth(),
                    onClick = { onDismiss() }
                ) { Text(stringResource(id =
R.string.instruction_ok)) }
            }
        },
        onDismissRequest = onDismiss
    )
}

```

```

@Composable
fun InstructionContent() {
    Text(modifier = Modifier.padding(12.dp),
        text = """
        Przed badaniem należy zezwolić na
        dostęp do tylnej kamery.
        Następnie zasłonić palcem obiektyw i
        lampę błyskową.
        Badanie trwa 30 s.
        """).trimIndent()
    Image(
        modifier = Modifier
            .padding(12.dp)
            .fillMaxWidth(),
        painter = painterResource(id =
R.drawable.instruction),
        contentDescription = stringResource(id =
R.string.instruction_title),
        contentScale = ContentScale.FillWidth
    )
}

```

### 14.1.18. LogoPW.kt

**package** pl.pw.mierzopuls.ui.components

```

import androidx.compose.foundation.Image
import
androidx.compose.foundation.layout.fillMaxWidth
import
androidx.compose.foundation.layout.padding
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import pl.pw.mierzopuls.R

```

```

@Composable
fun LogoPW(modifier: Modifier = Modifier) {
    Image(
        modifier = modifier
            .padding(16.dp)
            .fillMaxWidth(),
        painter = painterResource(id =
R.drawable.pw_mech_logo),
        contentDescription = stringResource(id =
R.string.app_pw_logo),
        contentScale = ContentScale.FillWidth
    )
}

```

### 14.1.19. ArrowIndicator.kt

**package** pl.pw.mierzopuls.ui.components

```

import androidx.compose.material.Icon
import androidx.compose.runtime.Composable
import androidx.compose.ui.res.painterResource
import pl.pw.mierzopuls.R

```

```

@Composable
fun ArrowIndicator(isExpanded: Boolean) {
    Icon(
        painter = painterResource(
            if (isExpanded) R.drawable.ic_arrow_hide

```

```

        else R.drawable.ic_arrow_expand
    ),
    contentDescription = null
)
}

```

### 14.1.20. Checkbox.kt

```
package pl.pw.mierzopuls.ui.components
```

```

import androidx.compose.material.Checkbox
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier

```

```

@Composable
fun Checkbox(
    modifier: Modifier = Modifier,
    title: String = "",
    checked: Boolean = false,
    onCheckedChange: (Boolean) -> Unit = {}
) {
    Checkbox(
        modifier = modifier,
        checked = checked,
        onCheckedChange = onCheckedChange
    )
    Text(
        modifier = Modifier,
        text = title
    )
}

```

### 14.1.21. AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.
com/apk/res/android"

    xmlns:tools="http://schemas.android.co
m/tools"
    package="pl.pw.mierzopuls">

    <uses-permission
android:name="android.permission.FOREG
ROUND_SERVICE" />
    <uses-permission
android:name="android.permission.CAMER
A" />
    <uses-permission
android:name="android.permission.MANAG
E_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.READ_
EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.VIBRA
TE" />

    <uses-feature
android:name="android.hardware.camera"
/>
    <uses-feature
android:name="android.hardware.camera.
autofocus" />

    <application

```

```

        android:name=".MierzoPulsApp"
        android:allowBackup="false"

        android:icon="@mipmap/ic_launcher"

        android:label="@string/app_name"

        android:roundIcon="@mipmap/ic_launcher
_round"
            android:supportsRtl="true"

        android:theme="@style/Theme.App.Starti
ng">
            <activity

                android:name=".MainActivity"

                android:configChanges="keyboardHidden|
orientation|screenSize"
                    android:exported="true"

                android:label="@string/app_name"

                android:screenOrientation="portrait"

                android:theme="@style/Theme.App.Starti
ng"

                tools:ignore="LockedOrientationActivit
y">
                    <intent-filter>
                        <action
                            android:name="android.intent.action.MA
IN" />

                        <action
                            android:name="android.content.action.D
OCUMENTS_PROVIDER" />
                        <category
                            android:name="android.intent.category.
LAUNCHER" />
                    </intent-filter>
                </activity>
            </application>

</manifest>

```