

TEAM 15

Bramka do bezkontaktowego pomiaru temperatury ciała

Autorzy:

Anh Quan Do 325 268
Michał Jęczmieniowski 325 282
Grzegorz Papaj 325 311
Filip Wojda 325 336
Agata Zatorska 325 338
Mateusz Bednarczyk 318 753

Opiekun projektu: dr inż. Konrad Godziszewski

Spis treści

1	Zakres i cel projektu	2
2	Organizacja pracy	2
2.1	Product Backlog	2
2.2	Sprint Backlog	3
2.3	Przebieg spotkań	5
3	Kamień milowy	5
4	Wykres Gantta	6
4.1	Założenia	6
4.2	Faktyczny stan	6
5	Sposób realizacji	6
5.1	Raspberry Pi 4B	7
5.2	MLX 90614	7
6	Co udało się osiągnąć	7
6.1	Korekcja błędów pomiarowych	8
6.2	Baza danych	8
6.3	Konwersja imienia i nazwiska SpeechToText	9
6.4	Aplikacja do kontaktu z bazą danych	9
6.4.1	Backend	9
6.4.2	Frontend	10
6.5	Zapis pomiarów do bazy danych	11
6.6	Zapis pomiarów do bazy danych	12
6.7	Wyświetlanie wyników i komunikacja z użytkownikiem	13
6.8	Konstrukcja bramki	14
7	Lista „commitów”	15
8	Repozytorium	17
9	Pierwsza prezentacja	17

1 Zakres i cel projektu

Jako nasz zespół pod nazwą **Team 15** wybraliśmy temat projektu, którego zadaniem było stworzenie bramki do bezkontaktowego pomiaru temperatury ciała. Wybór padł na ten temat, ponieważ wydał nam się on interesujący, zwłaszcza na fakt dynamicznego rozwoju technologii tego typu, który wynikał z dużego zapotrzebowania takich urządzeń podczas pandemii, w której kluczową kwestią był pomiar temperatury z jak największym ograniczeniem kontaktu z innymi ludźmi czy rzeczami.

Projekt pozwolił nam zapoznać się z interfejsami bezprzewodowymi, implementowaniem do układów mikrokontrolerów oraz nauką programowania oraz obsługi czujników.

Najważniejszym zadaniem, był oczywiście fakt, aby urządzenie działało poprawnie i mierzyło temperaturę faktyczną, zgodną z prawdą. Oprócz tego zrealizowaliśmy dodatkowe funkcjonalności, które pomogły w tym, aby bramka była prosta i intuicyjna w obsłudze, przy jednoczesnym zachowaniu ważnych i potrzebnych z perspektywy klienta funkcji na przykład takich jak zapis mierzonej temperatury, czy wyświetlacz, ukazujący obecną temperaturę.



2 Organizacja pracy

Prace nad projektem prowadziliśmy z wykorzystaniem metodyki Scrum. Zgodnie z jej założeniami każdy z członków zespołu otrzymał swoją rolę. Właścicielem produktu został Michał Jęczmieniowski, Scrum Masterem Agata Zatorska, a developerami Filip Wojda, Grzegorz Papaj oraz Anh Quan Do. Nasze Backlog'i oraz „to do” listy znajdowały się w aplikacji Notion, tak aby wszyscy członkowie zespołu mieli do nich łatwy dostęp.

2.1 Product Backlog

Przygotowania do realizacji projektu zaczęliśmy od stworzenia Produkt Backlog'u widocznego na rysunku 1. Zawarliśmy w nim wszystko co uważamy za niezbędne w naszym finalnym produkcie, jednak w trakcie projektu aktualizowaliśmy go w miarę potrzeb. Każdy element miał swój priorytet oraz status wykonania. Zadbaliśmy również, aby do każdego z zadań przypisana była adekwatna historyjka prezentująca wartość danej funkcji dla klienta.

