

**A G H**

**AKADEMIA GÓRNICZO-HUTNICZA  
IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,  
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

### Praca dyplomowa

*System tensometryczny do monitorowania rozkładu  
nacisku stóp biegacza na podłoże  
A strain gauge system for monitoring the pressure  
distribution of runner's feet on the ground*

Autor: *Klaudia Kromołowska*  
Kierunek studiów: *Informatyka*  
Opiekun pracy: *dr inż. Krzysztof Kluza*

Kraków, 2022



*Serdecznie dziękuję Panu dr Krzysztofowi Kluzie za poświęcony mi czas i pomoc przy realizacji niniejszej pracy, a także mojej rodzinie, za inspirację, wsparcie i wiarę we mnie.*



# **Spis treści**

<b>1. Wstęp.....</b>	<b>7</b>
1.1. Wprowadzenie.....	7
1.2. Cele pracy .....	8
1.3. Struktura pracy .....	8
1.4. Przegląd istniejących rozwiązań.....	9
<b>2. Projekt systemu.....</b>	<b>11</b>
2.1. Założenia systemu .....	11
2.2. Wymagania funkcjonalne .....	12
2.3. Wymagania niefunkcjonalne .....	13
2.4. Architektura systemu.....	14
2.5. Diagramy dla projektowanego systemu .....	17
2.5.1. Diagram przepływu danych .....	17
2.5.2. Diagram czynności.....	18
2.5.3. Diagram klas .....	20
2.5.4. Diagram sekwencji.....	23
2.6. Mapa ekranów aplikacji .....	25
2.7. Etapy tworzenia systemu.....	27
<b>3. Technologie i sprzęt wykorzystywany w tworzeniu systemu .....</b>	<b>29</b>
3.1. Sprzęt.....	29
3.2. Narzędzie STM32CubeMX oraz biblioteka HAL.....	33
3.3. Język programowania C .....	34
3.4. Środowisko programowania mikrokontrolerów .....	35
3.5. Matlab.....	35
3.6. Android Studio .....	36
3.7. Język programowania aplikacji mobilnych .....	36
3.8. Pozostałe użyte technologie i języki .....	36

3.8.1. Dagger i Hilt .....	36
3.8.2. XML jako język graficznego interfejsu użytkownika .....	37
3.8.3. SQL oraz DAO.....	37
<b>4. Implementacja.....</b>	<b>39</b>
4.1. Oprogramowanie mikrokontrolera .....	39
4.2. Interpretacja danych .....	42
4.3. Aplikacja mobilna .....	46
4.4. Wizualizacja danych.....	58
<b>5. Testy.....</b>	<b>61</b>
5.1. Testy modułowe.....	61
5.2. Testy systemowe.....	67
<b>6. Podsumowanie.....</b>	<b>69</b>
6.1. Wnioski .....	69
6.2. Perspektywy i plany rozwoju aplikacji.....	72
<b>Bibliografia .....</b>	<b>74</b>
<b>Załączniki .....</b>	<b>79</b>

# **1. Wstęp**

## **1.1. Wprowadzenie**

Początki zastosowań informatyki w medycynie sięgają już lat sześćdziesiątych ubiegłego wieku – właśnie wtedy, w roku 1964, uruchomiono pierwszy szpitalny system informatyczny [1]. Przez kolejne dekady możliwości zastosowań informatyki stały się tak szerokie, że ludzie przestali być w stanie wyobrazić sobie współczesną medycynę bez dodatku nowoczesnych technologii. Zaczynając od kwestii organizacyjnych, takich jak systemy ewidencji, elektroniczne karty i bazy danych, przez monitorowanie stanu pacjenta – rejestrując i analizując sygnały, aż po obrazowanie medyczne, czy komputerowe wspomaganie podczas operacji [1].

W obecnych czasach bardzo duży nacisk kładzie się na zdrowie, odpowiednie nawyki żywieniowe i różne formy ruchu. Według Światowej Organizacji Zdrowia przeciętny człowiek powinien wykonywać dziennie ok. 6-8 tysięcy kroków [2], co daje ok 2.5 miliona kroków rocznie – nic więc dziwnego, że najczęstsze urazy, kontuzje i bóle mają związek właśnie z nieprawidłowościami w naszym aparacie ruchu. Za częstą przyczynę problemów uważa się stopy [3] – gdy dochodzi do ich krzywienia, nieodpowiedniej rotacji, czy niewłaściwego balansu, efekt odzuwają kolejno wszystkie elementy aparatu ruchu: kolana, biodra, z czasem kręgosłup, a więc również cały organizm.

Najlepszym sposobem zapobiegania takim problemom, byłoby śledzenie każdego ruchu człowieka – stworzyłoby to szansę wczesnego wykrywania, a nawet zapobiegania kontuzjom, tym samym dając narzędzie do pracy nie tylko fizjoterapeutom, ale również wykwalifikowanym trenerom. Co więcej, wykrywając nieprawidłowości w ruchu dzieci, ludzie byliby w stanie eliminować problemy zdrowotne jeszcze na długo przed ich pierwszymi skutkami (np. wykrycie różnic w nacisku stóp małego dziecka i skorygowanie ich, skutkuje brakiem krzywienia kręgosłupa). W niniejszej pracy podjęto próbę stworzenia odpowiedniego systemu, który zbierałby dane dotyczące rozkładu nacisku na stopy oraz przedstawał je w sposób czytelny dla odbiorcy.

## 1.2. Cele pracy

Celem projektu jest implementacja systemu umożliwiającego przeprowadzenie pomiarów nacisku stóp na podłoże oraz przedstawienie tych pomiarów użytkownikowi.

Zakres pracy obejmuje następujące zagadnienia:

- przegląd literatury na temat istniejących sposobów przetwarzania sygnałów tensometrycznych pobranych ze stóp człowieka,
- wybór odpowiedniego rozwiązania sprzętowego,
- implementacja oprogramowania pobierającego pomiary nacisku,
- analiza odczytywanych pomiarów umożliwiająca przystępniejszą prezentację danych,
- implementacja aplikacji umożliwiającej wyświetlanie pomiarów urządzeniu mobilnym z oprogramowaniem Android,
- testy działania systemu i ewentualna optymalizacja aplikacji.

## 1.3. Struktura pracy

W pierwszym rozdziale znalazło się wprowadzenie do tematu pracy i zarys problematyki niżejszej pracy, innymi słowy celowość zastosowania systemów tensometrycznych z punktu widzenia dzisiejszej medycyny. Zaprezentowane zostały cel, planowany zakres prac projektowych oraz struktura pracy dyplomowej. W dalszej części zawarto informacje o innych, istniejących już na rynku komercyjnym, rozwiązaniach zaprezentowanego problemu.

Drugi rozdział opisuje projekt systemu – znalazły się tu elementy inżynierii oprogramowania. Po założeniach działania systemu, opisano wymagania funkcjonalne i niefunkcjonalne projektu oraz architekturę systemu. Na koniec dodano kilka istotnych diagramów, mapę ekranów aplikacji mobilnej oraz listę kolejnych etapów tworzenia systemu.

Kolejny, trzeci rozdział zawiera opis stosu technologicznego i sprzętu użytego w trakcie tworzenia systemu. Zwrócono uwagę zarówno na narzędzia umożliwiające tworzenie oprogramowania mikrokontrolera, takie jak język C oraz program STM32CubeMX, jak i środowisko Android Studio oraz język Kotlin, dzięki którym możliwe było stworzenie aplikacji mobilnej.

W czwartym rozdziale zawarto opis implementacji projektu. Znalazły się tu informacje o istotnych fragmentach kodu oprogramowania mikrokontrolera. Następnie zamieszczono informacje o tym, w jaki sposób dane są interpretowane i wizualizowane. Umieszczone tam również informacje o najważniejszych funkcjach, klasach i bibliotekach użytych do stworzenia aplikacji mobilnej na platformę Android.

W piątym rozdziale mieści się opis testów modułowych, sprawdzających działanie odrębnych elementów systemu oraz całej aplikacji.

W ostatnim, szóstym rozdziale umieszczone zostało podsumowanie pracy. Wspomniano zarówno o widocznych perspektywach i planach na dalszy rozwój aplikacji, jak i wnioskach wynikających z realizacji niniejszej pracy dyplomowej.

W dalszej części pracy widoczny jest spis wszystkich pozycji, które zostały wykorzystane do opisu problematyki pracy oraz listę załączników.

## 1.4. Przegląd istniejących rozwiązań

Obecnie na rynku istnieje tylko jedna, szeroko dostępna metoda diagnostyki opartej na pomiarach nacisku stóp na podłożu. W przypadku problemów z nieprawidłową postawą ciała, przewlekłymi bólami kończyn dolnych nasilającymi się podczas chodzenia, nawracającymi kontuzjami, czy w stanach po przebytych operacjach, lekarze zalecają baropodometrię [4]. Jest to komputerowe badanie stóp wykonywane w pozycji stojącej lub podczas chodzenia. Badanie polega na zmierzeniu sił reakcji podłożu na nacisk stóp pacjenta. Dostępne są maty do długości około dwóch metrów, a więc w praktyce to badanie nie umożliwia pomiaru np. w czasie biegu, gdy stawiane przez człowieka kroki są dużo dłuższe (długość kroku w czasie biegu rośnie o około połowę w porównaniu do spaceru [5]). Trudno też wyobrazić sobie wykonywanie takiego badania np. kilka razy w miesiącu, ponieważ jego koszt to każdorazowy wydatek ok. 130zł [6]. Dodatkowo należałoby wziąć pod uwagę fakt że, w przypadku sportowców, sposób stawiania stóp (a tym samym wyniki badania) będą znaczco różnić się po dużym wysiłku oraz pod koniec lekkiego treningu, który nie wpłynie w sposób istotny na zmęczenie mięśni [7].

Nasuwa się więc prosty wniosek – człowiek, w szczególności ktoś uprawiający sport, który chciałby regularnie badać swoją postawę musi znaleźć inne rozwiązanie. Z pomocą przyszła firma Digitsole [8] i jej inteligentne wkładki do butów z aplikacją. Według opisu producenta wkładki spełniają wszystkie istotne funkcjonalności – mierzą balans stopy, długość i stabilność kroku, strefy podparcia, wykrywają pronację/supinację, monitorują czas lotu, czas kontaktu. Wbudowany GPS zapewnia wygodne przeglądanie statystyk, a na dodatek producent zapowiada spersonalizowane wsparcie i wiele porad jak poprawić swoje biegi. Niestety według opinii użytkowników produkt ma bardzo dużo wad, co skutkuje finalną oceną 1.9/5 [9]. Użytkownicy zwracają uwagę na to, że tylko jedna wkładka jest „inteligentna”, a druga nie zawiera żadnej elektroniki, więc w rzeczywistości mierzona jest tylko jedna stopa. Aby zarejestrować bieg należy za każdym razem kalibrować wkładkę przez około 10 min, dodatkowo z uwagi

na to, że cały sprzęt mieści się we wnętrzu wkładki, są one dość twarde i niewygodne. W opiniach można również znaleźć informacje o zbyt małej baterii oraz niezbyt dokładnych pomiarach – wszystko to za dość przystępna cenę ok. 400 zł [8].

W poszukiwaniu lepszych rozwiązań dotarto aż do brytyjskiej firmy Arion [10]. Stworzona przez ten zespół wkładka zawiera 8 czujników nacisku, połączenie z telefonem poprzez Bluetooth. W aplikacji mobilnej można znaleźć informacje, takie jak: miejsce stopy, którym naciskamy po podłożę, długość kroku, kadencje, równowagę, czas kontaktu, tempo, stabilność, czas lotu, wskaźnik obciążenia mięśni. Statystyki pokazywane są w aplikacji w czasie rzeczywistym lub przez słuchawki sparowane ze smartfonem. Firma zakłada dwie opcje treningu, albo samodzielnie analizuje się swoje zarejestrowane wyniki, albo używa aplikacji jako trenera i to sztuczna inteligencja kieruje treningiem. W przeciwieństwie do poprzedniego produktu, konieczne jest tylko jednorazowe 10-minutowe kalibrowanie wkładek. Jedyny problem, na jaki zwracają uwagę użytkownicy, to uciążliwość aplikacji w przypadku nieodpowiedniego biegu. Aplikacja wysyła bowiem informacje w czasie rzeczywistym za każdym razem, gdy użytkownik znajduje się powyżej lub poniżej docelowego przedziału (np. długości kroku). Należy dodać, że wspomniana wyżej firma nie należy do popularnych, oferuje jedynie 4 różne rozmiary wkładek, każde za cenę ok. 1200 zł [10].

## 2. Projekt systemu

System monitorowania rozkładu nacisku połączony z aplikacją mobilną to typowy problem dla inżynierii oprogramowania. Aby stworzyć działający system, należy określić najpierw wymagania, zaprojektować go, a następnie przejść do implementacji i testów.

Tworząc projekt systemu rozpoczęto od podstawowych elementów inżynierii oprogramowania. W kolejnych sekcjach umieszczono charakterystykę użytkowników systemu oraz opis głównych procesów zachodzących w systemie. Określono również wymagania, jakim powinien sprostać system oraz opisano architekturę systemu, dodając zarówno jej graficzną reprezentację, jak i diagram przypadków użycia. W dalszej kolejności w rozdziale znalazły się najważniejsze diagramy – czynności, klas, sekwencji i przepływu danych, a także mapa ekranów aplikacji – ostatnia część projektowania części mobilnej. Rozdział zakończono opisem kolejnych etapów tworzenia niniejszego systemu.

### 2.1. Założenia systemu

#### Aktorzy:

**Wariant 1** – Aktorem jest użytkownik systemu – osoba która posiada wkładki i zainstalowaną aplikację mobilną. Ma dostęp do wszystkich funkcjonalności systemu bez konieczności logowania do aplikacji.

**Wariant 2** – W pewnych sytuacjach, zamiast jednego aktora może występować dwoje: osoba, która nosi wkładki w butach oraz druga, która ma podgląd na dane widoczne w aplikacji. Taka sytuacja wystąpi w przypadku, gdy rodzic lub trener będą chcieli śledzić ruch swojego dziecka, czy zawodnika.

#### Procesy:

**Jednorazowa konfiguracja** polegająca na włączeniu aplikacji, podaniu danych takich jak imię i waga oraz zezwoleniu na dostęp do Bluetooth i lokalizacji.

**Codzienne korzystanie z aplikacji** polegające na założeniu wkładek, włączeniu aplikacji oraz przejściu do wybranej przez użytkownika zakładki.

## 2.2. Wymagania funkcjonalne

Zgodnie z zaprezentowanym w poprzednim rozdziale celem pracy, przygotowany system ma umożliwiać analizę rozkładu sił nacisku na stopy. Na podstawie wspomnianej informacji, możliwe jest wyróżnienie kilku podstawowych funkcjonalności systemu, takich jak: pobieranie danych wejściowych przez mikrokontroler, przesyłanie danych przez bluetooth do aplikacji, przekształcanie, zapisywanie ich do bazy oraz graficzna prezentacja wyników. Tabele 2.1 i 2.2, zawierają odpowiednio: spis wymagań funkcjonalnych całego systemu oraz aplikacji mobilnej.

L.p.	Opis wymagania
1	Odczytywanie pomiaru z czujnika tensometrycznego
2	Odczytywanie wielu pomiarów jednocześnie
3	Wysyłanie danych pobranych z mikrokontrolera przez Bluetooth
4	Transmisja danych w czasie rzeczywistym
5	Aplikacja mobilna z możliwością zainstalowania w telefonie z systemem Android
7	Obliczenie funkcji nacisku na podstawie danych odbieranych przez mikrokontroler
8	Prezentowanie wyników pomiarów nacisku w skali procentowej
9	Wyświetlenie widoku wkładek w telefonie
10	Zapis danych z czujników nacisku zebranych w czasie treningu
11	Możliwość działania aplikacji w tle
12	Możliwość korzystania z aplikacji bez dostępu do internetu

**Tabela 2.1.** Wymagania funkcjonalne – ogólne, dotyczące całego systemu

L.p.	Opis wymagania
1	Uruchomienie i zatrzymanie animacji nacisku
2	Dodanie nowego treningu, zatrzymanie go, wznowienie, zapisanie lub anulowanie,
3	Śledzenie trasy na mapach Google podczas treningu
4	Zapis parametrów każdego treningu takich jak: trasa, czas trwania treningu, średnia prędkość, przebyty dystans oraz średni i maksymalny nacisk wywierany przez każdą ze stóp
5	Przeglądanie parametrów zapisanych w poprzednich biegach
6	Możliwość usunięcia zisanego biegu
7	Możliwość sortowania zapisanych biegów po wybranym parametrze
9	Możliwość zmiany kolorystyki animacji wkładek, imienia i wagi użytkownika
10	Możliwość zatrzymania i wznowienia biegu z poziomu paska powiadomień

**Tabela 2.2.** Szczegółowe wymagania funkcjonalne aplikacji mobilnej

## 2.3. Wymagania niefunkcjonalne

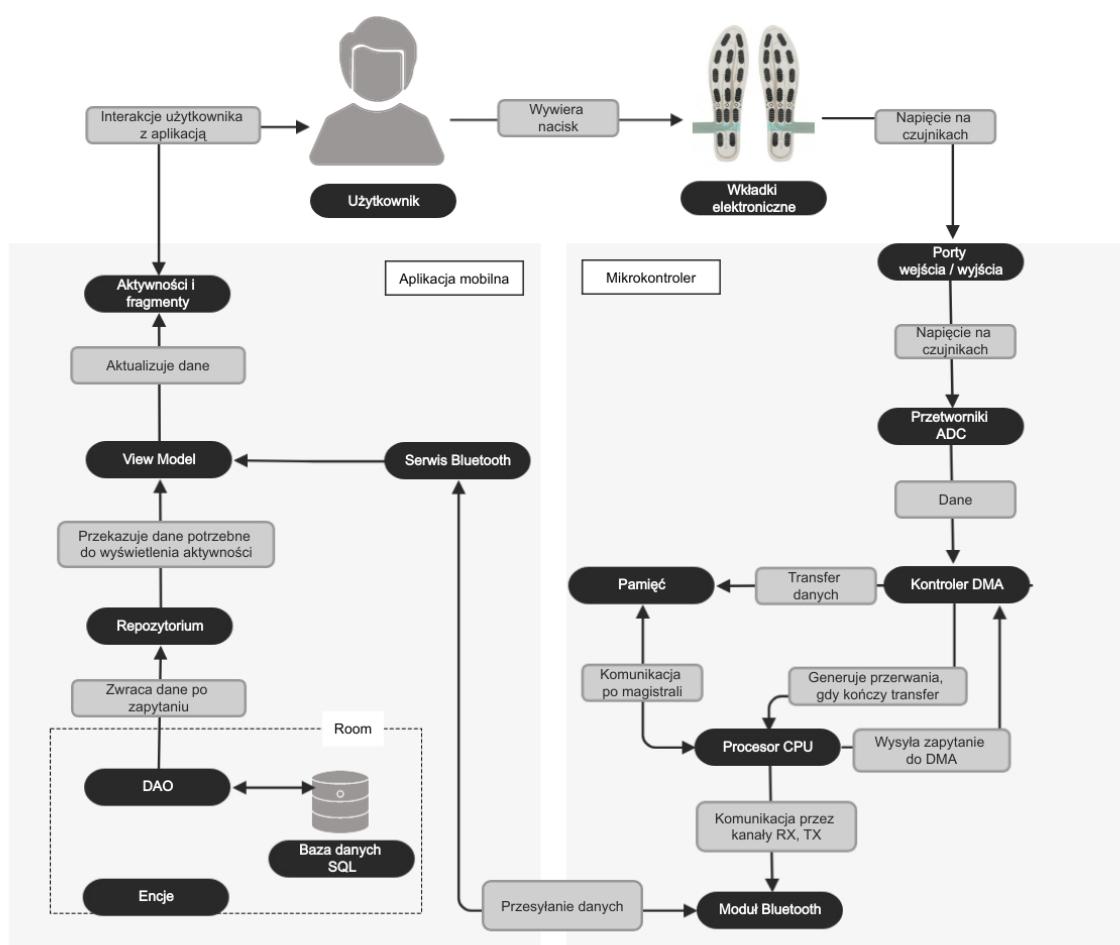
W tej sekcji opisano pozostałe wymagania wobec systemu, które muszą być spełnione – skupiające się przede wszystkim na sposobie pracy systemu, jego bezpieczeństwie, szybkości itp. W Tabeli 2.3 zawarto opis wymagań niefunkcjonalnych:

<b>Bezpieczeństwo</b>	Mikrokontroler powinien przesyłać dane przez Bluetooth jedynie do sparowanego urządzenia. Urządzenie mobilne nie będzie wysyłać danych do innych urządzeń unikając w tej sposób dodatkowego ryzyka.
<b>Dostępność</b>	System powinien być dostępny zawsze, gdy użytkownik ma dostęp do smartfona z modułem Bluetooth i urządzenia z wkładkami.
<b>Użyteczność</b>	System powstaje na potrzeby sportowców oraz amatorów, którzy chcą zadbać o swoje zdrowie poprzez monitorowanie rozkładu nacisku stóp na podłoże.
<b>Szybkość</b>	Uruchomienie aplikacji powinno trwać nie dłużej niż 3 sekundy. Wyświetlanie danych powinno być odświeżanie minimum 400 razy na minutę.
<b>Łatwość użytkowania</b>	Aplikacja wymaga czytelnego interfejsu użytkownika. Istotna jest estetyka i prostota obsługi – aplikacja powinna być obsługiwana w sposób intuicyjny.
<b>Zgodność</b>	Z uwagi na to, że będzie to pierwsza wersja aplikacji nie będzie trzeba dbać o zgodność z poprzednimi wersjami. Projekt powinien być wykonany w sposób umożliwiający łatwą edycję systemu oraz napisany w języku Kotlin w celu lepszej przejrzystości.
<b>Kompletność</b>	System będzie uznany za kompletny, jeżeli będzie w stanie pobrać dane, przekształcić je i wyświetlić w aplikacji mobilnej.
<b>Kompatybilność</b>	Aplikacja powinna działać w pełni na smartfonach z Androidem w wersji 9.0 oraz być wygodna do używania na ekranach w rozmiarze od 5 do 6.5 cali.

**Tabela 2.3.** Wymagania niefunkcjonalne

## 2.4. Architektura systemu

Po określaniu wymagań, kolejnym istotnym elementem pozwalającym na stworzenie kompletnego modelu systemu, jest zaplanowanie architektury całego systemu. Powinna ona dotyczyć przede wszystkim struktury, wzajemnych oddziaływań podsystemów oraz funkcjonalności systemu i jego możliwych interakcji z użytkownikiem. W celu lepszego zobrazowania przedstawionego zagadnienia, przygotowano graficzną reprezentację przypadków użycia (Rys. 2.1) oraz diagram przypadków użycia (Rys. 2.2).



Rys. 2.1. Architektura systemu

Aby lepiej zrozumieć zamieszczone na Rys. 2.1 elementy architektury, model uzupełniono o słownik najważniejszych pojęć:

**View Model** – to centrum komunikacji między interfejsem użytkowika, a repozytorium, jednocześnie buforuje najnowsze dane z tzw. LiveData i przekazuje do repozytorium.

**Repozytorium** – zarządza wieloma źródłami danych – w tym przypadku są to serwisy lokalizacji i bluetooth oraz baza danych.

**Room** – to warstwa bazy danych, która zapewnia ich łatwiejsze przechowywanie, używa DAO do wysyłania zapytań do bazy danych SQL

**DAO** – ang. *Data Access Object* – obiekt dostępu do danych. Mapuje zapytania SQL na funkcje, które mogą być wywoływanie w programie.

**Encje** – klasy z adnotacjami, które opisują tabelę bazy danych.

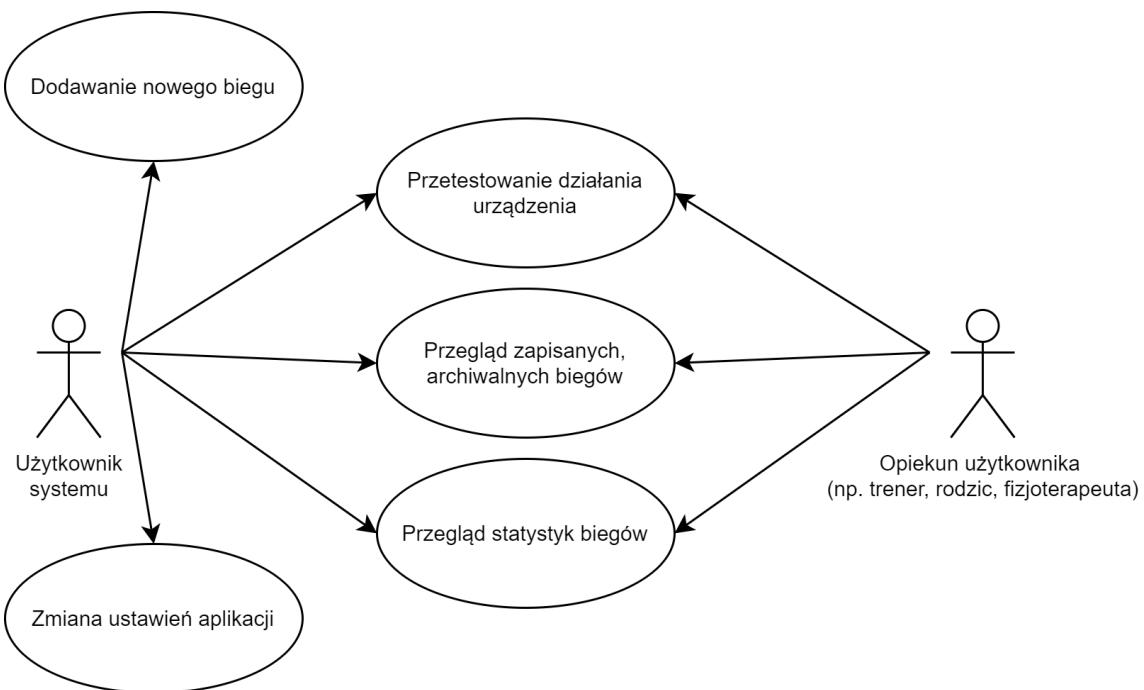
Wyróżniamy dwa podsystemy – część związaną z mikrokontrolerem oraz tą z aplikacją mobilną. Wszystko zaczyna się od użytkownika, który wywiera pewien nacisk na elektroniczne wkładki i chce odczytać w aplikacji mobilnej informację, w jaki sposób ten nacisk jest rozkładany na poszczególne części każdej ze stóp.

Czujniki tensometryczne umieszczone we wkładkach ulegają odkształceniom pod wpływem nacisku, tym samym zmieniając swoją rezystancję. Na odpowiednich portach mikrokontrolera pojawiają się pewne analogowe wartości, które za pomocą przetworników ADC (opisane szczegółowo w rozdziale 3.1) zamieniane są na postać cyfrową. DMA (również opisany szczegółowo w rozdziale 3.1) kontroluje transfer danych i przesyła dane do pamięci odciążając główny procesor – w ten sposób pełni więc rolę bezpośredniego łącznika między przetwornikami ADC, a pamięcią. Procesor CPU (ang. *Central Processing Unit*) odczytuje dane z pamięci i przekazuje je do modułu Bluetooth za pomocą kanałów RX/TX. Moduł Bluetooth wysyła dane, które następnie są przechwytywane przez serwis Bluetooth w aplikacji mobilnej.

Dane z serwisu Bluetooth przekazywane są do View Model do tzw. LiveData, czyli managerów danych, które służą do obserwowania zmian w danych i aktualizowania widoku. Pozostałe dane potrzebne do widoku powinny być pobrane z repozytorium. Aby dostać odpowiednie dane, repozytorium wysyła zapytania do bazy danych używając zdefiniowanych w DAO metod. Po wczytaniu odpowiednich danych, dane z repozytorium przekazywane są do View Model. Aktywności lub fragmenty interfejsu użytkownika reagują na zmiany w View Model i uaktualniają widok, tym samym odpowiadając na interakcję użytkownika.

Po opisaniu architektury, wiadomo już jak wygląda struktura systemu i oddziaływanie między jego poszczególnymi częściami. W kolejnej części skupiono się na przypadkach użycia, aby zobrazować kluczowe funkcjonalności systemu które są widoczne od strony użytkownika.

Jak widać na diagramie zaprezentowanym na Rys. 2.2, istnieje pięć podstawowych przypadków użycia. Poniższa lista zawiera zarówno nazwy przypadków użycia, jak i funkcjonalności, które się w nich zawierają.



Rys. 2.2. Diagram przypadków użycia

1. „Przetestowanie działania urządzenia” – użytkownik chce sprawdzić, w jaki sposób działa jego urządzenie lub jak rozkłada się nacisk stóp na powierzchnię wkładek. Użytkownik może uruchomić i w dowolnym momencie zatrzymać wizualizację rozkładu nacisku. Istnieje też możliwość używania tej części aplikacji przez opiekuna użytkownika (to jest np. rodzica, trenera lub fizjoterapeuty), w czasie, gdy użytkownik korzysta z urządzenia.
2. „Dodawanie nowego biegu” – użytkownik systemu chce dodać nowy bieg. Po naciśnięciu przycisku „START” rozpoczyna się śledzenie lokalizacji oraz uruchamia się zegar widoczny na ekranie. Użytkownik w każdym momencie może wstrzymać pomiary, anulować bieg lub go zakończyć, tym samym zapisując do bazy danych.
3. „Przegląd archiwalnych biegów” – gdy użytkownik chce przejrzeć swoje dotychczasowe treningi. Użytkownik ma możliwość posortowania biegów według wybranego kryterium, a także usunięcie każdego z nich. Do tej części aplikacji ma dostęp również opiekun użytkownika – może on przeglądać osiągnięcia podopiecznego wg wybranego kryterium.
4. „Przegląd statystyk” – gdy użytkownik lub opiekun chce zobaczyć podsumowanie dotychczasowych treningów. Użytkownik widzi podsumowane parametry ze wszystkich biegów, np. sumaryczną ilość wszystkich przebytych kilometrów, a także wykresy zawierające średnią prędkość i średni nacisk każdej ze stóp w kolejnych biegach. Użytkownik może przybliżyć lub oddalić widok wykresu za pomocą gestu, aby zobaczyć szczegółowo.

5. „Zmiana ustawień aplikacji” – użytkownik chce spersonalizować aplikację. Istnieje możliwość zmiany koloru prezentacji wyświetlanej w zakładce związanej z testowaniem urządzenia, a także zmiana zapisanych wcześniej danych osobowych.

## 2.5. Diagramy dla projektowanego systemu

Projekt, który jest tematem niniejszej pracy jest dość złożony. Aby lepiej uporać się z jego skalą, przygotowano kilka diagramów, które w uproszczony sposób prezentują poszczególne elementy całego systemu. Jako pierwszy zostanie zaprezentowany diagram przepływu danych – dotyczy on bowiem obu podsystemów – zarówno części związanej z mikrokontrolerem, jak i aplikacją mobilną. Następnie przedstawione zostaną: diagram czynności, diagram klas i diagram sekwencji, które skupią się na projektowaniu aplikacji mobilnej.

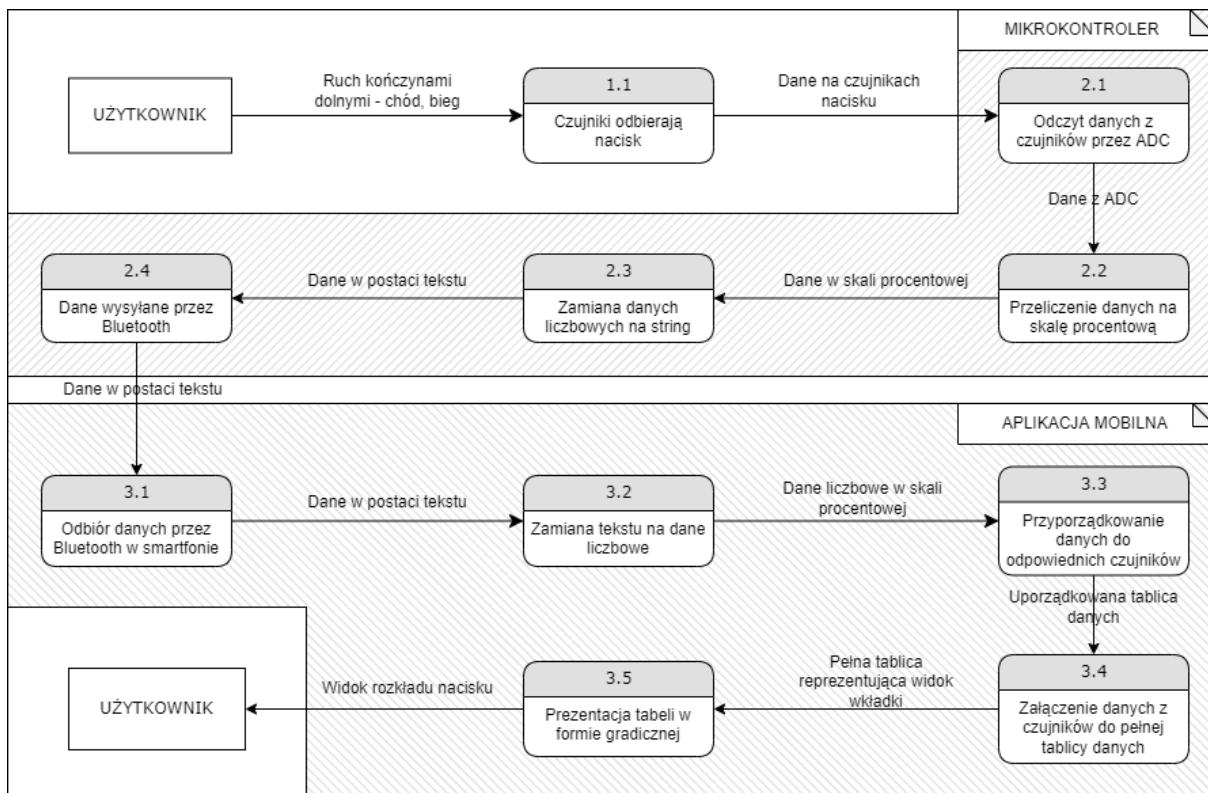
### 2.5.1. Diagram przepływu danych

Diagram 2.3 prezentuje ścieżkę, jaką przechodzą dane od momentu wykonania nacisku na wkładki przez człowieka, aż do chwili wyświetlenia ich na ekranie smartfona, w przypadku gdy użytkownik zdecyduje się przetestować urządzenie bez zapisywania parametrów do bazy danych. Mierzone napięcia na poszczególnych pinach mikrokontrolera docierają do pamięci dzięki przetwornikom ADC (Analog to Digital Converter). Zamieniają one wartość analogową (napięcie) na postać cyfrową. Wszystko odbywa się z pomocą bezpośredniego przydziału pamięci DMA (Direct Memory Access) w celu uniknięcia przeciążeń procesora.

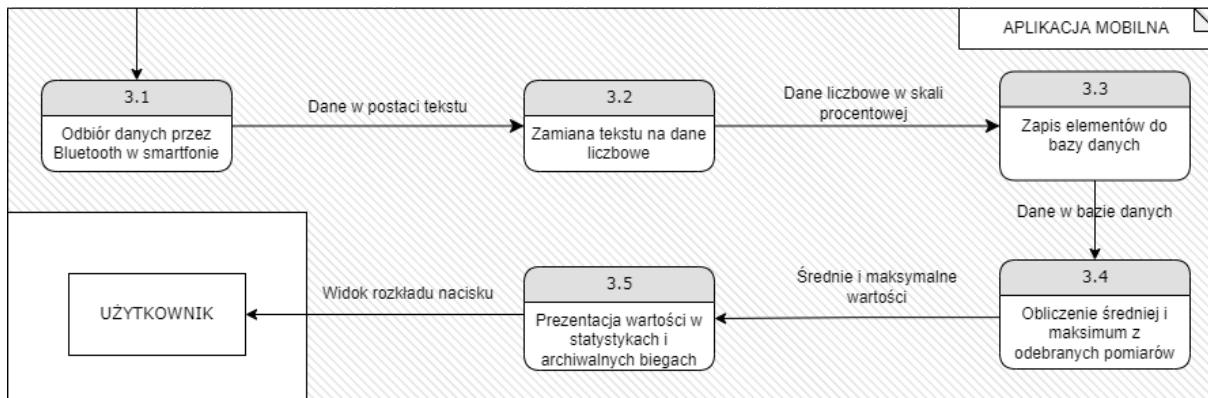
Dane są przekształcane do skali procentowej (0-100%), a następnie zamieniane na tekst i wysyłane poprzez Bluetooth. W aplikacji zapisana jest tablica z konfiguracją widoku stopy – przypisane są tam adresy, w których znajdują się czujniki i te do których nie sięga wkładka.

Dane w postaci tekstu docierają do aplikacji mobilnej, tam są zamieniane na wartości liczbowej, uporządkowywane i załączane do tablicy z konfiguracją widoku stopy. W dalszej kolejności na podstawie wyżej wspomnianej tablicy powstaje grafika – widok rozkładu nacisku stopy, który następnie jest prezentowany użytkownikowi.

Istnieje jeszcze jedna możliwość – użytkownik uruchamia aplikację i decyduje się na prowadzenie treningu – w ten sposób dane, zamiast być prezentowane w formie graficznej, powinny zapisać się do bazy danych. Diagram przepływu danych w tym przypadku będzie różnił się jedynie częścią związaną z aplikacją mobilną, co prezentuje Rys. 2.4.



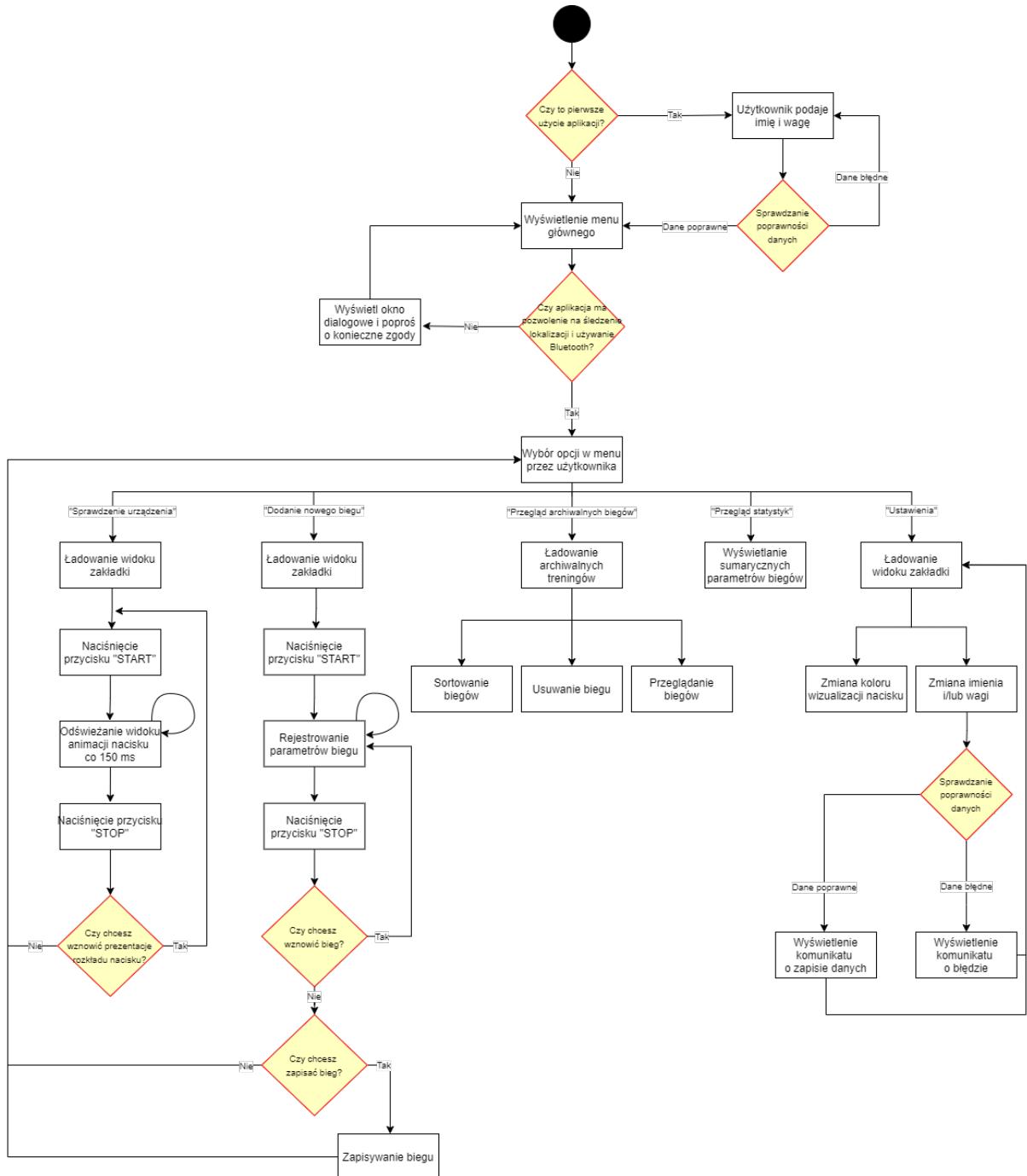
Rys. 2.3. Diagram przepływu danych w systemie



Rys. 2.4. Diagram przepływu danych – opcja nr 2

## 2.5.2. Diagram czynności

Diagram czynności, nazywany również diagramem aktywności, jest jednym z najważniejszych rodzajów diagramów. Celem jego użycia jest przedstawienie kolejnych kroków, które są wykonywane przez dany fragment systemu. Diagram czynności prezentujący niniejszy system znajduje się na Rys. 2.5.



Rys. 2.5. Diagram czynności dla aplikacji mobilnej

Pierwszą rzeczą jaka sprawdza system zaraz po uruchomieniu aplikacji jest to, czy jest to pierwsze jej uruchomienie na danym urządzeniu mobilnym. Jeżeli tak, aplikacja poprosi o wprowadzenie imienia użytkownika i jego obecnej wagi oraz sprawdzi, czy są poprawne (jeżeli nie, wyświetli komunikat i ponownie poprosi o odpowiednie dane). Zarówno w przypadku uzupełnienia danych poprawnymi wartościami, jak i kolejnego użycia aplikacji, wyświetcone zostanie menu główne. Jeżeli aplikacja nie ma jeszcze uprawnień do pobierania lokalizacji

i łączenia poprzez Bluetooth, zostanie wyświetcone odpowiednie okno dialogowe, informujące o konieczności wyrażenia odpowiednich zgód w celu dalszego korzystania z aplikacji. Następnie użytkownik wybierze jedną z pięciu opcji menu:

1. „Sprawdzanie urządzenia”

Po załadowaniu widoku i naciśnięciu przycisku „START” rozpocznie się wyświetlanie grafiki prezentującej rozkład nacisku. Widok jest odświeżany cyklicznie co 150ms, aż do momentu naciśnięcia przycisku „STOP”. Użytkownik może zakończyć korzystanie z tej części aplikacji, lub ponownie nacisnąć przycisk „START”.

2. „Dodanie nowego biegu”

Po załadowaniu widoku i naciśnięciu przycisku „START” rozpocznie się rysowanie na mapie Google widoku przebytej trasy i mierzenie czasu na widocznym zegarze. W czasie biegu będą rejestrowane parametry takie jak: dystans, lokalizacja, czas, średni i maksymalny nacisk każdej ze stóp. Po naciśnięciu przycisku „STOP” użytkownik może wznowić bieg, anulować usuwając zebrane dane oraz zakończyć, zapisując dane do bazy danych.

3. „Przegląd archiwalnych biegów”

Po załadowaniu widoku użytkownik może przeglądać biegi za pomocą paska przewijania, usunąć wybrany bieg lub posortować według dowolnego z zapisywanych parametrów.

4. „Przegląd statystyk”

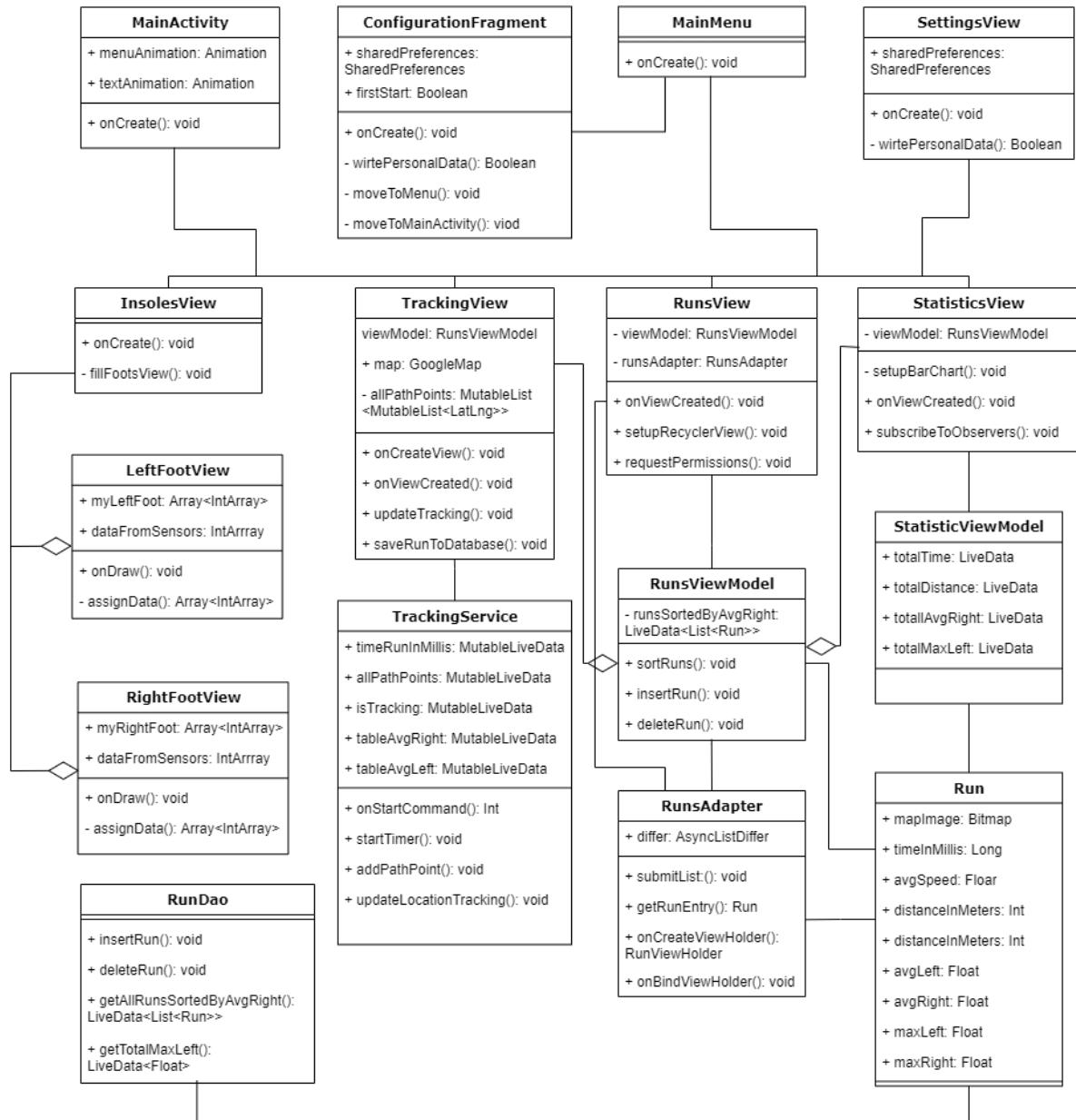
Wyświetlają się podsumowane parametry zebranych biegów, takie jak: sumaryczny przebyty dystans, łączny czas treningów, sumaryczna średnia prędkość i średni nacisk, a także całkowity maksymalny zarejestrowany nacisk każdej ze stóp.

5. „Ustawienia”

Po załadowaniu widoku użytkownik może dokonać zmiany koloru wizualizacji nacisku poprzez kliknięcie odpowiedniego przycisku lub dokonać zmiany wprowadzonych danych osobowych. Następnie aplikacja sprawdzi, czy wprowadzone dane są poprawne (czy spełniają odpowiednie kryteria) i wyświetli odpowiedni komunikat.

### 2.5.3. Diagram klas

Kolejnym diagramem, który jest najczęściej używany w projektowaniu systemów, jest diagram klas. Zawarty na Rys. 2.6 diagram obrazuje zbiór najważniejszych klas oraz związki między nimi, a także kluczowe dla każdej klasy metody i charakterystyczne obiekty.



Rys. 2.6. Diagram klas dla aplikacji mobilnej

- Klasa **ConfigurationFragment** uruchamia się jedynie przy pierwszym użyciu aplikacji. Prowadzi ona do klasy **MainMenu**, więc tym samym ona również wyświetla się tylko raz w całym cyklu życia aplikacji.
- Klasa **MainActivity** to główny widok menu, który pojawia się przy każdym (poza pierwszym) uruchomieniu aplikacji. Zarówno klasa **MainActivity**, jak i klasa **MainMenu**, prowadzi do wybranej spośród z pięciu dostępnych zakładek: **InsolesView**, **TrackingView**, **RunsView**, **StatisticView** oraz **SettingsView**.