Historyjka	Zadanie	Status	Priorytet	Sprint
—	Research dotyczący tematu projektu	Wykonane	Wysoki	Sprint 1
Jako personel chcę, aby oprogramowanie bramki pozwalało na fizyczną implementację.	Stworzenie oprogramowania obsługującego czujnik temperatury.	Wykonane	Wysoki	Sprint 1
Jako użytkownik chcę, aby bramka była w stanie dokładnie mierzyć temperaturę ciała, aby zapewnić precyzyjne wyniki.	Stworzenie podstawowej infrastruktury pomiarowej i implementacja czujników temperaturowych.	Wykonane	Wysoki	Sprint 1
Jako użytkownik chcę, aby bramka informowała mnie o byłej wysokiej lub zbyt niskiej temperaturze.	Zaimplementowanie sygnałów, oznajmiających, że zmierzona temperatura odbiega od przyjętej normy.	Wykonane	Średni	Sprint 1
Jako użytkownik chcę mieć możliwość odczytania pomiaru temperatury.	Zaimplementowanie funkcji odczytywania zebranych danych.	Wykonane	Średni	Sprint 2
Jako użytkownik chcę, aby bramka była jak najbardziej precyzyjna i odczyt temperatury uwzględniał wszelkie błędy pomiarowe	Korekta błędów pomiarowych OPEN	Wykonane	Wysoki	Sprint 2
Jako użytkownik chcę, aby bramka była intuicyjna w obsłudze, aby mogła być używana przez każdego bez problemów.	Stworzenie czytelnego interfejsu użytkownika.	Wykonane	Średni	Sprint 2
Jako personel medyczny chcę mieć historię pomiarów w celu kontroli stanu zdrowia pacjenta	Stworzenie bazy danych, zapisującej wyniki pomiarów.	Wykonane	Mały	Sprint 3
Jako użytkownik chcę, aby bramka była pełni bezdotykowa	Stworzenie możliwości podania swoich danych drogą głosową	Wykonane	Mały	Sprint 2
Jako użytkownik, chcę, aby bramka była zaprojektowana w sposób estetyczny.	Zaprojektowanie estetycznego wyglądu bramki	Wykonane	Wysoki	Sprint 3
Jako użytkownik chcę mieć pewność, że bramka działa bez zarzutu i została sprawdzona pod każdym kątem	Przeprowadzenie testów funkcjonalnych	Wykonane	Wysoki	Sprint 3

Rysunek 1: Product Backlog

2.2 Sprint Backlog

W pierwszym semestrze udało nam się przeprowadzić jeden Sprint trwający od 11 do 30 grudnia 2023 roku. Przed rozpoczęciem spotkaliśmy się na jego planowanie, podczas którego stworzyliśmy Sprint Backlog (rysunek 2). Znajdują się w nim elementy z Product Backlog'u, które chcieliśmy zrealizować w trakcie pierwszego sprintu wraz z krótkim opisem w jaki sposób planowaliśmy tego dokonać. Założyliśmy sobie 4 cele: research, stworzenie oprogramowania do obsługi czujnika temperatury, zaimplementowanie sygnałów oznajmiających o zbyt wysokiej lub zbyt niskiej temperaturze oraz stworzenie podstawowej infrastruktury pomiarowej. Wszystkie z nich udało nam się wykonać.

The screenshot shows a digital board for 'Sprint Backlog'. At the top, it says 'Obecny Sprint: Sprint 1' and 'Sprint 1'. Below is a timeline from '11.12.2023 - 30.12.2023'. The main section is titled 'Sprint 1' and contains a table of tasks:

Zadanie	Daty	Opis	Priorytet	Status	#
Research dotyczący tematu projektu	December 11, 2023 → December 13, 2023	Znalezienie źródeł i materiałów, które pomogą w zrealizowaniu projektu.	Wysoki	Wykonane	1
Stworzenie oprogramowania obsługującego czujnik temperatury.	December 14, 2023 → December 22, 2023	Development i testy jednostkowe oprogramowania obsługującego czujnik temperatury.	Wysoki	Wykonane	2
Zaimplementowanie sygnałów, oznajmujących, że zmierzona temperatura odbiega od przyjętej normy.	December 22, 2023 → December 23, 2023	Sygnały powinny informować o zbyt wysokiej lub niskiej temperaturze ciała, błędach w pomiarach spowodowanych np. zbyt szybkim przejściem przez bramkę.	Średni	Wykonane	4
Stworzenie podstawowej infrastruktury pomiarowej i implementacja czujników temperaturowych.	December 27, 2023 → December 30, 2023	Implementacja oprogramowania na fizycznym mikroprocesorze	Wysoki	Wykonane	3

+ New

Rysunek 2: Sprint Backlog

W semestrze 2 zaplanowaliśmy dwa kolejne sprints, które zostały zaplanowane jak na rysunku 3 i 4.