- Klasa **InsolesView**, prezentująca wizualizację nacisku, zawiera metodę `fillFootsView()`, która odwołuje się do głównych metod (`onDraw()`) metod `LeftFootView` i `RightFootView`.
  - Klasy **RightFootView** i **LeftFootView** działają w podobny sposób – każda z nich zawiera tablicę `dataFromSensors` oraz `myRightFoot/myLeftFoot`, które odpowiednio zawierają dane pobrane z czujników nacisku oraz całą dwuwymiarową tablicę danych zawierającą dane o nacisku na każdym miejscu wkładki. Metoda `onDraw()` odpowiada za rysowanie rozkładu nacisku.
- Klasa **TrackingView** związana jest z dodawaniem nowego biegu – zawiera między innymi obiekt `allPathPoints`, który zawiera listę kolejnych punktów lokalizacji, którą przebył użytkownik w czasie treningu. Zawiera metodę `updateTracking()` oraz `saveRunToDatabase()`, która zapisuje bieg do bazy danych.
  - Klasa **TrackingService** zawiera zmienne typu `MutableLiveData` takie jak: `allPathPoint`, `isTracking`, `timeRunInMillis`, a także `tableAvgRight/Left`, `tableMaxRight/Left` – przechowują one odpowiednio tablice dotychczasowych punktów lokalizacji, informację o tym czy bieg jest kontynuowany, czas w milisekundach, tablice średnich i maksymalnych wartości pomiarów nacisku każdej ze stóp.
- Klasa **RunsView** związana jest z wyświetlaniem archiwalnych biegów. Zawiera metodę `setupRecyclerView()`, która odpowiada za tworzenie kolejnych biegów na pasku przewijania. Zawiera obiekt będący instancją klasy `RunsViewMode`, dodatkowo jest powiązana z `RunsAdapter`, `Run` oraz `RunsDaoConstructor`.
  - Klasa **RunsViewModel** wywołuje odpowiednie metody związane z klasą `Run` i `RunDaoConstructor`, takie jak np. `sortRuns()`, `invertRun()` czy `deleteRun()`, które odpowiednio sortują (po wybranym parametrze), dodają lub usuwają bieg.
  - Klasa **RunsAdapter** zawiera metody związane z tzw. `RecyclerView` – jest to fragment ekranu z paskiem przewijania, który zawiera wszystkie dotychczasowe biegi.
  - Klasa **Run** to klasa danych – zawiera ona jedynie obiekty (dane zebrane podczas biegu) używane przez inne klasy. Nie zawiera żadnych metod i nie operuje na danych.
  - Klasa **RunsDao** to klasa zawierająca metody, które wiążą się bezpośrednio z poleceniami w języku SQL. Jest to np. `insertRun()`, `deleteRun()`, `getTotalMaxLeft()` która pobiera sumaryczną maksymalną wartość nacisku lewej stopy, czy `getAllRunsSortedByAvgRight()`, który pobiera z bazy danych wszystkie dotychczasowe biegi, sortując je według średniego nacisku prawej stopy.

- Klasa **StatisticView** to klasa odpowiadająca za wyświetlanie podsumowania biegów irysowanie wykresów – stąd metody takie jak `setupBarChart()`.
- Klasa **StatisticViewModel** to klasa która odpowiada za przechowanie sumarycznych danych, widoczne są w niej obiekty takie jak `totalTime()`, czy `totalDistance()`
- Klasa **SettingsView** związana jest z widokiem ustawień. Metoda `onCreate()` zapewni odpowiednie wyświetlanie widoku i wszystkich przycisków, a `writePersonalData()` sprawdzanie poprawności wpisanych danych osobowych.

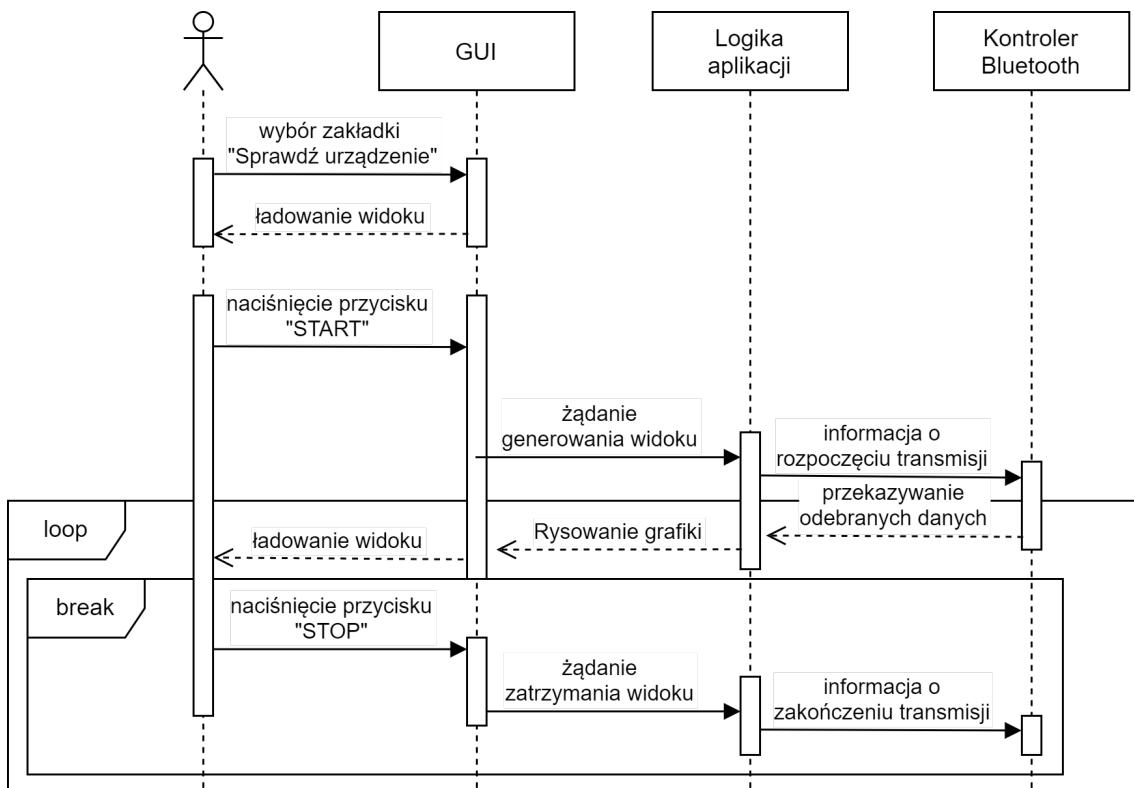
#### 2.5.4. Diagram sekwencji

Ostatnim diagramem, który wspiera proces modelowania systemu, jest diagram sekwencji. Ukazuje on zależności między obiektami z uwzględnieniem czasu potrzebnego na wykonanie operacji. Pozwala on na zaprezentowanie zachowań systemu w kontekście różnych przypadków użycia (dokładniej, w kontekście scenariuszy konkretnych przypadków użycia).

W związku z dużą złożonością każdej ze ścieżek zdecydowano o skupieniu się na dwóch, najistotniejszych i najbardziej charakterystycznych dla projektu przypadków użycia – "Przetestowanie działania urządzenia" (Rys. 2.7) oraz "Dodawanie nowego biegu" (Rys. 2.8).

Skrót GUI (ang. *Graphical User Interface* – oznacza graficzny interfejs użytkownika, czyli widok aplikacji prezentowany użytkownikowi. Dla uproszczenia wszystkie funkcje pomocnicze zastąpiono obiektem Łogika aplikacji". Fragment oznaczony słowem „loop” oznacza pętlę, a „break” jej przerwanie. W przypadku testowania działania urządzenia wyróżniamy cztery obiekty: użytkownika, GUI, logikę aplikacji i kontroler Bluetooth.

Po wyborze tej zakładki przez użytkownika, GUI ładuje jej widok (w tym przycisk „START”). Użytkownik naciska przycisk, a GUI wysyła do aplikacji żądanie wygenerowania odpowiedniego widoku. Aplikacja wysyła do kontrolera Bluetooth informację o rozpoczęciu transmisji, a on w odpowiedzi zwraca odebrane dane. Dzięki temu logika aplikacji rysuje grafikę, a GUI może załadować odpowiedni widok. Wszystko to dzieje się w pętli – cyklicznie przetwarzane są nowe dane, rysowana nowa grafika i ładowany jest nowy widok – aż do momentu naciśnięcia przycisk „STOP” przez użytkownika. W takiej sytuacji GUI przesyła do logiki aplikacji żądanie zatrzymania rysowania grafiki, a aplikacja przesyła do kontrolera Bluetooth informację o zakończeniu transmisji.



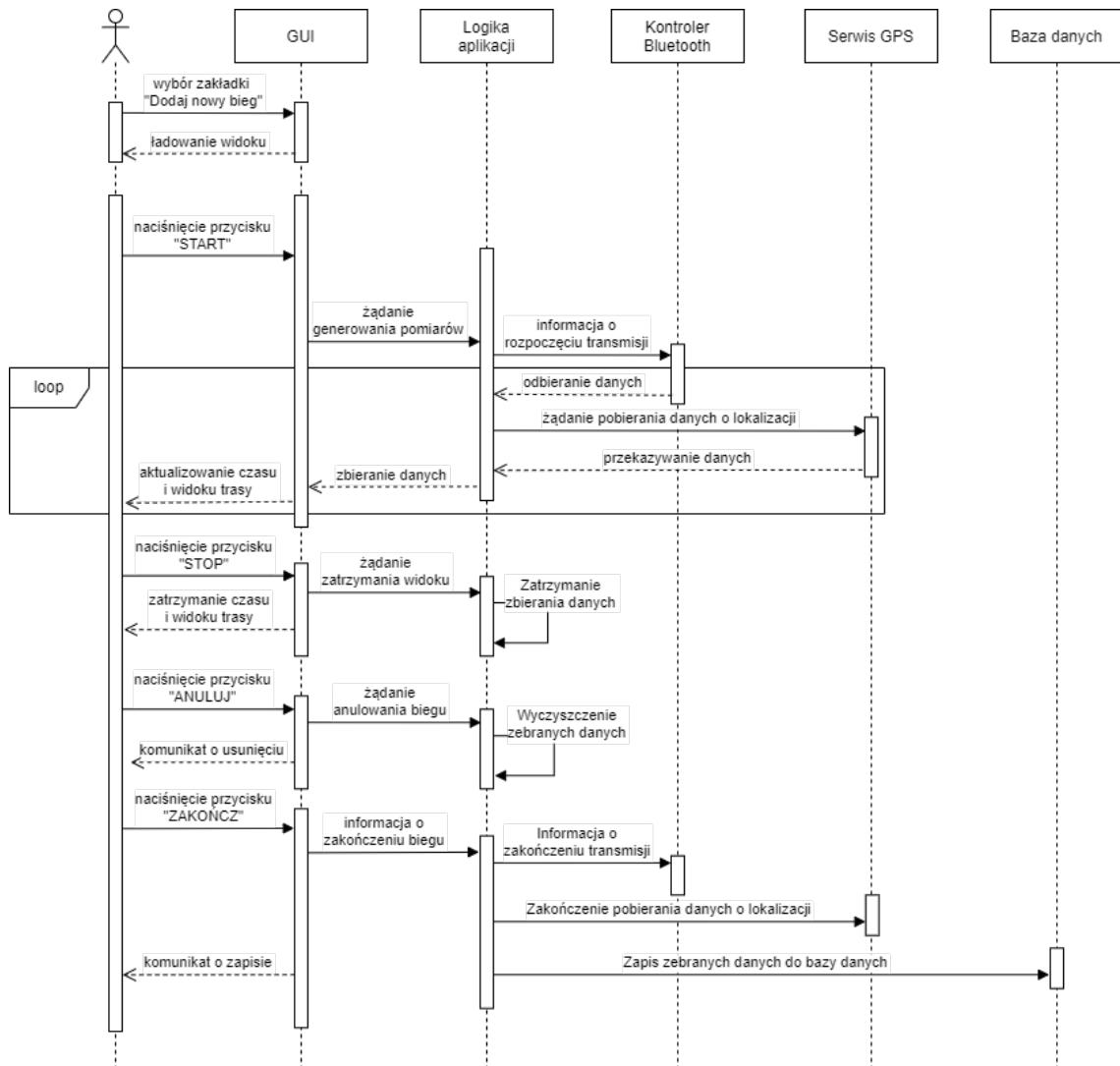
Rys. 2.7. Diagram sekwencji – przetestowanie działania urządzenia

Drugi diagram sekwencji (Rys. 2.8) prezentuje przypadek „Dodawanie nowego biegu” – w tej sytuacji wyróżniamy dwa dodatkowe obiekty. Poza użytkownikiem, GUI, logiką aplikacji i kontrolerem Bluetooth, widoczne są: serwis GPS oraz baza danych.

Sekwencja rozpoczyna się od wybrania odpowiedniej zakładki i załadowania widoku. Po naciśnięciu przycisku „START” podobnie jak w poprzednim przypadku wysyłana jest do logiki aplikacji żądanie rozpoczęcia pomiarów. Logika przesyła do informację o rozpoczęciu transmisji do modułu Bluetooth, oraz żądanie pobrania danych o lokalizacji do serwisu GPS. W odpowiedzi, moduł Bluetooth przesyła odebrane dane, a serwis GPS dane związane z lokalizacją. Logika aplikacji kolekcjonuje dane, a GUI aktualizuje widok trasy i czas trwania biegu. Wszystko to dzieje się w pętli. Naciśnięcie przez użytkownika przycisku „STOP” sprawia, że GUI wysyła żądanie zatrzymania widoku, a aplikacja zatrzymuje zbieranie danych, pozwalając GUI na zatrzymanie czasu i widoku trasy.

Naciśnięcie przycisku „ANULUJ” powoduje wysłanie żądania anulowania biegu przez GUI do logiki aplikacji, która reaguje wyczyszczeniem zebranych danych.

Ostatnią możliwością użytkownika jest naciśnięcie przycisku „ZAKOŃCZ”. W tej sytuacji GUI przesyła do logiki aplikacji informację o zakończeniu biegu. Logika przesyła do modułu Bluetooth informację o zakończeniu transmisji, a do serwisu GPS informację o zakończeniu pobierania danych o lokalizacji. Ostatnim elementem jest przesłanie danych przez logikę aplikacji do bazy danych oraz wyświetlenie przez GUI komunikatu o poprawnym zapisie danych.

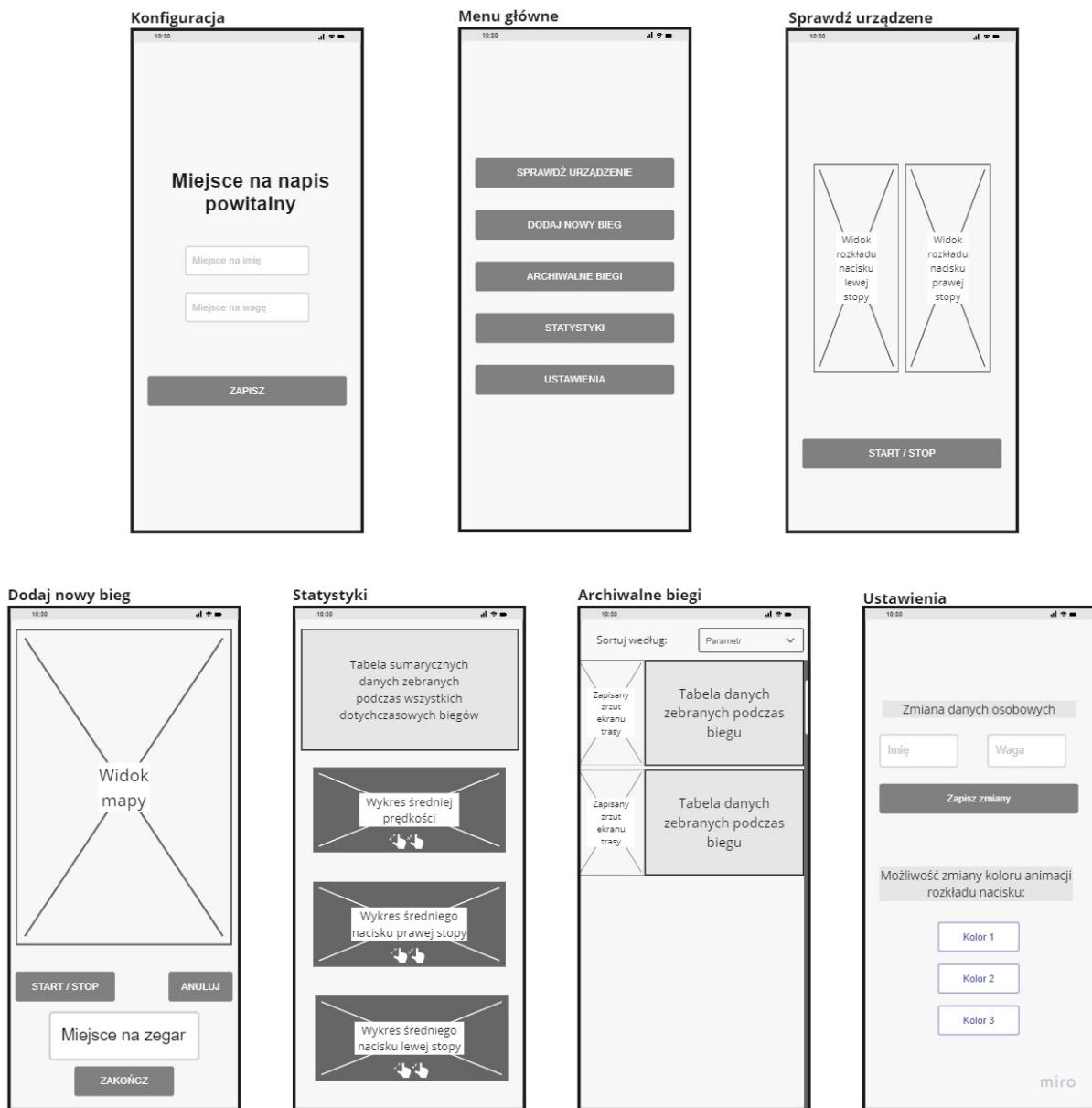


Rys. 2.8. Diagram sekwencji – dodawanie nowego biegu

## 2.6. Mapa ekranów aplikacji

Kolejnym elementem projektowania aplikacji mobilnej jest mapa ekranów (Rys. 2.9). Jest to schemat, który przedstawia graficznie kolejne ekrany aplikacji, co pozwala na zobrazowanie ich ilości, zawartości każdego z nich oraz ich wzajemnych powiązań.

- Okno „Konfiguracja” pojawia się jedynie przy pierwszym użyciu aplikacji, zawiera miejsce na napis powitalny, dwa pola tekstowe w których powinno zostać podane imię oraz obecna waga, a także przycisk „ZAPISZ”.
- „Menu główne” – pojawia się po zakończeniu konfiguracji lub jako pierwsze w przypadku kolejnego użycia aplikacji. Każdy z przycisków prowadzi do wybranej z pięciu pozostałych zakładek.



Rys. 2.9. Mapa ekranów aplikacji mobilnej

- Okno „Sprawdź urządzenie” zawiera miejsca w których powinien pojawić się widok rozkładu nacisku każdej ze stóp oraz przycisk służący do uruchamiania i zatrzymywania generowania kolejnych widoków.
- Zakładka „Dodaj nowy bieg” składa się z widoku mapy, miejsca na zegar oraz trzech przycisków, służących do uruchamiania i zatrzymywania treningu, anulowania go lub zakończenia i zapisania.

- „Archiwalne biegi” to ekran zawierający tabelę z sumarycznymi danymi zebranymi ze wszystkich dotychczasowych biegów. Poza nią, znajdują się tam jeszcze miejsca na trzy wykresy – średniej prędkości, średniego nacisku na prawą i lewą stopę.
- Okno „Statystyki” zawiera widok wszystkich dotychczasowych biegów – składa się on z widoku przebytej trasy i tabeli danych zebranych podczas biegu. Na górze ekranu znajduje się dodatkowo pole – dropdown, w którym można wybrać parametr według którego biegi zostaną posortowane.
- Ostatnią zakładką są "Ustawienia", w których umieszczone są pola tekstowe na nowe imię lub zmienioną wagę, przycisk „ZAPISZ” oraz trzy przyciski umożliwiające wybór kolorystyki grafiki w zakładce „Sprawdź urządzenie”.

## 2.7. Etapy tworzenia systemu

Ostatnim etapem projektowania systemu jest zaplanowanie kolejnych etapów prac. Poniżej znajduje się lista zadań koniecznych do stworzenia systemu monitorowania nacisku stóp.

1. Wybór platformy mikrokontrolera.
2. Wybór konkretnego modułu mikrokontrolera.
3. Konfiguracja projektu w STM32CubeMX.
4. Pomiar kanału przetwornika analogowo-cyfrowego.
5. Pomiar wielu kanałów przetwornika z użyciem DMA.
6. Podłączenie danych wysyłanie danych przez UART z użyciem DMA.
7. Implementacja algorytmu rzutowania danych na skalę procentową.
8. Test przesyłu danych.
9. Uruchomienie działania bluetooth.
10. Implementacja wysyłania danych z mikrokontrolera przez Bluetooth.
11. Implementacja algorytmu prezentacji danych.
12. Konfiguracja projektu w Android Studio.
13. Projekt aplikacji w interfejsie graficznym Android Studio.

14. Implementacja aplikacji mobilnej.
15. Implementacja odbioru danych z Bluetooth przez aplikację mobilną.
16. Integracja algorytmu wizualizującego dane z aplikacją mobilną.
17. Testy modułowe, integracyjne i systemowe.

Po opisaniu wszystkich wymagań jakie musi spełniać system, przyjrzeniu się diagramom, pozwalającym na lepsze zrozumienie działania systemu oraz sprecyzowaniu poszczególnych etapów tworzenia systemu, prace związane z projektowaniem systemu zostały zakończone. Dzięki wymienionym elementom wiadomo, jaki efekt końcowy powinien zostać osiągnięty. W dalszej kolejności należy przyjrzeć się narzędziom i technologiom, za pomocą których możliwe będzie zrealizowanie wszystkich celów i wymagań projektu.

## **3. Technologie i sprzęt wykorzystywany w tworzeniu systemu**

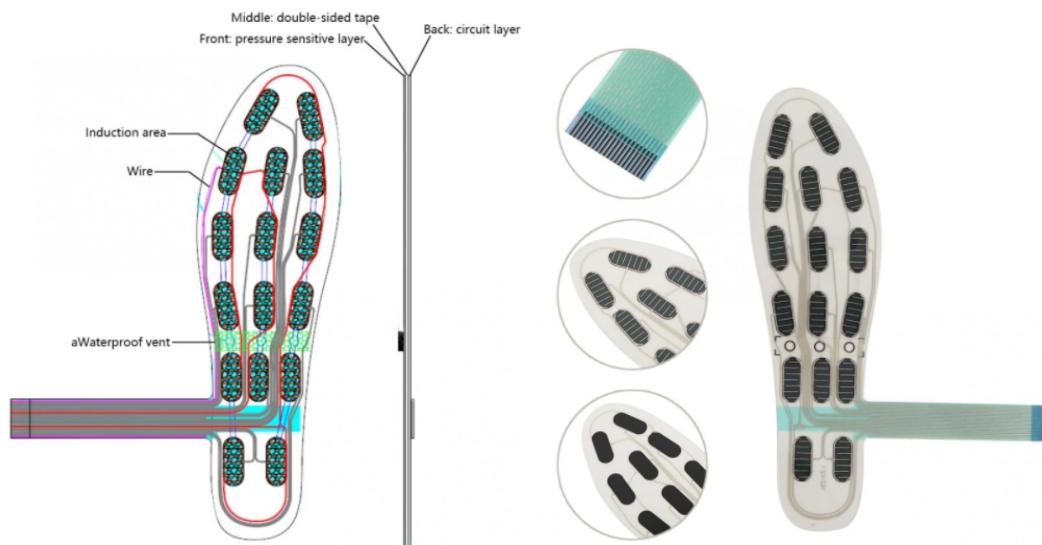
W dzisiejszych czasach istnieje cały szereg języków, kompilatorów, debuggerów i zintegrowanych środowisk programistycznych. Wiele z nich umożliwia implementację systemu monitorującego rozkład nacisku, należy jednak wybrać takie, które pozwolą na możliwie szybkie, bezpieczne i proste tworzenie oprogramowania. Niniejszy rozdział rozpoczęto od opisu warstwy sprzętowej systemu, przyglądając się zarówno samej wkładce, mikrokontrolerowi, jak i dodatkowym komponentom niezbędnym do realizacji projektu. Następnie opisano dodatkowe programy, języki programowania i technologie, które były niezbędne lub znacznie ułatwiały implementację oprogramowania.

### **3.1. Sprzęt**

**Elektroniczna wkładka** złożona jest z 16 czujników nacisku zalaminowanych w przezroczystej folii. Zgodnie z opisem producenta [11] czujniki wyzwalane są siłą 500 g i są w stanie mierzyć nacisk do 10 kg. Biorąc pod uwagę ich liczbę, zakres pomiaru jest wystarczający. Całe wkładki mają niewielką grubość – ok. 0.4 mm, są całkowicie pyłoszczelne i odporne na krótkie (30-sekundowe) zanurzenie w wodzie do głębokości 1 m. Rys. 3.1 prezentuje schemat oraz rzeczywisty widok pojedynczej wkładki.

**Mikrokontroler STM32L010RB.**

- architektura ARM – 32 bitowy układ daje dużo więcej możliwości niż 8 bitowe mikrokontrolery AVR. Ponadto ARM umożliwia użycie np. wielu kanałów ADC;
- produkt firmy ST – firma dostarcza mikrokontroler oraz odpowiednie narzędzia do pisania kodu, takie jak STM32CubeMX oraz biblioteki STM32HAL;
- rdzeń CorexM0+ – rodzina Corex-M jest przeznaczona dla mikrokontrolerów. "M0+" to oznaczenie ulepszonej wersji podstawowego rdzenia M0, o zmniejszonym poborze mocy;



**Rys. 3.1.** Schematy wkładki elektronicznej

- układ Ultra Low Power – zapewnia możliwie najniższe zużycie baterii i bardzo niski pobór mocy;
- DMA – Direct Memory Access, czyli bezpośredni dostęp do pamięci RAM. Moduł ten pozawala na odciążenie procesora, wyręczając go w przesyłaniu danych między pamięcią, a układami peryferyjnymi lub między różnymi obszarami pamięci.

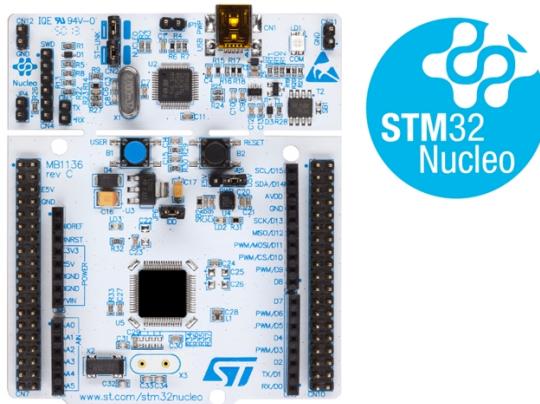
W Tabeli 3.1, dla porównania, zestawiono ze sobą kilka wybranych parametrów procesora STM32L010 [12] oraz Atmega128 [13] (wybrano te dwa ze względu na pewne podobieństwa, takie jak np. rozmiar pamięci Flash).

	<b>Atmega 128</b>	<b>STM32L010</b>
<b>CENA</b>	ok. 69.00zł	ok. 14.00zł
<b>ARCHITEKTURA</b>	8 Bit	32Bit
<b>ZEGAR</b>	16MHz	32MHz
<b>PAMIĘĆ</b>	128KB FLASH	128KB FLASH
<b>RAM</b>	4K x8	20K x8
<b>DMA</b>	brak	jest

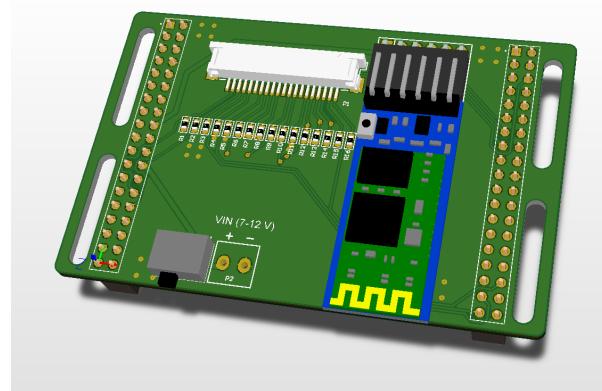
**Tabela 3.1.** Porównanie parametrów Atmega128 i STM32L010RB

Na bazie powyższego porównania można zauważyć, że pomimo niższej ceny wiele parametrów wspomnianego STM32 wypada dużo korzystniej.

**Płytki rozwojowe STMicroelectronics** – mikrokontroler STM32L010RB znajduje się na płytce rozwojowej z serii NUCLEO – STMicroelectronics NUCLEO-L010RB, której wygląd prezentuje Rys. 3.2. Poza wspomnianym mikrokontrolerem zestaw posiada bardzo dużą zaletę – wbudowany programator-debuger ST-Link/v2-1 [12]. Umożliwia on szybkie i wygodne programowanie oraz testowanie tworzonych programów. Całość może być zasilana na różne sposoby – między innymi z baterii poprzez pin VIN oraz poprzez złącze USB. Wykorzystujemy 16 kanałów ADC zaprojektowanych na płytce, aby pobierać pomiary z 16 czujników nacisku. Dodatkowo używane są wyprowadzenia pinów RX oraz TX, aby umożliwić połączenie się z modułem Bluetooth. Możliwe jest również odłamanie części z debugerem w celu zmniejszenia rozmiaru płytki, co w przypadku tego projektu jest istotne.



Rys. 3.2. Wygląd płytki  
Nucleo-L010RB

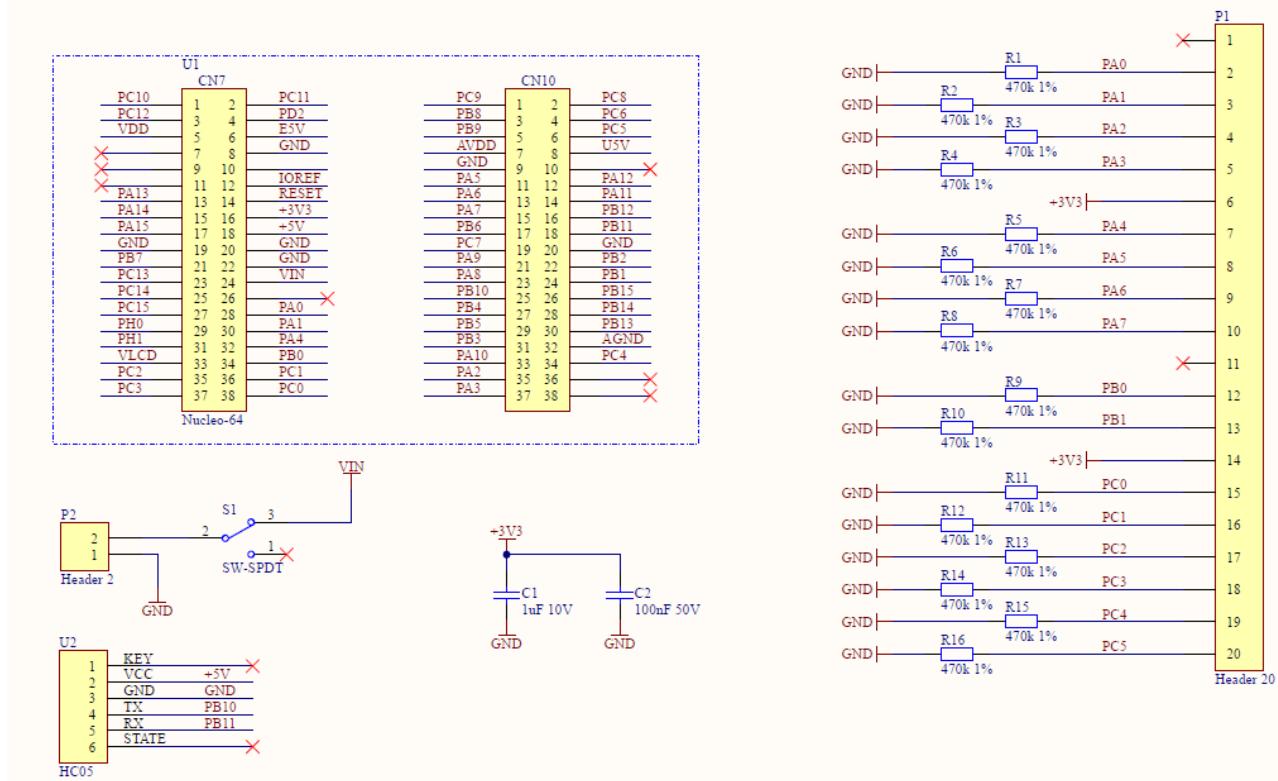


Rys. 3.3. Projekt płytki  
dedykowanej

### Płytki dedykowana PCB

Sama płytka rozwojowa to nie wszystko – potrzebne były dodatkowe złącza, przełączniki, przewody oraz rezystory. Aby uniknąć znacznego powiększania objętości warstwy sprzętowej, zdecydowano się użyć dodatkowej płytki PCB, zaprojektowanej specjalnie na potrzeby tego systemu. Rys. 3.3 prezentuje wygląd projektu wspomnianej płytki, a Rys. 3.4 schemat połączeń, które się na niej znajdują.

Dodatkowa płytka zapewnia wyprowadzenie kanałów ADC na złącze typu Molex, do którego podłączona jest wkładka. Dzięki niej zyskano również bezpośrednie połączenie z modułem Bluetooth HC-05. Zapewniono też wygodne miejsce na włącznik i miejsce podłączenia źródła energii. Całość została zamknięta w dość małym rozmiarze – ok. 70x50 mm, czyli jest nieco mniejsza niż pozostawiona po odłamaniu część wyżej wymienionej płytki rozwojowej Nucleo.

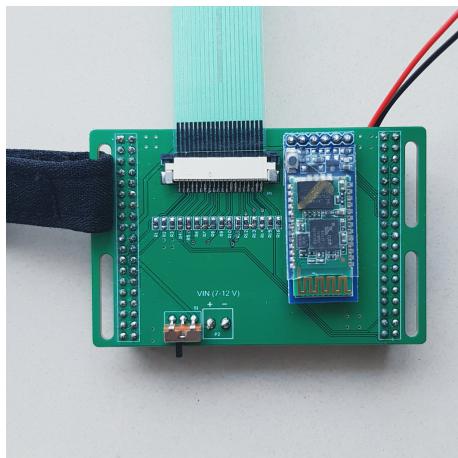


Rys. 3.4. Schemat płytka dedykowanej

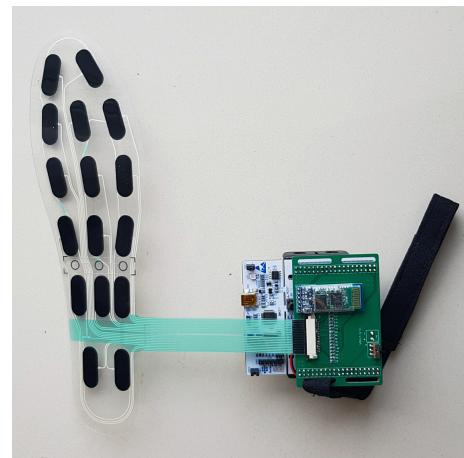
Na zaprojektowanej płytce znajdują się następujące komponenty:

- moduł Bluetooth HC-05,
- złącze typu Molex, 20 pinowe,
- rezystory SMD 0603 470k $\Omega$ ,
- przełącznik suwakowy,
- listwa goldpin żeńska, 8 styków, kątowa, jednorzędowa,
- dwie listwy goldpin żeńskie – 20 styków, prosta, dwurzędowa.

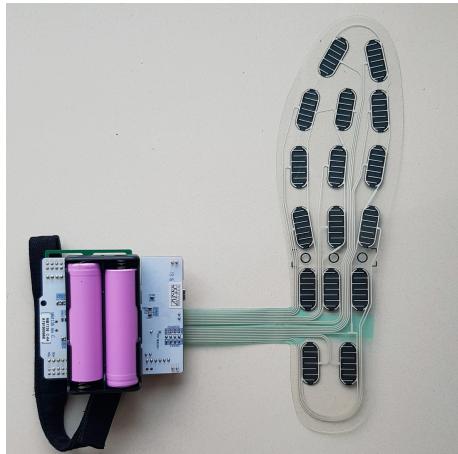
Po wlutowaniu wszystkich wspomnianych elementów na dedykowaną płytkę, należało jeszcze dodać odpowiednie źródło prądu. Następnie skonfigurowano płytę Nucleo-L010RB – niektóre mostki należało rozewrzeć lub zewrzeć, aby umożliwić realizację projektu. Po umieszczeniu nakładki w odpowiednim miejscu płytki Nucleo oraz wpięciu wkładki elektronicznej, powstała gotowa warstwa sprzętowa. Po zakończeniu implementacji możliwe będzie odłamanie części płytki Nucleo zawierającej debugger, dzięki czemu dodatkowo zostanie zmniejszony rozmiar sprzętu. Efekty prac przedstawione zostały na Rys. 3.5, Rys. 3.6, Rys. 3.7, Rys. 3.8.



Rys. 3.5. Widok zaprojektowanej płytki po wlutowaniu komponentów



Rys. 3.6. Widok gotowej warstwy sprzętowej (rzut z góry)



Rys. 3.7. Widok gotowej warstwy sprzętowej (rzut z dołu)

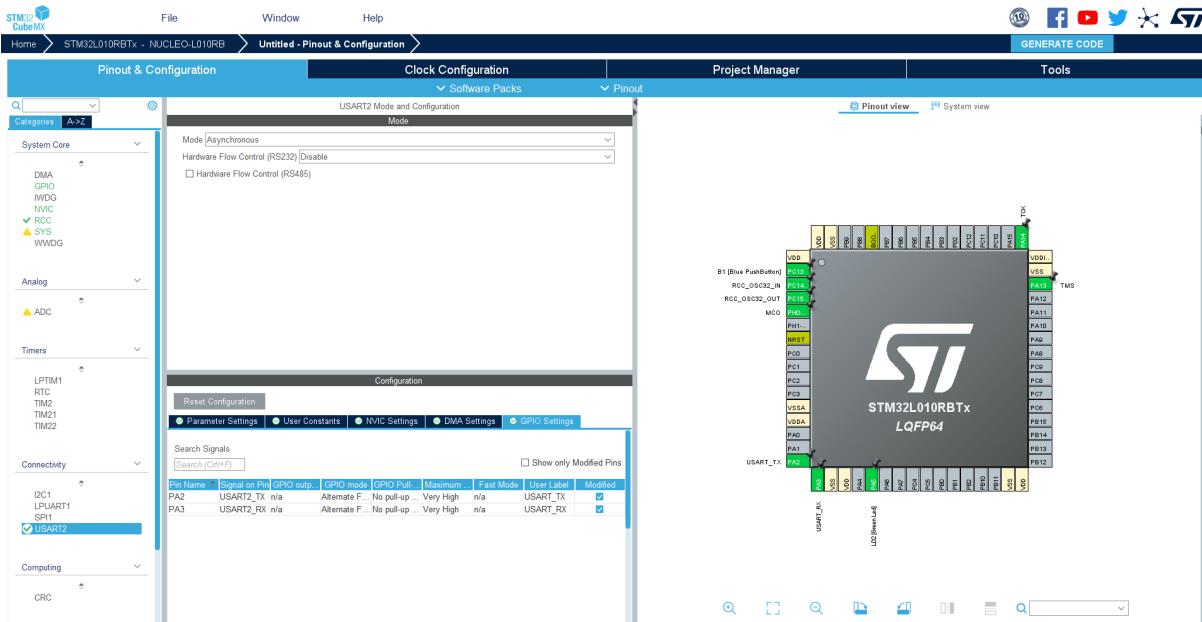


Rys. 3.8. Widok sprzętu po założeniu przez użytkownika

## 3.2. Narzędzie STM32CubeMX oraz biblioteka HAL

**CubeMX** to program, który umożliwia konfigurację peryferiów mikrokontrolera STM32 za pomocą interfejsu graficznego. Narzędzie po uruchomieniu proponuje wybór mikrokontrolera, następnie generuje plik projektu pytając, czy powinno zainicjalizować wszystkie porty w sposób domyślny. Po zakończeniu prac nad konfiguracją, istnieje możliwość wygenerowania pełnego kodu konfiguracyjnego w języku C. Jak widać na zrzucie ekranu zaprezentowanym

na Rys. 3.9, poza ustawieniami peryferiów, istnieje możliwość konfiguracji zegara. Po kliknięciu na poszczególne piny mikrokontrolera, rozwija się lista opcji, co jest możliwe do podłączania w wybranym miejscu. Dzięki temu możliwe jest łatwe przemapowanie pinów.



Rys. 3.9. Zrzut ekranu programu CubeMX, ustawienia standardowe

**Biblioteki HAL** pozwalają na korzystanie z peryferiów w prosty sposób. Dzięki nim kod staje się bardziej czytelny, dodatkowo znacznie przyspieszają programowanie – po szybkiej konfiguracji można przejść do kluczowych zagadnień projektu.

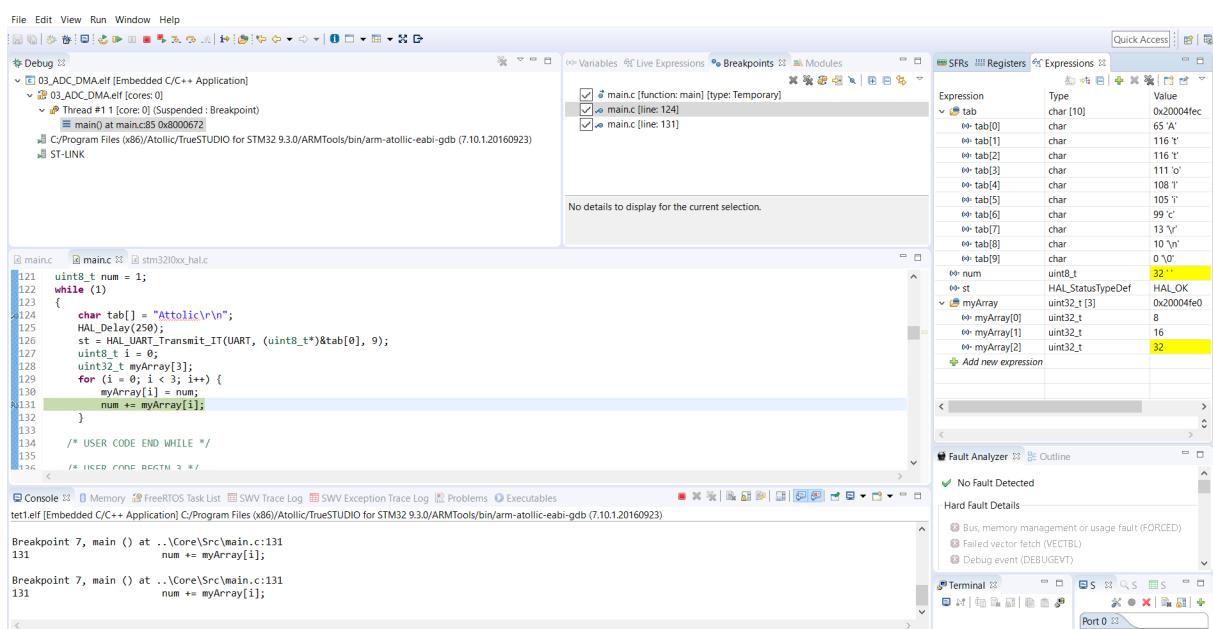
### 3.3. Język programowania C

Językami, których najczęściej używa się do programowania mikrokontrolerów są Asembler oraz język programowania C. Ze względu na dużo łatwiejsze tworzenie i rozwijanie programu oraz dostępność wielu wspomagających narzędzi, w projekcie zdecydowano się użyć języka C.

Ranking TIOBE [14] ukazuje, że C jest jedynym z najpopularniejszych języków programowania. Jego najważniejszą zaletą, obok szerokich zastosowań i możliwością pracy z różnorodnym sprzętem, jest szybkość wykonywania operacji. Warto wspomnieć też o stosunkowo łatwym dostępie do rejestrów sprzętowych – dzięki operacjom na wskaźnikach możliwy jest ich odczyt, modyfikacja i zapis.

## 3.4. Środowisko programowania mikrokontrolerów

Spośród wielu dobrych narzędzi programistycznych, wybrano pakiet **TrueStudio** firmy **Attolic**. IDE (Integrated Development Environment – zintegrowane środowisko programistyczne) bazuje na Eclipse, zawiera debugger i kompilator oraz jest całkowicie darmowe. W 2017 roku [15] firma ST przejęła firmę Attolic, tym samym zyskując narzędzie dedykowane dla swoich produktów. Od tamtego czasu firma rozwija pakiet TrueStudio dodając dodatkowe udoskonalenia dla deweloperów. Jak widać na zrzucie ekranu przedstawionym na Rys. 3.10, debugger daje bardzo wiele możliwości.



Rys. 3.10. Zrzut ekranu z programu Attolic

## 3.5. Matlab

**Matlab** to popularny program komputerowy, który służy do wykonywania obliczeń i tworzenia symulacji komputerowych, często używany przy projektach inżynierskich i naukowych.

**Curve Fitting Tool** to narzędzie Matlaba, które zawiera funkcje umożliwiające dopasowanie krzywych i powierzchni do podanych danych. Pozwala na analizę danych i porównywanie modeli w celu znalezienia najlepszego dopasowania. Istnieje możliwość analizy regresji liniowej, nieliniowej i własnych równań, a także wygładzania i interpolacji funkcji. Używana biblioteka zapewnia optymalizację parametrów w celu poprawy jakości dopasowania.

## 3.6. Android Studio

**Android** to system operacyjny firmy Google, który od kilku lat jest numerem jeden na świecie pod kątem popularności platform mobilnych. Nic więc dziwnego, że tak powszechny system oferuje również odpowiedni zestaw narzędzi do tworzenia aplikacji – **Android Studio**.

Jest to oficjalne IDE do tworzenia aplikacji na platformę Android. Opiera się na IntelliJ IDEA i jest wyposażone w szeroki zasób narzędzi do analizy i edycji kodu. Po uruchomieniu środowisko zaproponuje wybór aktywności, czyli elementów aplikacji – programista spośród wielu opcji może wybrać podstawową lub pustą aktywność, ale także powiązaną z Google Maps czy logowaniem. W kolejnym kroku Android Studio pozwala (między innymi za pomocą interfejsu graficznego) zaprojektować aplikację metodą "przeciągnij i upuść" (ang. *drag and drop*) jednocześnie generując kod XML. Po napisaniu, zakończeniu debugowania i zbudowaniu projektu istnieje możliwość uruchomienia go na dedykowanym emulatorze lub własnym urządzeniu mobilnym.

## 3.7. Język programowania aplikacji mobilnych

Oficjalnym językiem programowania aplikacji na platformę Android jest **Kotlin**. Jest to stosunkowo nowy, przężnie rozwijający się język, działający na maszynie wirtualnej Javy. Kotlin z założenia upraszcza składnię Javy – tym samym jest znacznie bardziej przejrzysty i uporządkowany. Możliwa jest migracja projektów z Javy na Kotlin (w Android Studio nawet automatyczna), a sam język współpracuje z takimi narzędziami jak Apache Maven, czy Gradle.

## 3.8. Pozostałe użytte technologie i języki

### 3.8.1. Dagger i Hilt

**Dagger** [16] to jeden z najczęściej używanych framework'ów wstrzykiwania zależności (DI, ang. *Dependency Injection*) dla platformy Android. DI polega na dostarczaniu gotowych obiektów i ich metod do obiektów, które z nich korzystają, zamiast samodzielnego ich konstruowania. Używany jest w celu uproszczenia budowania i udostępniania zależności w całej aplikacji. W praktyce, traktowany jest jako narzędzie do generowania kodu – dostaje informacje jak tworzyć instancje kolejnych klas, jak je budować i łączyć. Wszystkie te informacje są przekazywane do Daggera poprzez zawarte w kodzie adnotacje, takie jak np. `@Singleton`, `@Component.Builder`, czy `@Inject`.

Korzystanie z Dagger nie jest jednak łatwym zadaniem. Aby uprościć to zadanie w projekcie wykorzystano bibliotekę **Hilt** [16]. W rzeczywistości w niniejszym projekcie używana jest biblioteka Hilt, która korzysta wewnętrznie z Dagger, włączając wstrzykiwanie zależności do aplikacji Androida.