The screenshot shows a digital board for 'Sprint 2'. The main section is titled 'Sprint 2' and contains a table of tasks:

Zadanie	Daty	Opis	Priorytet	Status
Stworzenie możliwości podania swoich danych drogą głosową	February 19, 2024 → February 23, 2024	Dodanie do kodu funkcji umożliwiających rozpoznanie mowy użytkownika, w celu podania jego danych osobowych	Mały	W trakcie
Korekta błędów pomiarowych	February 22, 2024 → February 27, 2024	Uwzględnienie w kodzie błędów pomiarowych (współczynnik emisjności, podawane napięcie inne niż w manualu)	Wysoki	W trakcie
Zaimplementowanie funkcji odczytywania zebranych danych.	February 28, 2024 → March 4, 2024	Dodanie możliwości wyświetlenia wyniku przeprowadzonego badania	Średni	Niezaczęte
Stworzenie czytelnego interfejsu użytkownika.	March 1, 2024 → March 11, 2024	Zaprojektowanie czytelnego interfejsu użytkownika (prosty i klarowny przekaz dla użytkownika)	Średni	Niezaczęte

Rysunek 3: Sprint Backlog

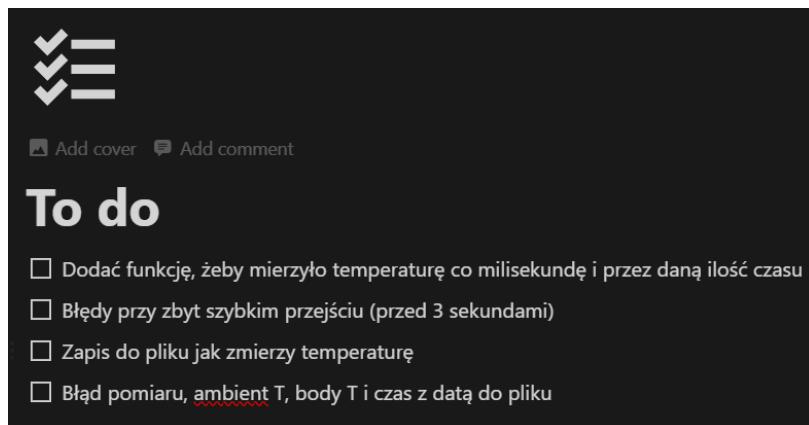
Sprint 3				
Zadanie	Daty	Opis	Priorytet	Status
Stworzenie bazy danych, zapisującej wyniki pomiarów.	March 18, 2024 → March 26, 2024	Stworzenie bazy danych przechowującej imię, nazwisko, wynik pomiaru oraz datę jego wykonania	Mały	Wykonane
Zaprojektowanie estetycznego wyglądu bramki	March 27, 2024 → April 5, 2024	Zaprojektowanie fizycznej bramki	Wysoki	Wykonane
Przeprowadzenie testów funkcjonalnych	April 1, 2024 → April 8, 2024	Przeprowadzenie testów w celu sprawdzenia funkcjonalności bramki	Wysoki	Wykonane

+ New Calculate COMPLETE 100%

Rysunek 4: Sprint Backlog

2.3 Przebieg spotkań

Po wykonaniu researchu, ze względów logistycznych, wszystkie spotkania przeprowadzaliśmy na żywo. Przed każdym z nich tworzyliśmy wstępna listę rzeczy, które chcemy na nim zrealizować i staraliśmy się trzymać założonego planu. Przykład takiej listy jest widoczny na rysunku 3. W pierwszej części projektu wszyscy pracowaliśmy razem, ponieważ komponenty, z których korzystaliśmy były dla nas nowością i chcieliśmy je dobrze zrozumieć, jednak przy następnych sprintach bardziej rozdzieliłyśmy pracę, tak aby każdy mógł skupić się nad swoją częścią. Pozwoliło to znaczowo zwiększyć efektywność. Na przykład kiedy jedna osoba pracowała nad backend'em aplikacji webowej inna pracowała już nad CSS.