### 3.8.2. XML jako język graficznego interfejsu użytkownika

Do zaimplementowania aplikacji mobilnej nie wystarczy jedynie sam język Kotlin. Do opisu układów interfejsu użytkownika znajdujących się na kolejnych ekranach używany jest język **XML**. Jest to język znaczników, podobny do znanego z tworzenia stron internetowych języka HTML. Każdy widok ekranu jest oddzielnym plikiem języka XML i musi zawierać jeden element główny typu View (może być to np. układ liniowy (ang. *Linear Layout*, względny *Relative Layout* lub układ ramek (*Frame Layout*)). Na obiekcie głównym znajdują się wszystkie inne elementy, takie jak przyciski, pola tekstowe, czy grafiki. Każdemu elementowi można przypisać różne właściwości, między innymi rozmiar, kolor czcionki, tła, grubość marginesów i wiele innych. Dodatkową zaletą jest możliwość tworzenia układów w języku XML i zmiana ich wyglądu w czasie działania aplikacji poprzez kod w języku Kotlin.

### 3.8.3. SQL oraz DAO

Ostatnim językiem programowania, bez którego projekt nie ma szans na powodzenie jest język **SQL**. Jest to język zapytań, używany do tworzenia i modyfikowania tabel oraz pobierania i wysyłania danych z baz danych. Dane w Androidzie mogą być przechowywane na różne sposoby – za pomocą tzw. SharedPreferences (przechowują one głównie niewielkie ilości danych, służą do przechowywania ustawień aplikacji), przy użyciu plików oraz w bazach danych SQLite. Ostatni sposób umożliwia przechowywanie nawet dużej ilości uporządkowanych danych i wysoką wydajność dostępu do nich – dla tego potrzebny jest język SQL. Połączenie języka SQL z Kotlinem zapewnia **DAO** (ang. *Data Access Object*). Za pomocą adnotacji takich jak `@Query()`, `@Insert`, czy `@Update` polecenia języka SQL są mapowane na funkcje języka Kotlin. Przykładowe przemapowanie widoczne jest w Listingu 3.1.

Listing 3.1 zawiera przykładowy kod, który przemapowuje zapytanie języka SQL (linia 1) na metodę `getAllRunsSortedByDate()` – dzięki temu w całej aplikacji mobilnej możliwe jest używanie jedynie wspomnianej wyżej metody w języku Kotlin, bez konieczności dodatkowego pisania zapytań w języku SQL.

**Listing 3.1.** Przykład użycia DAO

```
@Query("SELECT * FROM running_table ORDER BY timestamp DESC")
fun getAllRunsSortedByDate(): LiveData<List<Run>>
```

W niniejszym rozdziale opisano wszystkie najważniejsze technologie, które zostały użyte do realizacji projektu. Wspomniano zarówno o używanych językach programowania oraz środowiskach programistycznych, jak i dodatkowych narzędziach oraz stosowanych bibliotekach. Następnym etapem prac nad niniejszym projektem jest implementacja oprogramowania mikrokontrolera, wymiany danych między mikrokontrolerem a smartfonem oraz wytworzenie oprogramowania aplikacji mobilnej.

## 4. Implementacja

Cykl życia projektu składa się z co najmniej kilku faz [17]: planowania, analizy, projektowania i implementacji. W poprzednich rozdziałach przyjrzano się istniejącym rozwiązaniom, przeanalizowano problem, zaprojektowano system i opisano kolejne funkcjonalności systemu. Zaplanowano również kolejne etapy tworzenia systemu – zadano więc o fazę planowania, analizy i projektowania. W tym rozdziale skupiono się na ostatniej wspomnianej fazie – implementacji systemu.



Rys. 4.1. Schemat działania systemu

Na działanie niniejszego systemu (Rys. 4.1) składają się następujące elementy:

- implementacja oprogramowania mikrokontrolera, aby pobierał dane z czujników i wysyłał je przez Bluetooth,
- implementacja algorytmu przeskalowania danych z zakresu 0-65535 do 0-100,
- implementacja aplikacji mobilnej na system Android,
- implementacja algorytmu odpowiadającego za wizualizację zebranych danych.

W kolejnych sekcjach niniejszego rozdziału opisano szczegółowo każdy z ww. elementów.

### 4.1. Oprogramowanie mikrokontrolera

Programowanie mikrokontrolera rozpoczęto od stworzenia odpowiedniej konfiguracji w programie CubeMX. Najważniejszymi jej elementami są:

- inicjalizacja wszystkich 16-stu kanałów ADC oraz ustawienia ich parametrów, takich jak przesunięcie, oversampling, sampling,

- uruchomienie DMA,
- ustawienie zegara RCC,
- ustawienie źródła zegara TIM2 jako Internal Clock,
- uruchomienie Low Power UART,
- odpowiednia konfiguracja zegara.

Po zakończeniu konfiguracji, wygenerowano kod projektu.

**Listing 4.1.** Fragment kodu zawierający startową konfigurację

---

```
HAL_GPIO_WritePin(B1_GPIO_Port, B1_Pin, SET);  
HAL_Delay(1000);  
HAL_ADCEx_Calibration_Start(&hadc, ADC_SINGLE_ENDED);  
HAL_ADC_Start_DMA(&hadc, (uint32_t*)adcValue, 16);  
HAL_ADC_Start_IT(&hadc);  
HAL_UART_Transmit(&hlpuart1, (uint8_t*)"Start_transmision",  
      strlen("Start_transmision"), 500);  
HAL_Delay(1000);  
HAL_TIM_Base_Start_IT(&htim2);
```

---

Po zainicjalizowaniu peryferiów, najważniejszą częścią była kalibracja kanałów ADC oraz uruchomienie DMA. Zainicjalizowano również transmisję Bluetooth poprzez próbę nadania hasła `start_transmision`. Jak widać na załączonym fragmencie kodu (Listing 4.1), stale wykorzystywana jest tu opisana w rozdziale 3.2 biblioteka HAL.

Wspomniany kod zapewnił już cykliczny odczyt danych z czujników nacisku. Transmisja danych wykonuje się w głównej pętli programu (kod znajduje się w Listingu 4.2). Warto jednak wziąć pod uwagę, że wybrany moduł Bluetooth nie jest modułem typu Low Energy. Powinno się więc zadbać, by transmisja nie zabierała zbyt dużo zasobów baterii. Po rozważaniach w jaki sposób można zapewnić oszczędzanie energii, wysunięto następujące wnioski:

1. Transmisja powinna odbywać się dopiero po otrzymaniu z aplikacji informacji o rozpoczęciu transmisji;
2. Dane liczbowe powinny być zapisane w możliwie niewielkim buforze;
3. Dane nie są transmitowane, jeżeli ich maksymalna odczytana wartość nacisku jest mniejsza niż 10% zakresu;
4. Transmisja odbywa się co możliwie najdłuższy okres czasu zapewniający płynność odczytów – czyli co 150 ms.

**Listing 4.2.** Fragment kodu zawierający główną pętlę programu

---

```
while (1) {
    HAL_UART_Receive_IT(&hlpuart1, (uint8_t*)&buffer, 1);
    if(buffer == '1') flagBuffer = 1;
    if(buffer == '2') flagBuffer = 0;

    uint8_t i~= 0, max = 0;
    char *adcAsChar;
    if(flagBuffer==1){
        char finalString[18]={0};
        while(i<=15){
            // ZAMIANA DANYCH Z~CZUJNIKÓW NA SKALE 0-100%
            ...
        }
        strncat(finalString, "\r\n", 2);
        if (max>5){
            HAL_UART_Transmit(&hlpuart1, (uint8_t*)&finalString[0], 18, 100);
        }
        HAL_Delay(150);
    }else {
        HAL_Delay(500);
    }
}
```

---

Jak widać w Listingu 4.2, na początku odbierana jest dana liczbową z aplikacji mobilnej – jeżeli jest to jeden, zmienna `flagBuffer` również przyjmuje wartość 1. Jeżeli jest to 2, transmisja powinna zostać zakończona – `flagBuffer` przyjmuje wartość 0, a transmisja jest przerywana. W przypadku braku odebranej jakiejkolwiek transmisji, wprowadzono opóźnienie.

Jeżeli wartość `flagBuffer` jest równa 1, to dane z czujników są zamieniane na odpowiednią skalę i zamieniane do postaci tablicy 18-elementowej. Na 18 elementów składają się odczyty z 16 czujników – 1 czujnik = 1 bit oraz 2 bity końca linii.

Jeżeli wartość maksymalna z odczytywanych pomiarów jest większa niż 10%, to dane są transmitowane za pomocą Bluetooth (w przeciwnym razie transmisja byłaby jedynie stratą energii). Na koniec wątek jest usypanny na 150 ms, co pozwala na pomiar 400 razy w ciągu minuty. Maksymalna kadencja biegowa, czyli ilość kroków na minutę to ok 200-220 [5]. Zgodnie z twierdzeniem Nyquista–Shannona (o próbkowaniu) [18], próbki powinny być zbierane co najmniej dwa razy częściej niż okres pomiaru. Jak wspomniano wyżej, pomiary wykonywane są 400 razy w ciągu minuty, tym samym minimum dwa razy na każdy krok.

Dodatkowo w celu zaoszczędzenia energii w czasie przerwy w pracy funkcji, wprowadzono dodatkowe polecenie w funkcji `HAL_Delay()` – `__WFI` znajdującej się w bibliotece HAL. W tym celu stworzono własną funkcję opóźniającą – jej kod zaprezentowano w Listingu 4.3.

**Listing 4.3.** Własna funkcja opóźniająca

---

```
void HAL_Delay(uint32_t Delay)
{
    uint32_t tickstart = HAL_GetTick();
    uint32_t wait = Delay;

    if (wait < HAL_MAX_DELAY)
    {
        wait += (uint32_t)uwTickFreq;
    }

    while ((HAL_GetTick() - tickstart) < wait)
    {
        __WFI();
    }
}
```

---

Zastosowanie instrukcji `__WFI` wstrzymuje wykonywanie programu aż do pierwszego przerwania. Zegar systemowy sprawia, że przerwanie pojawia się co 1 ms, więc program będzie wielokrotnie usypany, co pozwoli na zaoszczędzenie części energii.

## 4.2. Interpretacja danych

Dla przeciętnego odbiorcy, najłatwiejszą do odbioru skalą jest skala procentowa. Niestety w rzeczywistości dane, które są odczytywane przez czujniki nacisku nie rosną liniowo, a więc trudno jest je zinterpretować. W poniższym podrozdziale opisano dokładnie, jakie kroki zastosowano w celu przekształcenia zbieranych wyników w taki sposób, aby były one wygodne do odczytu.

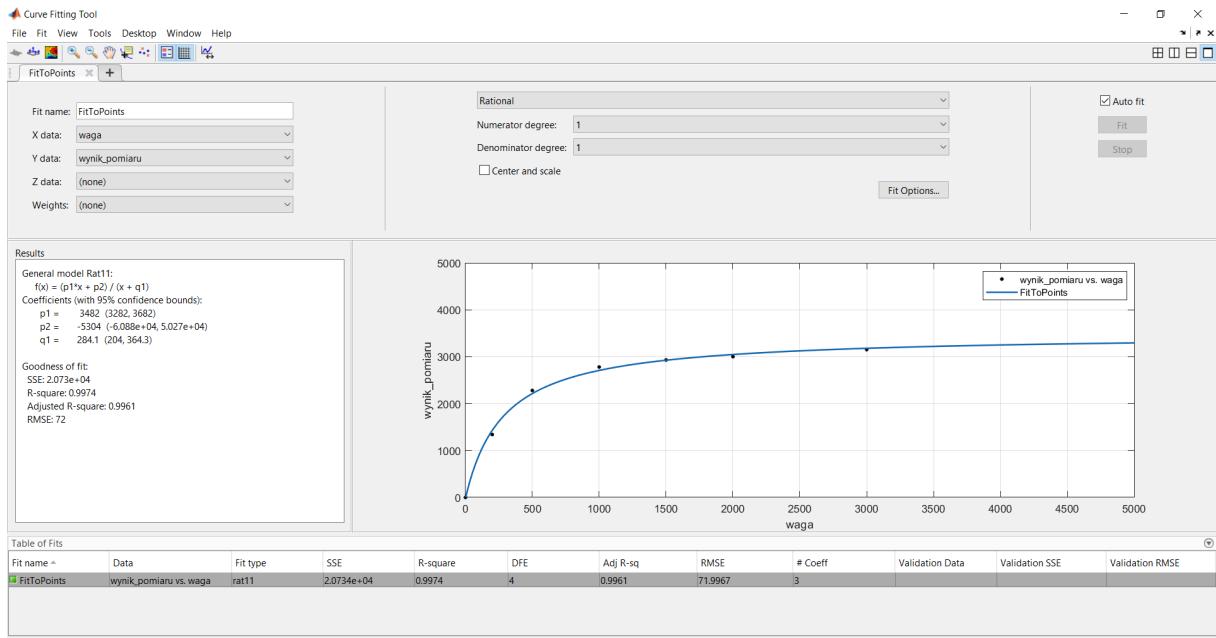
Rozpoczęto od zebrania danych. Pod wkładką umieszczono miękki materiał, aby symułować warunki rzeczywiste. Ścięty stożek, ułożony podstawą o mniejszej powierzchni do dołu, umieszczono na środku czujnika. Następnie na stożku umieszczały kolejne odważniki i zapisywano odczytywane pomiary. Całość powtórzono pięciokrotnie, za każdym razem wybierając inny, losowy czujnik. Efektem uśrednienia i zaokrąglenia zebranych pomiarów są dane zaprezentowane w Tabeli 4.1.

Numer pomiaru	Waga odważnika [g]	Odczytana wartość
1	200	1340
2	500	2280
3	1000	2780
4	1500	2930
5	2000	3000
6	3000	3150

**Tabela 4.1.** Tabela pomiarów po uśrednieniu wartości

Dane z Tabeli 4.1 dodatkowo podzielono przez 16, uzyskując skalę 0-4095 zamiast skali 0-65535. Zabieg ten wykonano w celu lepszego dopasowania funkcji o możliwie najniższym stopniu. Zebrane dane, to punkty do których należało dopasować funkcję nacisku. Najlepszym narzędziem, które można było w tym celu zastosować był Matlab. Po wpisaniu danych w konsołę programu, uruchomiono narzędzie Curve Fitting Tool (Rys. 4.2). Spośród wielu możliwych rozwiązań znalezionych we wspomnianym programie, wybrano najlepiej dopasowaną funkcję, o możliwie najmniejszym złożeniu:

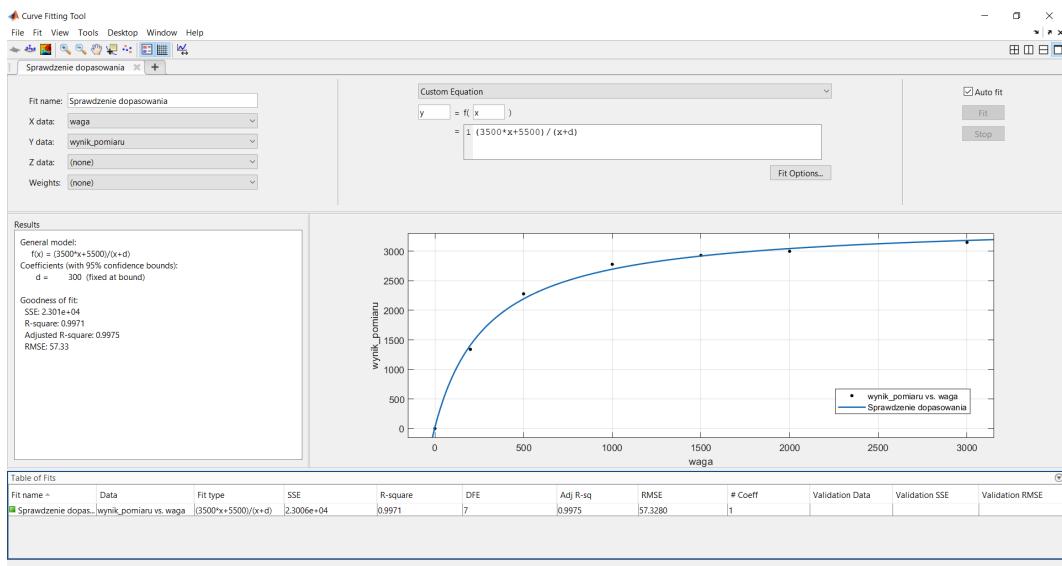
$$y = \frac{3482x - 5304}{x + 284.1} \quad (4.1)$$

**Rys. 4.2.** Curve Fitting Tool: Znalezienie funkcji

W związku z tym, że w rzeczywistości nie jest potrzebna nam aż taka dokładność, zdecydowano o zaokrągleniu parametrów funkcji. W efekcie powstała następująca funkcja:

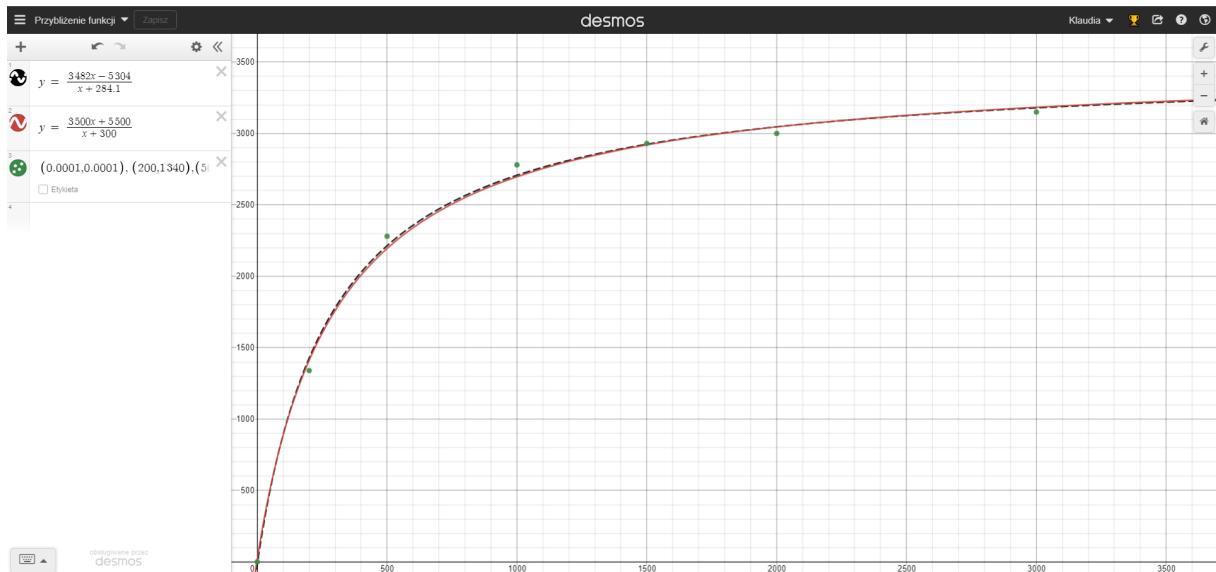
$$y = \frac{3500x - 5500}{x + 300} \quad (4.2)$$

Następnie sprawdzono za pomocą narzędzia Curve Fitting Tool, czy odchylenie funkcji od punktów nie zwiększyło się znacząco – efekt użycia wspomnianego narzędzia prezentuje zrzut ekranu zamieszczony na Rys. 4.3.



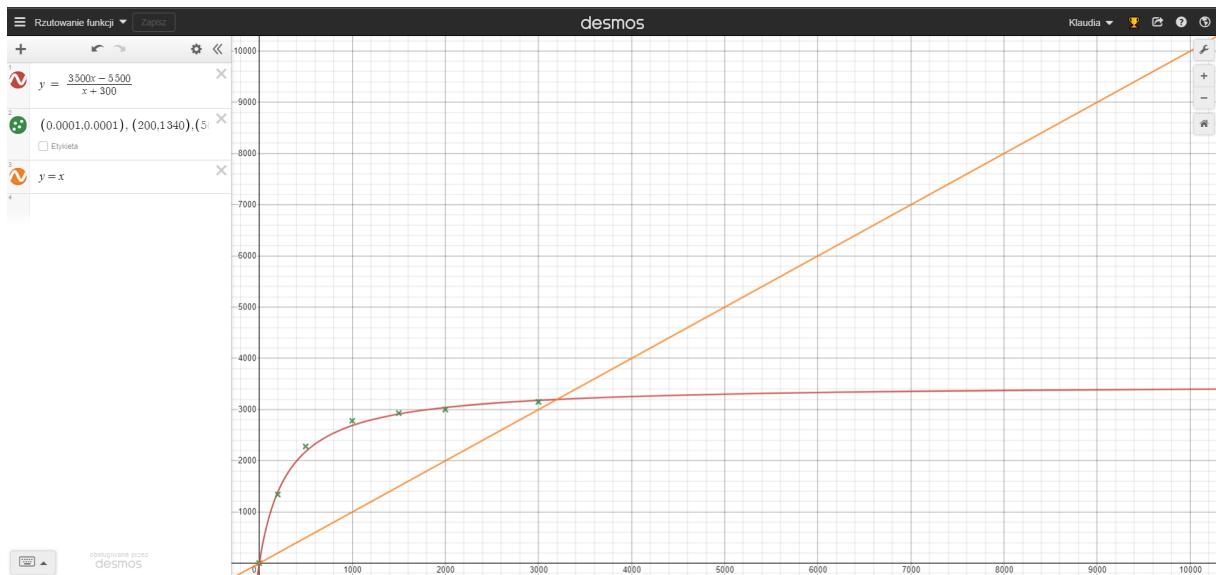
Rys. 4.3. Curve Fitting Tool: Dopasowanie funkcji

Na zrzucie ekranu przedstawionym na Rys. 4.4 porównano również, jak niewiele różnią się obie funkcje w rzeczywistości. Użyto w tym celu kalkulatora graficznego – Desmos [19].



Rys. 4.4. Desmos: Porównanie funkcji

Po znalezieniu odpowiedniej funkcji pozostała konieczność przemapowania wartości tak, aby wskazywały one na skalę procentową. W obecnej sytuacji, w przypadku gdy nacisk na czujnik jest około 4 kg, otrzymujemy podobne wartości co w przypadku, gdy nacisk jest dwukrotnie wyższy. Natomiast różnica między odczytem 10 g oraz 500 g to prawie połowa rozpiętości wykresu (co zaprezentowano na Rys. 4.5).



Rys. 4.5. Desmos: rzutowanie funkcji

W celu rozwiązania tego problemu, zastosowano proste przekształcenia matematyczne:

$$\begin{aligned}
 y &= \frac{3500x - 5500}{x + 300} \\
 y(x + 300) &= 3500x - 5500 \\
 xy + 300y &= 3500x - 5500 \\
 xy - 3500x &= -300y - 5500 \\
 x(y - 3500) &= -300y - 5500 \\
 x &= \frac{-300y - 5500}{y - 3500}
 \end{aligned} \tag{4.3}$$

Finalnie całe to działanie zawarto w dwóch liniach kodu języka C (Listing 4.4). Początkowo odczytano wartość z kanału ADC i podzielono ją przez 16 przypisując do zmiennej  $y$ . Następnie obliczono wynik wskazanej wyżej funkcji i przypisano do tablicy  $ADC\_Values$ .

**Listing 4.4.** Obliczenia w języku C

---

```

uint16_t y=(uint16_t)(adc_value[i]/16);
ADC_Values[i]=(-300*y-5500)/(y-3500)

```

---

## 4.3. Aplikacja mobilna

Zgodnie z diagramami zawartymi w rozdziale 2.5.1, gdy dane zostaną przekształcone w odpowiedni sposób, mikrokontroler wysyła je przez moduł Bluetooth do aplikacji. Zanim jednak dane zostaną wyświetlane w postaci rozkładu nacisku lub statystyk użytkownikowi przechodzą przez szereg klas, obiektów i metod. Niniejsza sekcja w całości poświęcona jest opisowi implementacji części mobilnej, a mimo tego nie jest możliwe dokładne opisanie całej logiki aplikacji. Skupiono się na najważniejszych oraz najbardziej kluczowych elementach tej części systemu.

Pracę nad aplikacją rozpoczęto od zaimplementowania zaprojektowanego w rozdziale 2.6 interfejsu graficznego użytkownika w języku XML. Następnie przygotowano odpowiednie ikony, grafiki i animacje oraz dodano do pliku build.gradle wszystkie potrzebne dodatkowe biblioteki i źródła.