Rysunek 5: To do list

3 Kamień milowy

Definicja naszego kamienia milowego zakładała sfinalizowanie następujących funkcjonalności do dnia 1 kwietnia 2024r.:

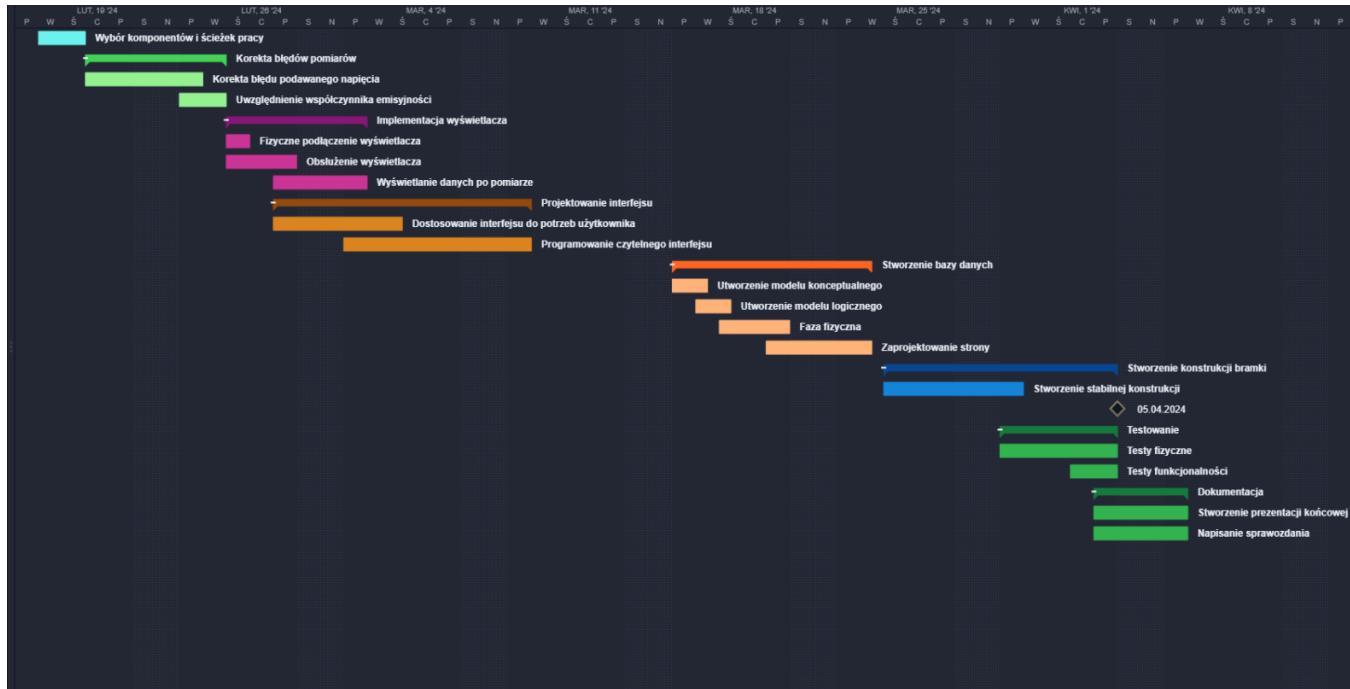
- Pomiar temperatury z wykorzystaniem czujnika MLX90614 z uwzględnieniem błędów pomiarowych
- Możliwość podania imienia i nazwiska drogą werbalną
- Zapis pomiarów do bazy danych wraz z datą pomiaru oraz imieniem i nazwiskiem
- Aplikacja do edytowania, usuwania oraz ręcznego dodawania rekordów do bazy danych

Udało nam się zrealizować wszystkie funkcje, które zaplanowaliśmy na kamień milowy do wyznaczonego deadline'u.

4 Wykres Gantta

4.1 Założenia

W ramach przygotowania do realizacji drugiej części projektu, stworzyliśmy wykres Gantta, ilustrujący planowany zgrubnie przebieg prac w ramach przedmiotu PROJ2. Zakładana realizacja projektu ma zawsze się w dwóch trzytygodniowych Sprintach i jest podzielona na osiem głównych obszarów prac. Jako datę kamienia milowego przyjęliśmy koniec drugiego sprintu tj. 5 kwietnia 2024r., kiedy to podejmiemy próbę scalenia fizycznych elementów konstrukcji bramki wraz z przygotowanym wcześniej oprogramowaniem.



Rysunek 6: Wykres Gantta

4.2 Faktyczny stan

Stworzony w przygotowaniach do drugiej części projektu wykres, posłużył nam jako organizator czasu pracy nad poszczególnymi zadaniami. Realizując założenia trzymaliśmy się dat określonych na wykresie z tolerancją kilku dni, ponieważ niektóre zadania zajęły nam mniej czasu niż zakładaliśmy, a niektóre okazały się być bardziej pracochłonne. Zmiany w organizacji pracy obejmowały przesunięcie prac nad obsłużeniem wyświetlacza, którego finalna implementacja nie powiodła się z powodu trudności technicznych omawianych z opiekunem projektu. Faktyczna realizacja została więc powiększona o trzeci sprint, który odbył się w okresie przed prezentacją końcową, a główne prace wykonane w ramach niego obejmowały próby implementacji wyświetlacza oraz jego naprawy po pierwotnym spaleniu poprzez niepoprawne podłączenie pinów. Ze względu na rozlutowanie części przewodów nie udało nam się połączyć wyświetlacza, więc skupiliśmy się na prezentacji instrukcji oraz danych w terminalu. Pozostałe założenia zawarte na wykresie Gantta zostały pomyślnie wykonane, a efekt prac zaprezentowany został podczas głównej prezentacji w audytorium centralnym.

5 Sposób realizacji

Do zrealizowania pierwszej części projektu zdecydowaliśmy się na wykorzystanie jednopłytkowego mikrokomputera **Raspberry Pi 4B** jako kontrolera oraz bezkontaktowego termometru **MLX 90614**, który został nam przydzielony przez opiekuna projektu.

5.1 Raspberry Pi 4B

Raspberry Pi był dla nas oczywistym wyborem z wielu względów, ale dwa z nich można wyróżnić.

Od samego początku założyliśmy, że zbudowanie kodu językiem niskiego poziomu (programowania) będzie dla nas niewydajne, zatem wspólnie zdecydowaliśmy, że część logiczna zostanie zrealizowana w języku wysokiego poziomu - **Pythonie**. Wybór ten poparliśmy tym, że mamy w tym języku programowania największe doświadczenie oraz ma on niezwykle szeroki wachlarz bibliotek, z których można skorzystać. Raspberry Pi miał już wbudowany edytor tekstowy do Pythona - **Thonny**.

Drugim najważniejszym powodem, dla którego wybraliśmy Raspberry Pi był jego system operacyjny - **Linux**, a dokładniej **Raspberry Pi OS** bazowany na **Debianie**. Ten system operacyjny również jest dla nas najbardziej znajomy i ma wiele zalet, w tym łatwe użytkowanie z Gita oraz połącznia SSH.

Oprócz tych dwóch argumentów, Raspberry Pi wydawał się też atrakcyjna propozycja ze względu na jego wiele portów. USB-C wykorzystane było do zasilenia minikomputera przy 15 Watach, 2 porty USB-A z 4 (dwa USB 3 i dwa USB 2) wykorzystane były przez odbiorniki bezprzewodowej myszki i klawiatury, a mini-HDMI dawało nam możliwość wykorzystania dowolnego monitora z wejściem HDMI. Dodatkowo Raspberry Pi posiada aż 40 pinów GPIO (General Purpose Input/Output). W naszym projekcie w pierwszej części wystarczyły 4 z nich, ale może się to zmienić wraz z rozszerzaniem naszej wizji końcowego produktu poprzez dołączanie dodatkowych części, np. ekranu LED. Obecność WIFI i bluetooth również ułatwia nam pracę, ponieważ mogliśmy przesyłać obraz pulpitu prosto do laptopa oraz podłączyć bezprzewodowe urządzenia. Jego mały rozmiar będzie też bardzo użyteczny przy montowaniu go na finalnej fizycznej bramce.