Prace nad kolejnymi częściami systemu podzielono na iteracje, a po każdej z nich sprawdzano działanie systemu. Taka metoda zarządzania projektem pozwoliła na uniknięcie nawarstwiania się błędów w aplikacji – po każdej iteracji sprawdzano działanie dodatkowego elementu systemu i od razu reagowano na ewentualne problemy.

Zaplanowano następujące etapy prac:

1. Implementacja okna konfiguracji i menu głównego.
2. Stworzenie bazy danych i implementacja komunikacji z bazą oraz wstrzykiwania zależności (ang. *dependency injection*)
3. Implementacja widoku i warstwy logiki do zakładki „Sprawdź urządzenie” i komunikacji poprzez Bluetooth.
4. Implementacja widoku zakładki „Dodaj nowy bieg”, jej logiki, dodawania elementu do bazy danych i mechanizmu śledzenia lokalizacji.
5. Implementacja widoku zakładki „Archiwalne biegi” oraz odpowiednich metod pozwalających na sortowanie biegów i ich usuwanie z bazy danych.
6. Implementacja widoku zakładki „Statystyki”, dodanie logiki (m.in. rysowania wykresów) i metod pozwalających na sumowanie wartości w bazie danych.
7. Implementacja widoku i logiki zakładki „Ustawienia” oraz testowanie działania całej aplikacji, mające na celu znalezienie jak największej ilości błędów.

W dalszej części pracy opisano każdą z wyżej wymienionych iteracji oraz fragmenty kodu utworzonych w tym czasie klas, obiektów i metod.

**Pierwsza iteracja** skupiała się na utworzeniu widoku konfiguracji i menu głównego. Klasa ConfigurationFragment prowadzi do dwóch innych – MainActivity oraz MainMenu. Obie różnią się jedynie animacją. W ConfigurationFragment() sprawdzane są najpierw tzw. sharedpreferences (Listing 4.5) – zawierają one informację, czy jest to pierwsze, czy kolejne uruchomienie aplikacji (Listing 4.6).

---

#### Listing 4.5. Dodawanie dostępu do SharedPreferences

---

```
val sharedpreferences : SharedPreferences =
    getSharedPreferences(getString(R.string.app_name), MODE_PRIVATE)
```

---

**Listing 4.6.** Sprawdzanie wartości zmiennej w SharedPreferences

---

```
val isFirstUse: Boolean =
    sharedPreferences.getBoolean(getString(R.string.is_accessed), false)
```

---

Jeżeli zmienna isFirstUse zwróci wartość true (dzieje się tak jedynie w przypadku pierwszego uruchomienia aplikacji na danym urządzeniu mobilnym), aplikacja uruchomi odpowiedni widok konfiguracji i poprosi użytkownika o podanie imienia i obecnej wagi. Po sprawdzeniu poprawności danych wyświetli odpowiedni komunikat, zapisze dane do sharedPreferences (Listing 4.7) i w przypadku poprawnych danych wyświetli menu. W przypadku gdy isFirstUse zwróci wartość false, a więc gdy jest to kolejne uruchomienie, okno konfiguracji nie wyświetli się, a aplikacja przejdzie do widoku klasy MainActivity.

**Listing 4.7.** Przykład edycji danych w sharedPreferences

---

```
sharedPreferences.edit()
    .putString(SAVED_NAME, name)
    .putFloat(SAVED_WEIGHT, weight.toFloat())
    .apply()
```

---

Klasa MainActivity składa się głównie z implementacji widoku – początkowo zaprogramowano powitalną animację (Listing 4.8) oraz przyciski prowadzące do kolejnych widoków. Przykład sposobu implementacji przycisku zawarto na Listingu 4.9.

**Listing 4.8.** Implementacja animacji głównej aktywności

---

```
backgroundImageView.animate().translationY(-1900f).setDuration(2000).startDelay
    = 2200
textOnStart.animate().translationY(140f).alpha(0f).setDuration(2000).startDelay
    = 1500
menus.startAnimation(menuAnimation)
textHome.startAnimation(textMenuAnimation)
```

---

**Listing 4.9.** Przykładowa implementacja przycisku menu

---

```
buttonSettings.setOnClickListener {
    val intent = Intent(this, SettingsView::class.java)
    startActivity(intent)
}
```

---

**Druga iteracja** związana była przede wszystkim z opisany w rozdziale 3.8.1 wstrzykiwaniem zależności, a także implementacją dostępu do bazy danych. Prace nad tą częścią rozpoczęto od stworzenia bazy danych (Listing 4.10) – znacznym ułatwieniem było DAO. Jak widać na wspomnianym Listingu, korzystanie z DAO wiąże się z wieloma adnotacjami. Poniższa lista zawiera krótkie wyjaśnienie każdej z używanych adnotacji.

- `@Module` – klasa w której można dodać powiązania do typów, które nie mogą zostać wstrzyknięte przez konstruktor.
- `@InstallIn` – wskazuje, w których kontenerach zależności modułów muszą być dostępne.
- `@Singleton` – pojedyncza instancja klasy, unikalna dla każdego konkretnego komponentu i dostępna jedynie w jego zakresie.
- `@Provides` dodaje powiązanie dla typu interfejsu, który nie może być wstrzyknięty przez konstruktora. Typ który jest zwracany to typ powiązania, a parametry wskazują na zależności.
- `@ApplicationContext` – wstępnie zdefiniowane powiązanie – można go użyć jako zależności w odpowiednim kontenerze.
- `@HiltAndroidApp` rozpoczyna generowanie kodu Hilt.
- `@AndroidEntryPoint` dodaje kontener Dependency Injection.
- `@Inject` informuje, którego konstruktora użyć do tworzenia instancji danej klasy.
- `@Binds` to skrócona wersja dodawania powiązań dla typu (czyli adnotacji `@Provides`). Metoda z tą adnotacją musi być abstrakcyjna i znajdować się w module – wtedy Hilt nie będzie tworzył instancji modułu, a zamiast tego zwracał jedynie wstrzyknięty parametr, zapewniając większą wydajność.
- `@EntryPoint` – dodaje zależności w klasach, które nie są obsługiwane przez Hilt.

---

**Listing 4.10.** Stworzenie bazy danych

---

```
@Module
@InstallIn(ApplicationComponent::class)
object AppModule {
    @Singleton
    @Provides
    fun provideRunningDatabase(
        @ApplicationContext context: Context,
```

---

```

) = Room.databaseBuilder(
    context,
    RunDatabase::class.java,
    MAIN_DATABASE_NAME
).fallbackToDestructiveMigration().build()
...
}

```

---

Po utworzeniu bazy danych należało zdefiniować jej zawartość – wszystkie zmienne, które będą zawartością bazy danych umieszczone w klasie Run (Listing 4.12). Adnotacja `@Database` (Listing 4.11) określa zawartość encji, czyli elementów, w których dane powinny być zbierane i przechowywane w postaci właściwości i tabel oraz numer wersji bazy danych. W przypadku aktualizowania tabeli, czy zmiany ilości jej kolumn, zmieniony musi zostać również numer wersji. Warto dodatkowo zwrócić uwagę na fakt, że klasa Run to tak naprawdę klasa danych (ang. *data class*), czyli klasa nie zawierająca żadnych metod (poza standardowymi, wygenerowanymi przez kompilator), a jedynie zmienne.

**Listing 4.11.** Fragment kodu opisujący zawartość bazy danych

---

```

@Database(
    entities = [Run::class],
    version = 1
)
@TypeConverters(BitmapCreator::class)
abstract class RunDatabase : RoomDatabase() {
    abstract fun getRunDao(): RunDAO
}

```

---

**Listing 4.12.** Zawartość data class Run

---

```

@Entity(tableName = "running_table")
data class Run(
    var mapImage: Bitmap? = null,
    var timestamp: Long = 0L,
    var timeInMillis: Long = 0L,
    var avgSpeedInKMH: Float = 0f,
    var distanceInMeters: Int = 0,
    ...
    var avgLeft: Float = 0f,
    var maxLeft: Float = 0f
) {
    @PrimaryKey(autoGenerate = true)
    var id: Int? = null
}

```

---

Kolejnym etapem było stworzenie klasy RunDaoConstructor, której fragment jest widoczny na Listingu 4.13. Każda metoda tej klasy odnosi się do innego, zaimplementowanego z klasie RunDAO przemapowania zapytania w języku SQL na metodę Kotlin. Fragment klasy RunDAO zawarty jest na Listingu 4.14 – dzięki załączonemu fragmentowi widać, jak bardzo pomocne jest użycie DAO i jak łatwo można dzięki niemu korzystać z baz danych w aplikacjach mobilnych na system Android.

**Listing 4.13.** Fragment klasy RunDaoConstructor

```
class RunDaoConstructor @Inject constructor(private val runDao: RunDAO) {  
    suspend fun insertRun(run: Run) = runDao.insertRun(run)  
    suspend fun deleteRun(run: Run) = runDao.deleteRun(run)  
    fun getAllRunsSortedByDate() = runDao.getAllRunsSortedByDate()  
    fun getTotalAvgLeft() = runDao.getTotalAvgLeft()  
    ...  
}
```

**Listing 4.14.** Fragment kodu z użyciem DAO

```
@Dao  
interface RunDAO {  
    @Insert(onConflict = OnConflictStrategy.REPLACE)  
    suspend fun insertRun(run: Run)  
  
    @Delete  
    suspend fun deleteRun(run: Run)  
  
    @Query("SELECT * FROM running_table ORDER BY timestamp DESC")  
    fun getAllRunsSortedByDate(): LiveData<List<Run>>  
  
    @Query("SELECT AVG(avgSpeedInKMH) FROM running_table")  
    fun getTotalAvgSpeed(): LiveData<Float>  
    ...  
}
```

**Trzecia iteracja** dotyczyła przede wszystkim wizualizacji rozkładu nacisku oraz implementacji komunikacji poprzez Bluetooth. W związku z tym, że mechanizm wizualizacji szczegółowo opisano w rozdziale 4.4, poniżej skupiono się na przedstawieniu implementacji serwisu Bluetooth. Obiekt BluetoothController zawiera dwie istotne metody: `receiveData()`, dzięki której uruchamiany jest oddzielna korutyna i przechwytywanie wyjątków (widoczne na Listingu 4.15) oraz `connect()`, która odpowiada załączenie z odpowiednim urządzeniem i odbieranie wiadomości przez Bluetooth (Listing 4.16).

Korutyny są podobne do wątków, jednak nie zużywają tak dużej ilości zasobów. Nie są również związane bezpośrednio z żadnym z wątków, a zawieszenie działania korutyny nie powoduje zawieszenia działania całego wątku. Po zatrzymaniu działania korutyny, jej zasoby są zwracane do puli (z której mogą korzystać w tym czasie inne procesy). W przypadku ponownego uruchomienia, odtwarzana jest ona na dowolnym wątku – może być on inny od tego użytego za pierwszym razem. W przypadku niniejszego projektu główną częścią korutyny jest uruchomienie funkcji `connect()` z parametrem odpowiedniego urządzenia oraz zamknięcie połączenia w przypadku odpowiedniego komunikatu

**Listing 4.15.** Fragment kodu zawierający korutynę

---

```
coroutineScope.launch(Dispatchers.IO) {
    try {
        connect(device.toString())
    } catch (e: CancellationException) {
        Log.i(TAG, "CANCELLED ${bluetoothSocket?.isConnected}")
    } finally {
        bluetoothSocket?.close()
        bluetoothSocket = null
    }
}
```

---

Kolejnym istotnym elementem było zaimplementowanie metody `connect()`, która pozwalały na odbiór ramek danych z mikrokontrolera. Pojawił się tu pierwszy poważny problem – urządzenie mobilne nie zawsze odbierało całą ramkę 18 bitów danych. Zdarzało się, że najpierw pobierane było przykładowo pięć pierwszych bitów, a następnie kolejne trzynaście. Dla bezpieczeństwa, należało również doimplementować mechanizm sprawdzający poprawność ramki – czyli czy ostatni bit ramki to '`_n`'. Dane odczytywane są więc w pętli, która jest przerywana w przypadku nieodebrania żadnej ramki, poprawnego sklejenia pełnej ramki oraz przekroczenia długości pełnej ramki (oznaczałoby to niepoprawnie sklejoną ramkę). Za poprawną ramkę uznawana jest taka, która składa się z 18 elementów oraz której ostatni element jest bitowym odpowiednikiem liczby 10 (zgodnie z tablicą ASCII '`_n`' odpowiada wartości 10). Dodane opóźnienie 50 ms pozwala na zaoszczędzenie pracy aplikacji (tym samym baterii smartfona), przy jednokrotnej pewności, że żadna ramka nie zostanie pominięta.

**Listing 4.16.** Fragment kodu przedstawiający mechanizm odbierania wiadomości przez Bluetooth

---

```
while (bluetoothSocket.isConnected) {
    val outputStream = ByteArrayOutputStream()
    while (true) {
        var mmBuffer = ByteArray(32)
        val count = mmInStream.read(mmBuffer)
        if (count <= 0) {
```

---

```
        outputStream.flush()
        outputStream.close()
        break
    }
    outputStream.write(mmBuffer, 0, count)
    if (outputStream.size() == 18 && outputStream.toByteArray()[17] ==
        10.toByte()) {
        bluetoothData.postValue(outputStream.toByteArray())
        outputStream.flush()
        outputStream.close()
        break
    } else if (outputStream.size() > 18) {
        outputStream.flush()
        outputStream.close()
        break
    }
    delay(50)
}
}
```

Odebranie danych to nie wszystko – należy jeszcze dodać odświeżanie widoku rozkładu nacisku i dostęp do tych danych w odpowiedniej klasie. W tym celu użyta została metoda `observe()`, która zauważa zmiany w zmiennych typu `LiveData` (w przypadku niniejszego projektu jest to `controllerData`) i w razie zmian wywołuje metodę `fillFootView`, widoczną na Listingu 4.17. Metoda ta odwołuje się do odpowiedniej metody klasy zajmującej się rysowaniem rozkładu nacisku stóp, jako parametr podając dane odebrane przez Bluetooth (zmienione do postaci tablicy zmiennych typu `int`).

**Listing 4.17.** Przekazywanie danych do widoku rozkładu nacisku

```
BluetoothController.controllerData.observe(this, ::fillFootView)
private fun fillFootView(bytes: ByteArray?) {
    rightFootImage.updateControllerData(bytes?.take(16)?.map{
        it.toInt() - 40
    }?.toIntArray() ?: IntArray(16))
    ...
}
```

**Czwarta iteracja** powiązana była z zakładką „Dodaj nowy bieg”. Widok tej części aplikacji budowany jest w klasie `TrackingView`, ale zawiera ona również metody które między innymi pozwalają na obsługę zatrzymywania i wznowiania biegu, anulowania jego zapisu, przesuwania mapy (na której rysuje się widok przebytej trasy). Klasa ta umożliwia również zapis treningu do bazy danych. Drugą ważną klasą dotyczącą tej części projektu, jest klasa `TrackingService`.

Odpowiada ona za całe śledzenie trasy i poszczególnych jej parametrów – zarówno lokalizacji i czasu jej trwania, jak i średniego i maksymalnego nacisku każdej ze stóp. Wartości wspomnianych parametrów zapisywane są do typu `LiveData`. `LiveData` to klasa typu `data holder`, która służy do obserwowania zmian danych i ich aktualizowania. Dodatkową zaletą jest fakt, że dane są zmieniane jedynie w aktywnych elementach aplikacji, co dodatkowo chroni aplikację przed błędami wynikającymi z wysyłania danych do interfejsu użytkownika. Przykład modyfikowalnej zmiennej typu `LiveData` zaprezentowano na Listingu 4.18.

**Listing 4.18.** Przykładowa mutowalna zmienna typu `LiveData`

---

```
val tableAvgRight = MutableLiveData<Float>()
tableAvgRight.postValue(0.0F)
```

---

Do tak stworzonej zmiennej można odnieść się za pomocą metody `observe()` opisanej w poprzedniej iteracji. Podobnie jak w przypadku implementacji komunikacji przez Bluetooth, do śledzenia lokalizacji i wszystkich parametrów biegu korzystamy z korutyn. Dzięki odpowiedniej implementacji tego rodzaju wątku trening nie będzie automatycznie kończony przy wyjściu z odpowiedniej zakładki, czy nawet zamknięciu całej aplikacji.

**Piąta iteracja** pozwoliła na utworzenie widoku zawierającego listę archiwalnych treningów, którą w razie potrzeby można posortować po wybranym parametrze. Metoda `onBindViewHolder` jest częścią widoku `RecyclerView` – sprawia ona, że poszczególne części `RecyclerView` są umieszczane w odpowiednim miejscu, zaoszczędza również użycie pamięci i przyspiesza ładowanie kolejnych elementów. Listing 4.19 zawiera fragment klasy `RunsAdapter` (w tym metodę `onBindViewHolder()`) i ukazuje metodę tworzenia fragmentów poszczególnych archiwalnych biegów. Na wspomnianym Listingu widoczny jest sposób identyfikowania pozycji (na jej podstawie wiadomo również, który bieg należy usunąć z bazy danych) oraz to jak tworzona i zapisywana jest data biegu.

**Listing 4.19.** Fragment funkcji `onBindViewHolder()`

---

```
class RunsAdapter : RecyclerView.Adapter<RunsAdapter.RunViewHolder>() {
    inner class RunViewHolder(itemView: View) :
        RecyclerView.ViewHolder(itemView) {
        override fun onBindViewHolder(holder: RunViewHolder, position: Int) {
            val run = differ.currentList[position]
            holder.itemView.apply {
                Glide.with(this).load(run.mapImage).into(itemRunImage)
                val calendar = Calendar.getInstance().apply {
                    timeInMillis = run.timestamp
                }
            }
        }
    }
}
```

---

---

```

        val dateFormat = SimpleDateFormat("dd.MM.yy", Locale.getDefault())
        viewDate.text = dateFormat.format(calendar.time)
        ...
    }
}
}
}

```

---

Klasa RunsView odpowiada za wyświetlanie listy biegów, sortowanie jej według wybranego parametru oraz usuwanie w razie potrzeby. Listing 4.20 zawiera fragment wspomnianej klasy odpowiadający za usuwanie biegu. W tym celu użyta została metoda `onSwiped()`, co oznacza że bieg będzie usuwany przy próbie przesunięcia biegu w prawo lub lewo względem ekranu. Początkowo identyfikowana jest pozycja na której znajduje się bieg, aby następnie znaleźć odpowiedni bieg po jego numerze ID. Następnie bieg jest usuwany z bazy danych i wyświetlany jest odpowiedni komunikat.

**Listing 4.20.** Usuwanie biegu z poziomu interfejsu użytkownika

---

```

override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
    val position = viewHolder.adapterPosition
    val runEntry: Run = runningAdapter.getRunEntry(position)
    viewLifecycleOwner.lifecycleScope.launch {
        viewModel.deleteRun(runEntry)
        Toast.makeText(this, "Poprawnie usunięto bieg",
            Toast.LENGTH_LONG).show()
    }
}

```

---

Kolejnym elementem jest sortowanie biegów – Listing 4.21 zawiera fragmenty klasy `RunsViewModel()`, której zdefiniowane są metody pozwalające na sortowanie według wybranego przez użytkownika parametru.

**Listing 4.21.** Fragment kodu zawierający startową konfigurację

---

```

class RunsViewModel @ViewModelInject constructor(
    private val runDaoConstructor: RunDaoConstructor,
) : ViewModel() {
    private val runsSortedByAvgRight =
        runDaoConstructor.getAllRunsSortedByAvgRight()
    private val runsSortedByMaxRight =
        runDaoConstructor.getAllRunsSortedByMaxRight()
    ...

    init {
        runs.addSource(runsSortedByDate) { result ->
            if (sortType == SortingDataType.DATE) {

```

---

---

```

        result?.let { runs.value = it }
    }
}
...
}
}
}

```

---

**Szósta iteracja** odnosiła się do ostatniej z ważnych zakładek – „Statystyk”. Widok tej zakładki powinien składać się z ośmiu podsumowanych parametrów:

- całkowitego dystansu pokonanego w czasie wszystkich treningów,
- całkowitego czasu tych treningów,
- całkowitej ilości spalonych kalorii,
- sumarycznej średniej prędkości,
- sumarycznej średniej nacisku prawej oraz lewej stopy,
- całkowitego maksymalnego nacisku na prawą oraz lewą stopę.

Dodatkowo w tej zakładce znajdują się trzy wykresy średnich nacisku i prędkości podczas każdego z treningów. W tej iteracji zaimplementowano dwie klasy: `StatisticView` odpowiadającą za wyświetlanie parametrów i rysowanie wykresów oraz `StatisticViewModel` pobierającą z bazy danych odpowiednie podsumowania parametrów (fragment tej klasy jest widoczny na Listingu 4.22)

**Listing 4.22.** Fragment klasy StatisticViewModel

---

```

class StatisticViewModel @ViewModelInject constructor(
    runDaoConstructor: RunDaoConstructor,
) : ViewModel() {
    val totalTimeRun = runDaoConstructor.getTotalTimeInMillis()
    val totalCaloriesBurned = runDaoConstructor.getTotalCaloriesBurned()
    val totalAvgSpeed = runDaoConstructor.getTotalAvgSpeed()
    val runsSortedDate = runDaoConstructor.getAllRunsSortedByDate()
    ...
}

```

---

Po przygotowaniu `StatisticViewModel()` należało stworzyć klasę `StatisticView`. W tym celu konieczne było zaimplementowanie metod, które pozwolą na odświeżanie danych po każdym kolejnym dodaniu lub usunięciu biegu. Listing 4.23 przedstawia fragment funkcji, dzięki której wszystkie wymienione wyżej parametry będą na bieżąco aktualizowane (o zmianie informowana jest za pomocą metody `observe()`). Listing 4.24 ukazuje natomiast sposób aktualizowania danych znajdujących się na wykresie.

**Listing 4.23.** Przykład aktualizowania danych w tabeli

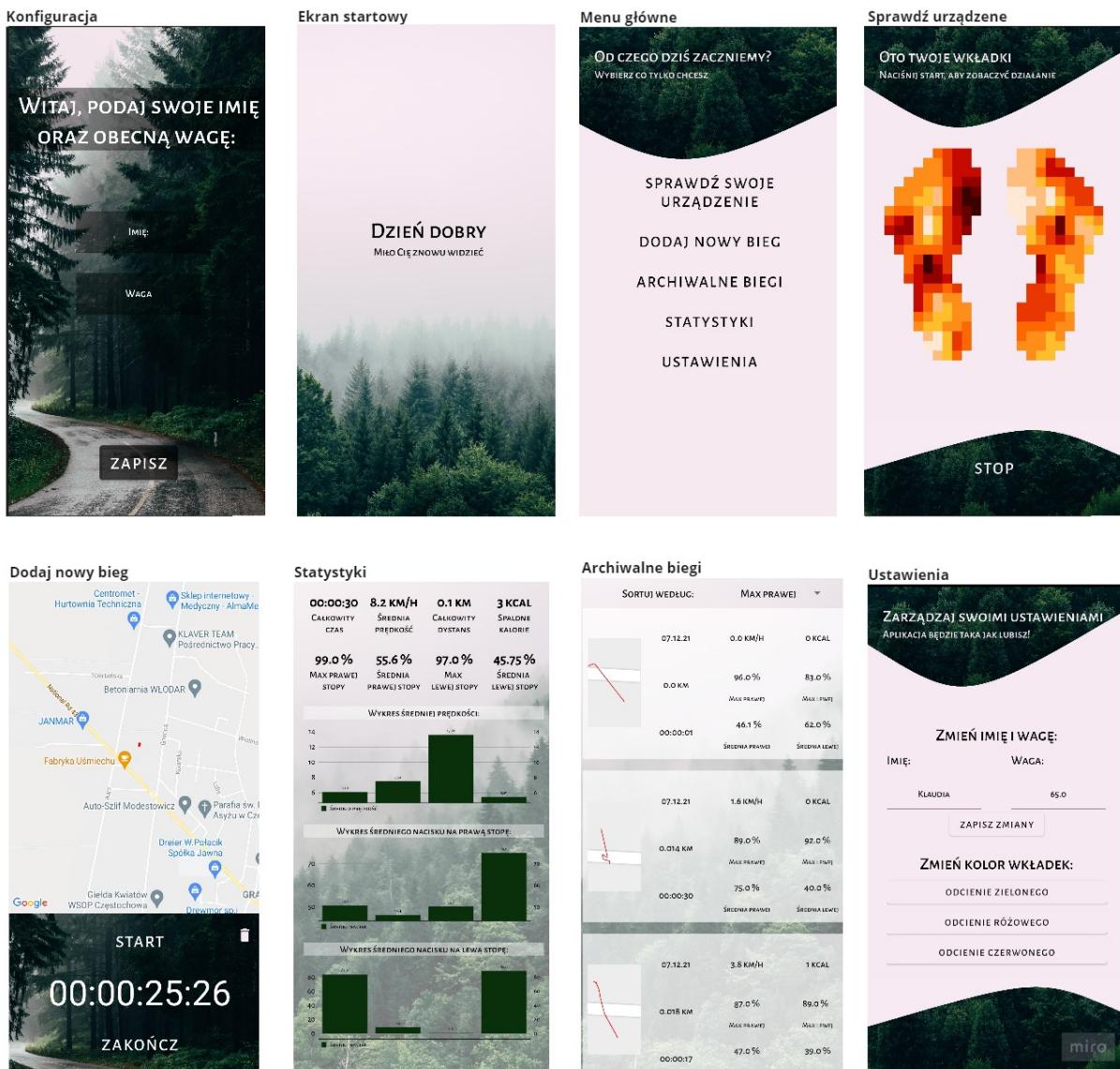
```
private fun subscribeToObservers() {
    viewModel.totalTimeRun.observe(viewLifecycleOwner, {
        it?.let {
            val totalTimeRun =
                TrackingBenefit.getFormattedStoppedWatchTime(it)
            viewTotalTime.text = totalTimeRun
        }
    })
    ...
}
```

**Listing 4.24.** Przykład aktualizowania wykresu

```
viewModel.runsSortedDate.observe(viewLifecycleOwner, { it ->
    it?.let { it ->
        val allAvgSpeed = it.indices.map { i~-> BarEntry(i.toFloat(),
            it[i].avgSpeedInKMH) }
        val barDataSet = BarDataSet(allAvgSpeed,
            getString(R.string.average_speed)).apply {
            valueTextColor = Color.BLACK
            color = ContextCompat.getColor(requireContext(),
                R.color.colorAccent)
            valueTypeface = context?.let { ResourcesCompat.getFont(it,
                R.font.alegreya_sans_sc_medium) }
        }
        avgSpeedBarChart.data = BarData(barDataSet)
        avgSpeedBarChart.invalidate()
        ... // Miesjce na zaktaulizowanie reszty wykresów
    }
}
```

**Ostatnia iteracja** składała się z dwóch części – implementacji zakładki „Ustawienia” oraz poprawa działania całej aplikacji. Widok ustawień, zgodnie z mapą ekranów prezentowaną w rozdziale 2.6, składa się z trzech przycisków umożliwiających zmianę koloru prezentacji widoku nacisku. We wspomnianej zakładce znajdują się również takie same pola tekstowe, jak w widoku konfiguracji i podobnie wykorzystywane są Shared Preferences.