Raspberry Pi 4B jest bardzo popularny przy tego typu projektach i jest szeroko dostępny przy przystępnej cenie. Minikomputer został nam dostarczony przez opiekuna projektu.

5.2 MLX 90614

MLX 90614 to bezkontaktowy termometr oparty na technologii podczerwieni. Mierzy on promieniowanie cieplne emitowane przez badany obiekt oraz otoczenie i przekształca je w odczyt temperatury w Celsjuszach. Następnie temperatura ta jest przekazywana przez interfejs komunikacyjny, takie jak **I2C** (inter-integrated circuit) lub SMBUs. Nasz zespół zdecydował się na użycie tego pierwszego, co wymagało ściągnięcia odpowiednik bilibotek w pythonie.

Termometr został podłączony do pinów na Raspberry Pi czterema przewodami:

- **Przewód czerwony** - odpowiedzialny za zasilanie i idzie do pina **numer 1**. Ponieważ nasz termometr jest wersja działająca przy około 3V, przewód podpinamy do pina który dostarcza 3.3V.
- **Przewód żółty** - jest linią komunikacyjną SDA (Serial Data) w protokole i2c i jest podłączany do **pina numer 3**.
- **Przewód niebieski** - jest linią komunikacyjną SCL (Serial Clock) w protokole i2c i jest podłączany do **pina numer 5**.
- **Przewód czarny** - ten przewód podłączany jest do **pina numer 6** który odpowiada za uziemienie.

Termometr MLX 90614, tak jak Raspberry Pi, również jest łatwo dostępny i cechuje się bardzo niską ceną.

6 Co udało się osiągnąć

Raspberry Pi obsługuje czujnik temperatury MLX za pomocą programu, który napisaliśmy w Pythonie. W jego pisaniu posiliłowaliśmy się bibliotekami do obsługi magistrali I2C i czujnika oraz bibliotek do pracy z czasem.

Działa on w nieskończonej pętli działającej co sekundę i rozpoczyna się wykonaniem pomiarów obiektu, którym w naszym przypadku jest pacjent, i otoczenia, właśnie za pomocą tych bibliotek.

Kolejnym krokiem jest ustalenie, czy mierzona jest temperatura pacjenta, który znajduje się przed czujnikiem. Dzieje się to za pomocą sprawdzenia czy moduł różnicy temperatur otoczenia i pacjenta jest większy od 5 stopni Celsjusza, ponieważ jeśli sensor nie wykryje żadnego obiektu to traktuje temperaturę otoczenia jako temperaturę pacjenta. Na tej podstawie możemy określić czy pacjent stoi przed czujnikiem, czy też nie. Poprzez te 5 stopni zabezpieczamy sobie odpowiednio dużą granicę błędu, tak aby nie wykonywać pomiaru niepotrzebnie. Gdy obiekt zostanie wykryty, rozpoczyna się pomiar poprzez zmianę wartości flagi oraz wyświetlenie stosownej informacji w konsoli.

Pomiar trwa 3 sekundy. Polega on na 150-krotnym zmierzeniu temperatury pacjenta oraz, przy ostatniej iteracji, wyciągnięciu średniej. Jeżeli w dowolnym momencie pacjent opuści pole widzenia czujnika, natychmiast zostanie o tym poinformowany a pomiar zostanie przerwany. Lista zawierająca dotychczasowe wartości temperatur zostaje wtedy wyczyszczona, co skutkuje koniecznością ponownego rozpoczęcia pomiaru.

Po udanym pomiarze, uzyskaną średnią temperaturę traktujemy jako faktyczną temperaturę ciała pacjenta i porównujemy ją z ustalonymi przez nas wartościami wzorcowymi. Jeśli temperatura pacjenta jest mniejsza niż 34 stopnie Celsjusza stwierdzamy, że jest ona poniżej normy. Jeśli jest w przedziale 38, 39,5 stwierdzamy gorączkę. Powyżej 39,5 jest to stan krytyczny, a pozostały przedział, tj. 34, 38 traktujemy jako temperaturę w normie. Oczywiście w żaden sposób nie jesteśmy przywiązani do tych wartości temperatur i możemy je w każdej chwili zmienić.