Podczas całej implementacji natrafiono na kilka problemów, najtrudniejszą częścią była implementacja działania fragmentów aplikacji na różnych wątkach – korutynach, dzięki czemu możliwe jest działanie aplikacji w tle oraz implementacja działającego systemu odbierania danych poprzez Bluetooth. Rys. 4.6 przedstawia efekty pracy nad aplikacją mobilną:



Rys. 4.6. Zrzuty ekranów aplikacji mobilnej [20, 21]

## 4.4. Wizualizacja danych

Ostatnim elementem prac jest dodanie algorytmu wizualizacji tego, jak wygląda rozkład nacisku każdej ze stóp. Prezentacja rozkładu nacisku zaimplementowana jest w klasach: `LeftFootView`, `RightFootView` oraz obiekcie `FunctionsForFootView`. Obiekt `FunctionsForFootView` przechowuje funkcje z których korzystają wyżej wymienione klasy. One same natomiast odpowiadają za przekształcenie tablicy danych typu całkowitoliczbowego `int` na widok rozkładu nacisku.

`LeftFootView` (podobnie jak `RightFootView`) składa się z dwóch podstawowych funkcji – nadpisywanej funkcji `onDraw` dziedziczonej po klasie `View` oraz `assignDataToFoot`. Obiekt `FunctionsForFootView` zawiera w sobie dwie funkcje pomocnicze – `createColors` oraz funkcję `calculateAverageForEmptyPoints`. Aby lepiej wyjaśnić sposób i cel działania wspomnianych metod (`onDraw`, `assignDataToFoot`, `calculateAverageForEmptyPoints`, `createColors`), poniżej opisano każdą z nich.

**onDraw** – funkcja wywoływana jest cyklicznie od momentu naciśnięcia przycisku „START” w odpowiedniej zakładce aplikacji, aż do wciśnięcia przycisku „STOP”. Każdorazowo wczytuje ona czystą tablicę, zawierającą jedynie informacje, o tym w którym miejscu znajduje się wkładka i czujniki nacisku. Następnie pobierane są dane pochodzące z czujników nacisku, które w następnej kolejności są przetwarzane przez funkcje `assignDataToFoot` i `calculateAverageForEmptyPoint`. Efektem uruchomienia obu tych funkcji jest dwuwymiarowa tablica danych zawierająca informację, o tym jaki jest nacisk w danym punkcie wkładki. Na koniec używając zagnieżdzonej pętli, każdemu punktowi przypisano kolor oraz narysowano go na płaszczyźnie.

**assignDataToFoot** – to prosta funkcja, która jako dane wejściowe przyjmuje dwie tablice – tablicę (1) z danymi z czujników oraz dwuwymiarową tablicę (2), na podstawie której później rysowana jest wizualizacja rozkładu nacisku. Funkcja ta przypisuje odpowiednim miejscom w tablicy (2) wartości z czujników nacisku, a następnie zwraca tablicę (2). Na wyjściu tablica (2) zawiera następujące dane:

- 0 – w miejscu gdzie nie ma wkładki,
- 1 – w miejscu gdzie jest wkładka, ale nie ma czujnika nacisku,
- wartości z czujników nacisku.

**calculateAverageForEmptyPoint** – funkcja, która odnajduje wartości pól w miejscach, w których nie znajduje się czujnik. Jako daną wejściową otrzymuje całą tablicę widoku rozkładu nacisku i tę samą tablicę zwraca. Funkcja przechodzi po każdym elemencie tablicy. Jeżeli wartość tego pola jest równa 1 (czyli jest to miejsce, w którym jest wkładka, ale nie ma czujnika nacisku) to liczona jest średnia z okolicznych pól (pod warunkiem, że istnieją, są niezerowe i różne od 1). Nie uwzględnia się miejsc o wartości równej zero, ponieważ w tych miejscach nie można mówić o nacisku (nie ma tam stóp). Pola o wartości równej jeden nie sąbrane pod uwagę, ponieważ oznacza to, że algorytm jeszcze nie wykonał żadnych działań w tych miejscach. W przypadku, gdy żadne z okolicznych pól nie spełnia kryteriów do policzenia średniej, punktowi przypisywana jest wartość 1.

**createColors** – to funkcja, która przydziela kolor wskazanemu punktowi na podstawie jego wartości. Jako daną wejściową przyjmuje właśnie daną liczbową – wartość odpowiedniego

elementu tablicy, zwraca natomiast kod odpowiedniego koloru. Funkcja każdorazowo pobiera skalę kolorów w formie listy, by reagować na zmiany ustawień użytkownika.

Wspomniane funkcje realizują wizualizację rozkładu nacisku, która była ostatnim elementem implementacji oprogramowania. Początkowo zwrócono uwagę na najważniejsze elementy oprogramowania mikrokontrolera oraz sposób przetwarzania odebranych danych. Następnie przyjrzano się najistotniejszym elementom kodu aplikacji mobilnej i wspomnianej już wizualizacji nacisku. Po zakończeniu prac nad tworzeniem systemu, rozpoczęto fazę testów.

## **5. Testy**

Testowanie to nie tylko uruchamianie programu w celu sprawdzenia, czy aplikacja jest wydajna i czy działa w całości. Aby dokładnie ocenić jakość programu i zmniejszyć ryzyko awarii należy dokładnie przeanalizować problem i przyjrzeć się działaniu poszczególnych części systemu. W pierwszej części tego rozdziału zaprezentowano testy jednostkowe, które sprawdzają działanie poszczególnych, oddzielnych modułów systemu. Najważniejszą częścią pracy nad testami jest zidentyfikowanie obszarów i cech, których dokładne sprawdzenie jest szczególnie ważne. Mowa tu o:

- działaniu modułu Bluetooth,
- poborze prądu,
- zamianie danych liczbowych na tekstowe i odwrotnie,
- prezentacji graficznej wkładki,
- działania poszczególnych modułów aplikacji mobilnej.

### **5.1. Testy modułowe**

Po określaniu zbiorów cech do przetestowania, należy określić warunki testowe, czyli odpowiedzieć na pytanie „Co należy przetestować?”. Poniżej znajduje się lista warunków testowych wraz z informacjami, jaki był dokładny przypadek testowy – czyli jak sprawdzono działanie danego elementu systemu.

1. Czy używane baterie są wystarczające do zapewnienia prawidłowego działania całego systemu?

W celu odpowiedzi na to pytanie przeprowadzono doświadczenie. Z dokumentacji [22] wynika, że moduł Bluetooth w czasie prób parowania pobiera najwięcej prądu – ok. 80 mA. Doświadczalnie sprawdzono, że podczas ciągłej transmisji (wysyłanie danych

400 razy w ciągu minuty), bez wprowadzania rozwiązań mających na celu zaoszczędzenie energii, moduł Bluetooth pobiera 25 mA. Mikroprocesor odciążony przez użycie bezpośredniego przydziału pamięci DMA (ang. *Direct Memory Access*) potrzebuje stosunkowo niewiele prądu – są to maksymalnie 2 mA.

Największym problemem przy odpowiednim doborze baterii było napięcie wejściowe – zgodnie z dokumentacją STM32-Nucleo [23] napięcie na kanale VIN musi mieścić się w zakresie 7-12 V. Ponieważ napięcie baterii spada wraz z jej rozładowywaniem, przy zastosowaniu 2 baterii 3.7 V połączonych szeregowo nie zostało wiele zapasu (0.4 V).

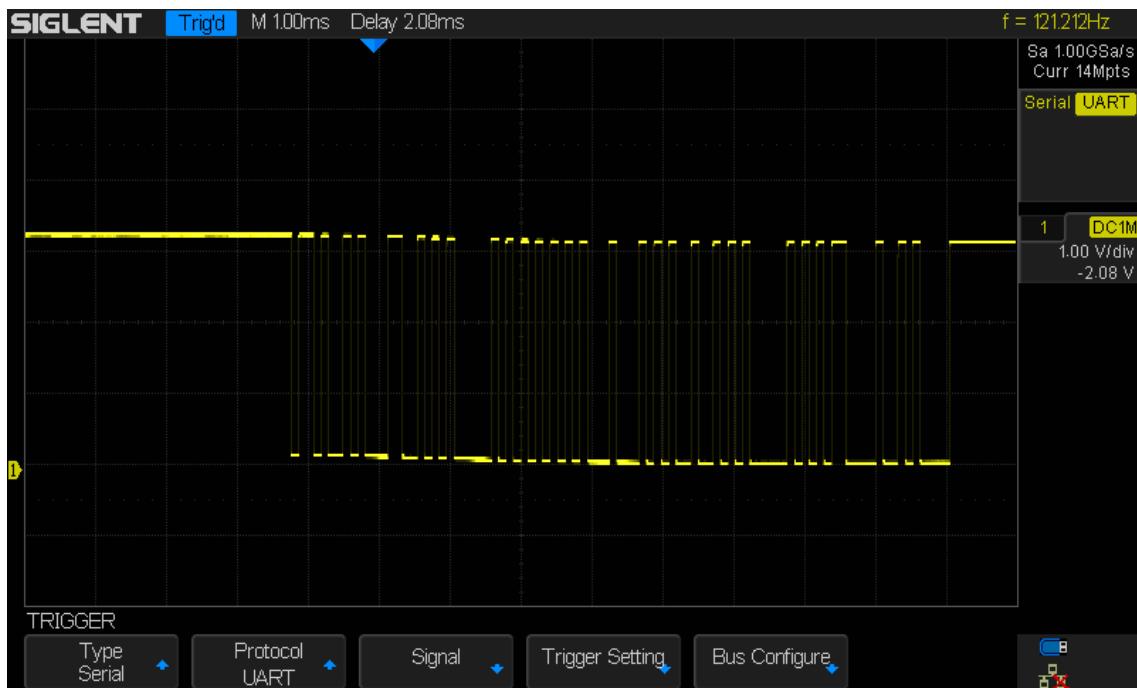
Podpięte urządzenie pozostało na 12 godzin testów – przez cały ten czas moduł próbował sparować się z innym urządzeniem, a więc zużywał ok. 3 razy więcej energii. Można więc uznać, że system został przetestowany w najgorszej możliwej sytuacji, a wynik testu jest pozytywny. Po wskazanym wyżej czasie, najniższe zarejestrowane na bateriach napięcie to 7.2 V, co oznacza, że użyte baterie są wystarczające do zapewnienia funkcjonowania systemu.

## 2. Czy bluetooth przesyła jakiekolwiek informacje?

Aby sprawdzić, czy system wysyła jakiekolwiek informacje poprzez moduł Bluetooth, użyto oscyloskopu. Urządzenie podpięto pod kanały RX oraz masę (GND), następnie uruchomiono program. Jak widać na załączonych zrzutach ekranu z oscyloskopu (Rys. 5.1 oraz Rys. 5.2), moduł Bluetooth wysyła dane w postaci bitów.



Rys. 5.1. Nadanie „AT”



Rys. 5.2. Nadanie „KLAUDIA”

### 3. Czy bluetooth przesyła odpowiednie dane?

Skoro udało się potwierdzić, że moduł Bluetooth działa i wysyła dane, postanowiono sprawdzić, czy są one poprawne. Aby nie odczytywać kolejnych bitów ręcznie i uniknąć pomyłki, urządzenie podpięto do analizatora stanów logicznych firmy Saleae [24]. Program Logic firmy Saleae odpowiadał za odczyt danych bitowych i przedstawienie ich wartości z tabeli ASCII. Test wykazał, że dane przesyłane przez moduł Bluetooth są właściwe – program Logic wyświetlił poprawnie słowa, które zostały zakodowane w programie mikrokontrolera.

### 4. Czy w czasie braku pracy wkładki włączone jest oszczędzanie energii?

Moduł STM32L010RB nie został wybrany przypadkowo – „L” wiąże się z niskim zużyciem baterii. Niskie zużycie baterii jest ważne zarówno dla wygody użytkownika i trwałości urządzenia, jak i ekonomii oraz ekologii.

Aby dodatkowo zaoszczędzić baterię, zdecydowano, że w czasie, gdy wkładki nie odczytują żadnego nacisku (a pozostały włączone) nie powinny wysyłać żadnych danych przez moduł Bluetooth. W celu sprawdzenia powyższej funkcjonalności wykonano test, polegający na ręcznym wpisaniu do programu w języku C wartości, które normalnie odczytywane są w czasie braku ruchu. W tym czasie mikrokontroler był połączony poprzez moduł Bluetooth ze smartfonem, a mimo tego aplikacja nie odbierała żadnych danych. Po zmianie dowolnego bitu na wyższy (imitując nacisk na czujnik), Bluetooth

automatycznie wyłączył tryb uśpienia i przekazał aplikacji mobilnej odpowiednie dane. Efekty testu przedstawiono na zrzutach ekranu (Rys. 5.3, Rys. 5.4).

Zmienna `max` oznacza maksymalną wartość spośród pobranych z czujników. Jeżeli jest ona mniejsza niż 10, oznacza to że najwyższy odczytany wynik jest mniejszy niż 10% maksymalnej wartości – takie dane można uznać za brak ruchu. Zmienna `flag_IfTransmit` przyjmuje wartość 0, gdy transmisja przez Bluetooth nie została wykonana oraz 1, gdy wysłano dane.

Name	Type	Value
> <code>adcAsString</code>	char [3]	0x20004fe4
> <code>finalString</code>	char [50]	0x20004fb0
↳ <code>flag_IfTransmit</code>	uint8_t	0 '\0'
↳ <code>i</code>	uint8_t	16 '\020'
↳ <code>max</code>	uint8_t	3 '\003'

Expression	Type	Value
↳ <code>adcValuesInScale</code>	uint16_t [16]	0x200000b0 <adcValuesInScal...
↳ <code>adcValuesInScale[0]</code>	uint16_t	1
↳ <code>adcValuesInScale[1]</code>	uint16_t	1
↳ <code>adcValuesInScale[2]</code>	uint16_t	2
↳ <code>adcValuesInScale[3]</code>	uint16_t	1
↳ <code>adcValuesInScale[4]</code>	uint16_t	1
↳ <code>adcValuesInScale[5]</code>	uint16_t	1
↳ <code>adcValuesInScale[6]</code>	uint16_t	1
↳ <code>adcValuesInScale[7]</code>	uint16_t	2
↳ <code>adcValuesInScale[8]</code>	uint16_t	3
↳ <code>adcValuesInScale[9]</code>	uint16_t	3
↳ <code>adcValuesInScale[10]</code>	uint16_t	1
↳ <code>adcValuesInScale[11]</code>	uint16_t	1
↳ <code>adcValuesInScale[12]</code>	uint16_t	1
↳ <code>adcValuesInScale[13]</code>	uint16_t	1
↳ <code>adcValuesInScale[14]</code>	uint16_t	1
↳ <code>adcValuesInScale[15]</code>	uint16_t	1
↳ <code>flag_IfTransmit</code>	uint8_t	0 '\0'

Rys. 5.3. Zrzut ekranu – brak transmisji

Name	Type	Value
> <code>adcAsString</code>	char [3]	0x20004fe4
> <code>finalString</code>	char [50]	0x20004fb0
↳ <code>flag_IfTransmit</code>	uint8_t	1 '\001'
↳ <code>i</code>	uint8_t	16 '\020'
↳ <code>max</code>	uint8_t	92 '\1'

Expression	Type	Value
↳ <code>adcValuesInScale</code>	uint16_t [16]	0x200000b0 <adcValuesInScal...
↳ <code>adcValuesInScale[0]</code>	uint16_t	2
↳ <code>adcValuesInScale[1]</code>	uint16_t	3
↳ <code>adcValuesInScale[2]</code>	uint16_t	2
↳ <code>adcValuesInScale[3]</code>	uint16_t	1
↳ <code>adcValuesInScale[4]</code>	uint16_t	1
↳ <code>adcValuesInScale[5]</code>	uint16_t	1
↳ <code>adcValuesInScale[6]</code>	uint16_t	1
↳ <code>adcValuesInScale[7]</code>	uint16_t	1
↳ <code>adcValuesInScale[8]</code>	uint16_t	1
↳ <code>adcValuesInScale[9]</code>	uint16_t	1
↳ <code>adcValuesInScale[10]</code>	uint16_t	1
↳ <code>adcValuesInScale[11]</code>	uint16_t	1
↳ <code>adcValuesInScale[12]</code>	uint16_t	1
↳ <code>adcValuesInScale[13]</code>	uint16_t	1
↳ <code>adcValuesInScale[14]</code>	uint16_t	1
↳ <code>adcValuesInScale[15]</code>	uint16_t	92
↳ <code>flag_IfTransmit</code>	uint8_t	1 '\001'

Rys. 5.4. Zrzut ekranu – transmisja

## 5. Czy program odpowiednio szyfruje dane do postaci tekstowej?

W celu przesłania danych z mikrokontrolera przez Bluetooth, potrzebna jest tablica zmiennych typu `char`, o znanej długości. Dane odczytywane z kanałów ADC to dane liczbowe z zakresu 0-100%. Jeden element tablicy to 1 bit danych, czyli możliwość przesłania liczby między 0-255. Zamieniono więc daną liczbową na zmienną typu `char`, np. liczba 40 została wysłana jako '(' – zgodnie z systemem kodowania znaków ASCII [25].

Aby sprawdzić, czy wskazany algorytm działa prawidłowo, podano kilka przykładowych danych liczbowych i sprawdzono jaki jest efekt konwersji na zmienne typu `char`.

Tablica `adcValuesInScale100` przechowuje wartości pobrane z czujników danych w skali 0-100%, a zmienna `finalString` tablicę zmiennych typu `char`, która zostanie

przesłana przez moduł Bluetooth do aplikacji mobilnej. Po porównaniu danych zaprezentowanych na zrzucie ekranu (widocznym na rys. 5.5) z tablicą ASCII [25], widać, że dane konwertowane są prawidłowo.

adcValuesInScale100	uint16_t [16]	0x200000d4 <adcValuesInScal...
↳ adcValuesInScale100[0]	uint16_t	1
↳ adcValuesInScale100[1]	uint16_t	1
↳ adcValuesInScale100[2]	uint16_t	2
↳ adcValuesInScale100[3]	uint16_t	1
↳ adcValuesInScale100[4]	uint16_t	82
↳ adcValuesInScale100[5]	uint16_t	2
↳ adcValuesInScale100[6]	uint16_t	2
↳ adcValuesInScale100[7]	uint16_t	61
↳ adcValuesInScale100[8]	uint16_t	88
↳ adcValuesInScale100[9]	uint16_t	2
↳ adcValuesInScale100[10]	uint16_t	72
↳ adcValuesInScale100[11]	uint16_t	1
↳ adcValuesInScale100[12]	uint16_t	1
↳ adcValuesInScale100[13]	uint16_t	2
↳ adcValuesInScale100[14]	uint16_t	2
↳ adcValuesInScale100[15]	uint16_t	2
finalString	char [18]	0x20004fd4
<a href="#">+ Add new expression</a>		

Rys. 5.5. Zrzut ekranu konwersji danych do zmiennych typu char

## 6. Czy smartfon odbiera dane z Bluetooth?

Kolejnym elementem niniejszego systemu jest odbieranie danych wysyłanych z mikrokontrolera przez urządzenie mobilne. Należało sprawdzić czy odbierane dane są poprawne: czy urządzenie odbiera takie same dane oraz czy są one odbierane w odpowiedniej kolejności. W tym celu kilkakrotnie przeprogramowano mikrokontroler, wpisując w kod wartości, które powinny zostać wysłane, a następnie porównywano je z tymi odbieranymi przez aplikację mobilną. Test wykazał, że dane są odbierane w poprawny sposób, z zachowaniem właściwej kolejności kolejnych bitów.

7. Czy aplikacja odpowiednio rozszyfrowuje dane do postaci liczbowej?

Biorąc pod uwagę, że wiemy już, że dane są odpowiednio konwertowane i wysyłane, a także odbierane przez aplikację, należało sprawdzić, czy ponowna konwersja na system

liczbowy również działa prawidłowo. W tym celu pominięto odbiór danych, podano przykładową tablicę zmiennych typu `char` odebraną przez aplikację. Sprawdzono, czy efektem działania algorytmu jest powstanie tablicy zmiennych typu `int` zawierającej odpowiednie wartości liczbowe – to znaczy dokładnie takie, jakie zakodowano wcześniej w oprogramowaniu mikrokontrolera.

Zgodnie z tym, co widoczne jest na zrzucie ekranu na Rys. 5.6, test powiodł się – dane w postaci tekstuowej są odpowiednio odczytywane i konwertowane do danych liczbowych.

Rys. 5.6. Konwersja danych odebranych przez Bluetooth

8. Czy archiwalne biegi wyświetlane są prawidłowo, a bieg jest usuwany po przesunięciu?

Dokładne przetestowanie zakładki „Archiwalne biegi” umożliwia zarówne sprawdzenie połączenia z bazą danych, możliwości dodawania i usuwania kolejnych biegów, jak również poprawności poleceń języka SQL, a co za tym idzie poprawności sortowania danych.

Sprawdzono następujące funkcjonalności:

- Czy po zapisaniu biegu jest on automatycznie dodawany do archiwalnych biegów?
  - Czy po przesunięciu widoku pojedynczego biegu poza ekran (ang. *swipe*) bieg jest usuwany z bazy danych?
  - Czy dane sortowane są w odpowiedni sposób?

Po sprawdzeniu powyższych funkcjonalności test uznano za zakończony – wszystkie wskazane elementy zadziałyły poprawnie.

9. Czy widok wkładki wyświetlany jest prawidłowo?

Kolejnym istotnym elementem jest wyświetlanie widoku rozkładu nacisku stóp. Aby przetestować, czy widok wyświetla się prawidłowo niezależnie od danych wejściowych kilkakrotnie uruchamiano aplikację, za każdym razem podając inne przykładowe dane wejściowe (efekt widoczny na Rys. 5.7). Testy dowiodły, że niezależnie od tego, czy wszystkie wartości są równe 0%, 100%, czy też każda jest inną, widok wkładki jest prawidłowy.



Rys. 5.7. Zrzuty ekranu z aplikacji mobilnej z widokiem rozkładu nacisku

## 5.2. Testy systemowe

Po przejściu przez testy poszczególnych modułów systemu, warto sprawdzić, czy cały system również działa prawidłowo. W tym celu poproszono dziesięciu użytkowników systemu Android o zainstalowanie aplikacji na swoim smartfonie i przetestowanie działania aplikacji. Należy dodać, że użytkownicy korzystali z różnych wersji oprogramowania Android i telefonów z różnymi rozmiarami ekranów. Po 20 minutach używania wkładek, każdego z użytkowników poproszono o wypełnienie krótkiej ankiet dotyczącej oceny działania systemu. Mieli oni ocenić poszczególne elementy systemu za pomocą skali 1-5, gdzie 1 oznaczało „bardzo źle”, a 5 „bardzo dobrze”. Zbiorcze wyniki zaprezentowano Tabeli 5.1, a dodatkowe komentarze użytkowników w dalszej części pracy.

Po zakończeniu testów przeanalizowano wyniki:

- Wygoda systemu – użytkownicy bardzo dobrze ocenili wygodę samej wkładki – doceniali fakt, że jest ona bardzo cienka i praktycznie niewyczuwalna w czasie biegu. Cały system jest natomiast dość duży, a jego wagi (w szczególności dużych baterii) nie można nie odczuć w czasie dłuższego biegu, co skutkowało średnią oceną tego elementu.
- Działanie Bluetooth – użytkownicy nie mieli żadnych zastrzeżeń do tej części systemu.

L. p.	Wygoda systemu	Działanie Bluetooth	Dokładność pomiarów	Wygląd aplikacji	Działanie aplikacji	Chęć dalszego korzystania
<b>1</b>	4	5	5	5	5	5
<b>2</b>	4	5	4	5	4	5
<b>3</b>	3	5	4	4	5	3
<b>4</b>	5	5	5	5	4	5
<b>5</b>	4	4	5	5	5	5
<b>6</b>	3	5	5	5	5	4
<b>7</b>	4	5	5	5	5	5
<b>8</b>	4	5	5	4	4	5
<b>9</b>	4	5	4	5	4	4
<b>10</b>	4	5	5	5	5	5
<b>średnia</b>	3.9	4.9	4.7	4.8	4.6	4.6
<b>średnia ogólna</b>				<b>4.58</b>		

**Tabela 5.1.** Wyniki testów systemu

- Dokładność pomiarów – użytkownicy zostali poproszeni o wykonanie ćwiczeń typu: stanie na jednej nodze, podskoki, stanie na palcach itp. Na tej podstawie mieli ocenić, czy widok rozkładu nacisku prezentowany w aplikacji jest ich zdaniem zgodny z rzeczywistością. To najistotniejszy test, pokazujący w jakim stopniu problem poruszany na początku pracy udało się rozwiązać. Jak widać w Tabeli 5.1 – test wypadł dobrze.
- Wygląd aplikacji – kolejnym tematem, na który mieli wypowiedzieć się użytkownicy, był widok aplikacji mobilnej – bardzo pozytywnie oceniono tę część systemu. Nawet użytkownicy o mniejszym rozmiarze ekranu nie wspominali o żadnych problemach z interfejsem graficznym użytkownika.
- Działanie aplikacji – tu użytkownicy mieli ocenić samo działanie aplikacji mobilnej – czy nie występują błędy, niepożądane wyłączenia aplikacji itp. Należy wziąć pod uwagę, że wszystkie testy wykonano na smartfonach z systemem Android w wersji 9 i nowszych – żaden z użytkowników z nie miał problemu z uruchomieniem i korzystaniem z aplikacji.
- Chęć dalszego korzystania – większość użytkowników zdecydowałaby się dalej korzystać z urządzenia w celach monitorowania swojego zdrowia. Jedynym zgłaszanym warunkiem było zmniejszenie rozmiaru urządzenia i baterii, by zwiększyła się wygoda użytkowania.

Biorąc pod uwagę wszystkie wspominane wyżej informacje, zaprezentowany system można uznać za w pełni działający i spełniający wszystkie założenia, w tym wymagania funkcjonalne i niefunkcjonalne zaprezentowane w rozdziale 2.

## **6. Podsumowanie**

Celem projektu inżynierskiego była implementacja systemu umożliwiającego przeprowadzenie pomiarów nacisku stóp biegacza na podłożę oraz przedstawienie wyników wspomnianych pomiarów użytkownikowi.

### **6.1. Wnioski**

W ramach prac dokonano analizy rynku, dokładnie zaplanowano kolejne etapy prac, zaprojektowano system, a następnie wytworzono i przetestowano odpowiednie oprogramowanie. Wszystkie szczegółowe cele (opisane w rozdziale 1) oraz wymagania (opisane w rozdziale 2) zostały spełnione, można więc uznać, że projekt zakończono sukcesem.

Inspiracją do tematu pracy była rosnąca dbałość o zdrowie i świadomość przyczyn problemów z aparatem ruchu wśród osób aktywnych. Dodatkowo, w ostatnich latach uwidoczyła się zwiększająca się popularność tzw. „wearable technology”, czyli urządzeń elektronicznych, które można nosić jako elementy odzieży, czy nawet wszechzepiać w ciało użytkownika. Pomimo rosnącej świadomości społeczeństwa, wciąż brakuje technologicznych rozwiązań, które byłyby szeroko dostępne i jednocześnie dawały wysoką jakość pomiarów (taką, na której mogliby bazować zarówno amatorzy, jak i profesjonalni sportowcy). Dla osób planujących karierę sportową (lub będących w jej trakcie), przy ich częstotliwości treningów, już niewielkie nieprawidłowości mogą wiązać się z poważnymi problemami zdrowotnymi.

Tworzenie systemu rozpoczęto od analizy rynku, stworzono prototypową wersję sprzętu, opracowano oprogramowanie mikrokontrolera ze szczególnym zwróceniem uwagi na niski pobór prądu. Zaimplementowano algorytm komunikacji przez Bluetooth oraz opracowano metodę wyświetlania grafiki przedstawiającej rozkład nacisku (w tym dedykowany algorytm uśredniający (4.4)). Zaimplementowano również aplikację mobilną, której działanie przetestowano na rzeczywistym urządzeniu oraz emulatorze dostępnym w programie Android Studio.

Szczególnie ciekawe w pracy było jednoczesne zastosowanie tak wielu technologii. Wytworzenie pełnego systemu wymagało zarówno znajomości podstaw elektroniki oraz umiejętności programowania w języku C, jak również programowania w języku Kotlin i znajomości języka XML. Wykorzystano również narzędzia STM32CubeMX oraz Android Studio, a także kilka

mniej istotnych dla projektu, takich jak program Logic firmy Saleae, czy aplikacja Serial Bluetooth Terminal, ale także programy takie jak Matlab czy Desmos. Z uwagi na użyte języki programowania, była więc konieczna znajomość zarówno obiektowego, jak i strukturalnego paradymatu programowania.

W ramach podsumowania efektów prac nad niniejszym projektem, warto przyjrzeć się Tabeli 6.1. Zawiera ona porównanie istniejących, komercyjnych produktów opisanych w rozdziale 1.4 do systemu zaimplementowanego w ramach niniejszej pracy.

Nazwa produktu	Digitsole [8, 9]	Arion [10]	Niniejszy projekt
Ocena użytkowników	1.9 [9]	brak danych	4.58
Cena	400zł	1200zł	400zł*
Czas pracy	8h	10h	minimum 72h
Śledzenie GPS	tak	tak	tak
Konieczność kalibracji	każdorazowo 10min	jednorazowo 10min	brak
Ocena jakości biegu	tak	tak	nie
Rozmiar zespołu	pomiędzy 10-50 osób	Ok. 25 osób	1 osoba
Ilość sensorów	brak danych	8	16
Typ sensorów	brak danych	akcelerometr, żyroskop i czujniki nacisku	czujniki nacisku
Miejsce umieszczenia elektroniki	w jednej z wkładek	na zewnątrz obu butów	na zewnątrz obu butów
Zbieranie danych bez połączenia z telefonem	tak	nie	nie
Analiza danych pod kątem ryzyka urazu	tak	tak	nie

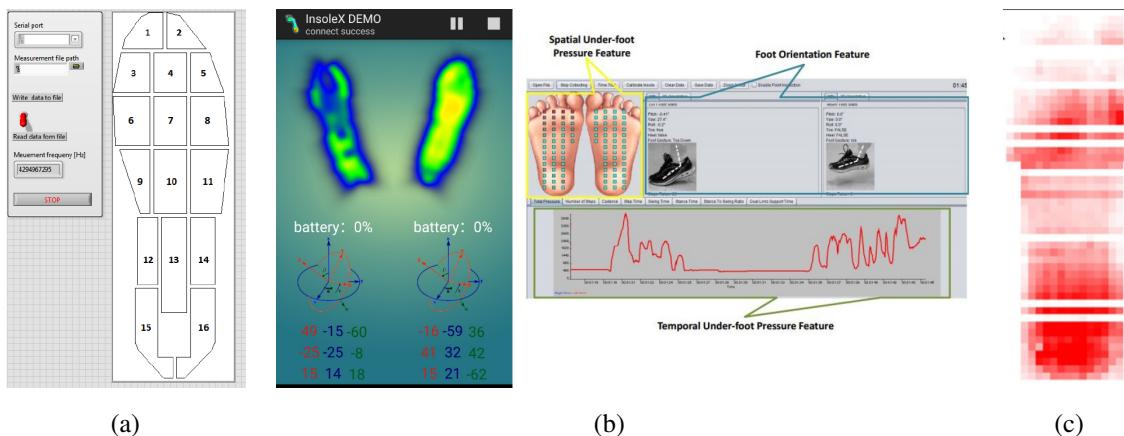
\* – 400zł to koszt sprzętu – dwóch wkładek elektronicznych, płytka rozwojowych NucleoL010RB, dedykowanych płytka, innych potrzebnych komponentów i baterii.

**Tabela 6.1.** Porównanie efektów pracy z rozwiązaniami komercyjnymi

System zaimplementowany w niniejszej pracy ma pewne zalety względem rozwiązań komercyjnych: czas pracy jest znacznie dłuższy, ma więcej czujników nacisku, co oznacza dokładniejsze pomiary, a dodatkowo nie wymaga kalibracji i koszt urządzenia jest dużo niższy. Oczywiście przedstawione rozwiązanie ma również swoje wady – między innymi wymaga stałego połączenia z telefonem przez Bluetooth. W niniejszym projekcie brakuje również funkcjonalności, takich jak analiza ryzyka kontuzji, czy ocena jakości biegu. Pomimo tego, porównując liczbę osób pracujących jednocześnie nad implementacją całego systemu, niniejsza praca wyapada bardziej korzystnie.

Warto zaznaczyć, że poza rozwiązaniami komercyjnymi, istnieją również artykuły i publikacje naukowe o tematyce podobnej do tej, którą omówiono w niniejszej pracy. Na szczególną uwagę zasługują trzy systemy opisane w publikacjach naukowców i badaczy:

- Pierwsza z nich [26] zawiera opis systemu składającego się z mikrokontrolera i wizualizacji danych. Szczególnie ciekawe wydaja się tu budowa wkładki: poza warstwą składającą się z aż 48 czujników nacisku, umieszczono dodatkową warstwę czujników, zawierającą żyroskop, kompas i akcelerometr. Komunikacja odbywa się poprzez Bluetooth Low Energy, a czas pracy wynosi do 24h. Dodatkową zaletą jest fakt, że poza wizualizacją nacisku na urządzeniu mobilnym istnieje możliwość tworzenia wykresów nacisku w specjalnie przygotowanym programie komputerowym. W porównaniu do rozwiązania przedstawionego w niniejszej pracy ma ona mało atrakcyjny interfejs graficzny 6.1c oraz nie umożliwia zebrania uśrednionych pomiarów z całego treningu.
- Drugi artykuł [27] wspomina o bardzo podobnym systemie, jak ten przedstawiony w niniejszej pracy - wkładka składająca się z 16 czujników nacisku połączona z mikrokontrolerem i źródłem energii. Komunikacja odbywa się tu za pomocą modułu radiowego HC-11, który zapewnia ok 40m zasięgu zamiast 10m (jak w przypadku modułu Bluetooth HC-05). Niestety sam sprzęt jest dużo większy niż zaprezentowany w niniejszej pracy. Dodatkową wadą jest fakt, że jedynie oprogramowanie umożliwiające podgląd rozkładu nacisku jest przygotowany jako program komputerowy (Rys. 6.1a). Nie istnieje więc możliwość użycia go w czasie spaceru, ani prześledzenia aktywności całego treningu. Czas zużycia baterii nie jest znany. Należy jednak zwrócić uwagę, że użyty we wspomnianej publikacji mikroprocesor należy do grupy Cortex-M4, która zużywa kilkakrotnie więcej prądu od użytego w projekcie Cortex-M0+.
- Ostatnim artykuł [28] został opublikowany w połowie 2021 roku. Dane pobierane z elektronicznej wkładki są zbierane w pamięci mikrokontrolera, a następnie za pomocą Wi-Fi przesyłane do komputera. Oprogramowanie napisane w języku Java przetwarza dane na zdjęcia, które następnie poddaje się analizie. Niestety wspomniany system nie daje możliwości podglądu rozkładu nacisku w czasie rzeczywistym, nie przygotowano również aplikacji mobilnej, a dane mogą być przekazywane jedynie za pomocą Wi-Fi. Dodatkowo wygenerowane obrazy nacisku nie są zbyt dokładne (Rys. 6.1b), chociaż niewątpliwą zaletą niniejszego systemu jest późniejsza komputerowa analiza wszystkich zebranych obrazów.



Rys. 6.1. Wizualizacje nacisku zaprezentowane we wspomnianych artykułach

## 6.2. Perspektywy i plany rozwoju aplikacji

Warto podkreślić, że przedstawiony projekt nie wyczerpuje w pełni tematyki pracy. System wytworzony w ramach niniejszej pracy inżynierskiej może być dalej rozwijany. Mimo że już teraz widocznych jest wiele perspektyw na dalszy rozwój oprogramowania, należaałoby w pierwszej kolejności porozmawiać z ewentualnym odbiorcą. Istnieje duże prawdopodobieństwo, że sportowcy – zarówno amatorzy, jak i profesjonaliści – chętnie wskazaliby parametry i funkcjonalności, które są dla nich szczególnie istotne i które można byłoby dodać do systemu.

Z uwagi na ogromne braki sprzętu u dostawców elektroniki [29, 30] i ograniczone ramy czasowe projektu, nie udało się zrealizować projektu na oddzielnej, dedykowanej płytce PCB. Wynika z tego problem z rozmiarem sprzętu – do zasilenia płytki NucleoL010RB potrzebne było duże napięcie, co skutkowało wyborem dość dużych baterii. Należy również zaznaczyć, że płytę Nucleo oraz dedykowaną nakładkę można byłoby zastąpić jedną płytą PCB, o dużo mniejszych wymiarach.

Jedyną alternatywą było przeniesienie programu z układu STM32 na ESP32 – są one energooszczędne i mają wbudowany moduł Bluetooth. Niestety utracono by możliwość korzystania z narzędzia STM32CubeMX, a sam program wymagałby całkowitego przeredagowania.

Sama aplikacja mobilna ma duży potencjał rozwoju – można byłoby dopracować zbieranie danych w taki sposób, aby możliwe było prześledzenie kolejnych momentów nacisku w czasie. W ten sposób szczegółowe dane, które zostały zebrane podczas treningu można byłoby w łatwy sposób przekazać trenerowi, lekarzowi czy fizjoterapeutie. Ważnym elementem rozwoju byłoby dodanie zapisywania danych na karcie, w urządzeniu – umożliwiłoby to oddalenie się od telefonu na czas treningu, tym samym zachowując wszystkie zebrane dane. Kolejną funkcjonalnością byłoby umożliwienie rejestracji w aplikacji i przypisywania danych do konta. Dzięki takiemu rozwiązaniu dane nie byłyby tracone w przypadku zmiany telefonu. Można byłoby dopracować aplikację, aby umożliwiała połączenie z innymi popularnymi aplikacjami

mającymi na celu wspieranie zdrowia np. Samsung Health. Warto również wspomnieć o możliwości stworzenia widgetu aplikacji na smartwatch – z całą pewnością ułatwiłoby to korzystanie z systemu w czasie treningu.

W dalszej kolejności rozwoju aplikacji, warto pomyśleć o bardziej zaawansowanych technologicznie rozwiązaniach – np. algorytmach, które w czasie rzeczywistym podawałyby informację tym, że użytkownik biegnie nierówno, za szybko, czy stawia za duże kroki.

Należy dodać, że już teraz zaplanowano dalsze prace nad systemem – oto lista prezentująca, co znajdzie się w kolejnej wersji systemu:

- Wykorzystany sprzęt:
  - zastąpienie wszystkich elementów sprzętowych jedną, niewielką, płytą PCB z wlu-towanym mikrokontrolerem i modułem Bluetooth Low Energy,
  - zmniejszenie źródła zasilania,
  - dodanie karty pamięci.
- Oprogramowanie mikrokontrolera:
  - dodanie oprogramowania umożliwiającego wykorzystanie Bluetooth Low Energy – w celu zaoszczędzenia energii pobieranej z baterii,
  - umożliwienie zapisywania danych na karcie pamięci w przypadku braku dostępnego urządzenia do sparowania poprzez Bluetooth.
- Oprogramowanie aplikacji:
  - możliwość śledzenia nacisku w kolejnych momentach z archiwalnych treningów,
  - integracja z innymi aplikacjami, takimi jak Samsung Health,
  - dodanie opcjonalnej rejestracji i logowania do aplikacji,
  - implementacja dodatkowego oprogramowania na smartwatch.



# Bibliografia

- [1] Ryszard Tadeusiewicz. *Informatyka medyczna*. Lublin: Uniwersytet Marii Curie Skłodowskiej, 2011. ISBN: 9788362773169.
- [2] World Health Organization. *WHO guidelines on physical activity and sedentary behaviour*. Spraw. tech. 2020.
- [3] José Luís Pimentel Rosário. „A review of the utilization of baropodometry in postural assessment”. W: *Journal of Bodywork and Movement Therapies* 18.2 (2014), s. 215–219. ISSN: 1360-8592. DOI: <https://doi.org/10.1016/j.jbmt.2013.05.016>.
- [4] Tiago Baumfeld i in. „Advances of baropodometry in human health”. W: *Ann Musc Disord* 2.2 (2018).
- [5] J. Abramczuk. *Twoja liczba kroków podczas biegu. Sprawdź!* 2017. URL: <https://100hrmax.pl/liczba-krokow-podczas-biegu-sprawdz-sie/>.
- [6] ORTOENTIKA. *Czym jest baropodometria?* [dostęp: 2021-11-20]. Paź. 2020. URL: <https://www.ortodentika.pl/czym-jest-baropodometria/>.
- [7] Nowa ortopedia. *Komputerowe badanie stóp*. [dostęp: 2021-11-20]. URL: <https://nowaortopedia.pl/fizjoterapia-rehabilitacja/komputerowe-badanie-stop/>.
- [8] Firma Digitsole. *Smart insoles run*. [dostęp: 2021-11-20]. URL: <https://digitsole.com/shop/smart-insoles-run/>.
- [9] Amazon.com. *Customer reviews - Digitsole Smart Insole*. [dostęp: 2021-11-20]. URL: <https://www.amazon.com/Digitsole-INTS001BL4547-Profiler-Smart-Insole/product-reviews/B01MRGZ7GY>.
- [10] Arion. *About Arion*. [dostęp: 2021-11-20]. URL: <https://www.arion.run/>.
- [11] Legact. *Specyfikacja produktu - FS-INS-16Z-V1-16*. [dostęp: 2021-12-15]. Dokumentacja w języku Chińskim. URL: <https://files.fm/f/nvrykgcc>.
- [12] ST life.augmented. *Datasheet - STM32L010RB*. [dostęp: 2021-11-20]. 2019. URL: <https://www.st.com/resource/en/datasheet/stm32l010rb.pdf>.

- [13] Atmel Corporation. *Atmega128/L Datasheet Summary*. [dostęp: 2021-11-20]. 2011. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/2467S.pdf>.
- [14] TIOBE Software. *The C programming language*. [dostęp: 2021-11-20]. 2021. URL: <https://www.tiobe.com/tiobe-index/c/>.
- [15] Evertiq. „STMicroelectronics przejmuje Atollic”. W: *Evertiq.pl* (2017). [dostęp: 2021-11-20].
- [16] raywenderlich Tutorial Team i M. Carli. *Dagger by Tutorials (First Edition): Dependency Injection on Android with Dagger & Hilt*. Razeware LLC, 2021. ISBN: 9781950325177.
- [17] Marcin Paczuski. *Cykl życia projektu*. [dostęp: 2021-11-20]. 2016. URL: <https://sii.pl/blog/cykl-zycia-projektu-faza-planowania/>.
- [18] John Lindon. *Encyclopedia of spectroscopy and spectrometry*. Kidlington, Oxford, United Kingdom: Academic Press is an imprint of Elsevier, 2016. ISBN: 978-0-12-803224-4.
- [19] Sangeeta Gulati. „Graphing with Desmos—An online graphing calculator”. W: *At Right Angles 3.2* (2014), s. 61–66.
- [20] Filip Zrnzević. *Aerial photography of forest*. [dostęp: 2021-11-20]. URL: <https://unsplash.com/photos/QsWG0kjPQRY>.
- [21] Filip Zrnzević. *Green leafed trees during daytime photo*. [dostęp: 2021-11-20]. URL: [https://unsplash.com/photos/\\_EMkxLdko9k](https://unsplash.com/photos/_EMkxLdko9k).
- [22] ITead Studio. *HC-05 Bluetooth module datasheet*. [dostęp: 2021-11-20]. 2010. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/2467S.pdf>.
- [23] ST life.augmented. *STM32 Nucleo-64 boards. User manual*. [dostęp: 2021-11-20]. 2019. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/2467S.pdf>.
- [24] Inc Saleae. *Logic Analyzers from Saleae - #1 with Professional Engineers*. [dostęp: 2021-12-10]. 2021. URL: <https://www.saleae.com/>.
- [25] *ASCII Code - The extended ASCII table*. [dostęp: 2021-11-20]. URL: <https://www.ascii-code.com/>.
- [26] Feng Lin i in. „Smart Insole: A Wearable Sensor Device for Unobtrusive Gait Monitoring in Daily Life”. W: *IEEE Transactions on Industrial Informatics* 12 (czer. 2016), s. 1551–3203. DOI: [10.1109/TII.2016.2585643](https://doi.org/10.1109/TII.2016.2585643).
- [27] Michał Ostaszewski, Jolanta Pauk i Kacper Lesiewski. „A portable plantar pressure system: Specifications, design, and preliminary results”. W: *Technology and health care: official journal of the European Society for Engineering and Medicine* 28 (sierp. 2020). DOI: [10.3233/THC-208001](https://doi.org/10.3233/THC-208001).

- [28] Larissa Barbosa Oliveira i in. „Reliability of wireless insole baropodometry of normal individual's gait”. W: *Acta Ortopédica Brasileira* 29 (2021), s. 238–241.
- [29] Oskar Pacelt. *Brak półprzewodników, niedobór elektroniki – kiedy sytuacja wróci do normy?* [dostęp: 2021-11-20]. 2021. URL: <https://botland.com/blog/brak-polprzewodnikow-niedobor-elektroniki-kiedy-sytuacja-wroci-do-normy/>.
- [30] Arkadiusz Strzała. *Kryzys na rynku elektroniki coraz głębszy*. [dostęp: 2021-11-20]. 2021. URL: <https://www.gry-online.pl/hardware/kryzys-na-rynk-elektroniki-coraz-glebszy-braki-pradu-i-lockdowny/z220862>.



## **Załączniki**

1. Internetowe repozytorium umieszczone w serwisie Github ([https://github.com/KlaudiaKromolowska/System\\_monitorowania\\_nacisku\\_stop](https://github.com/KlaudiaKromolowska/System_monitorowania_nacisku_stop)), zawierające następujące katalogi i pliki:
  - 1.1. Katalog „Mikrokontroler” – zawierający oprogramowanie mikrokontrolera oraz plik z konfiguracją programu STM32CubeMX;
  - 1.2. Katalog „Android” – zawierający pliki z kodem aplikacji mobilnej;
  - 1.3. Katalog „MATLAB” – zawierający plik konfiguracji z narzędzia Curve Fitting Tool;
  - 1.4. Plik „Praca dyplomowa” – zawierający niniejszą pracę w formacie pdf.