Ostatnim krokiem jest zapisanie zebranych danych, obecnie do pliku csv. Dane zapisujemy w formacie: temperatura pacjenta, temperatura otoczenia, (obie w zaokrągleniu do 2 miejsc po przecinku) czas wykonania pomiaru, dokładność pomiaru dla danego przedziału temperatur, imię, nazwisko.

6.1 Korekcja błędów pomiarowych

Dokładność opisana jest za pomocą funkcji, przyjmującej temperaturę otoczenia i pacjenta jako parametry. Jeśli temperatura otoczenia zawiera się w przedziale od 0 do 50, a temperatura pacjenta w przedziale od 0 do 60, funkcja zwraca wartość 0.5. W przeciwnym razie jest to 1. Te wartości są odczytane z karty katalogowej czujnika. Następnie cały cykl powtarzany jest od początku.

Niedokładności w pomiarze temperatury czujnikiem bezprzewodowym mogą również z faktu na budowę i strukturę różnych obiektów i ich zdolność do oddawania promieniowania podczerwonego do otoczenia. Tę zależność opisuje tak zwany współczynnik emisyjności ciała ϵ . Im wyższy współczynnik emisyjności (maksymalna wartość to 1) tym ciało potrafi wyemitować więcej energii. W naszym przypadku obiekt mierzony, czyli ciało ludzkie, ϵ ma wartość 0.972. Aby otrzymać faktyczną temperaturę, nie taką jaką otrzymaliśmy podczas pomiaru, a dokładną wartość temperatury ciała, musimy podany ϵ oraz uzyskane podczas pomiaru wartość temperatury obiektu jak i otoczenia, podstawić pod ogólny wzór:

$$T_{\text{obiektu}} = \sqrt[4]{\frac{T_{\text{pomiaru}}^4 - (1-\epsilon)*T_{\text{otoczenia}}^4}{\epsilon}}.$$

Przy przykładowym pomiarze temperatury otoczenia 25 i temperatury pomiarowej 36 stopnia Celsjusza, po korekci wynik to 36.19 stopni Celsjusza.

Ostatnim z aspektów, który musielibyśmy zmienić, aby nasz wynik stanowił faktycznie dobry wynik, był korekta błędu związanego z innym woltagiem napięcia. Mianowicie nasz system zasilamy napięciem 3.3 V, gdzie optymalnym dla czujnika jest napięcie 3 V. Z tej różnicą mogą wynikać nieprawidłowości, które jednak zniwelowaliśmy poprzez zastosowanie się do noty katalogowej naszego czujnika i wykorzystania wzoru: $T_{\text{compensated}} = T_0 - (VDD-3)*0.6$.

6.2 Baza danych

W projekcie postanowiliśmy zaimplementować bazę danych w celu zwiększenia efektywności zarządzania danymi pomiarowymi.

Ze względu na posiadane już przez nas doświadczenie z **Oracle** zdecydowaliśmy się na wybór tej właśnie bazy danych. Składa się ona z jednej relacji przechowującej rekordy danych pomiarowych. Relacja ma 8 atrybutów: number pomiaru, temperatura ciała, temperatura otoczenia, imię, nazwisko, data pomiaru, błąd pomiaru oraz wartość boolean mówiącą o tym, czy pomiar został wprowadzony przez personel do bazy danych ręcznie.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 NR_POMIARU	NUMBER (38, 0)	No	(null)	1 (null)	
2 TEMPERATURA_CIALA	NUMBER (38, 2)	No	(null)	2 (null)	
3 TEMPERATURA_OTOCZENIA	NUMBER (38, 2)	Yes	(null)	3 (null)	
4 IMIE	VARCHAR2(50 BYTE)	No	(null)	4 (null)	
5 NAZWISKO	VARCHAR2(50 BYTE)	No	(null)	5 (null)	
6 DATA_POMIARU	TIMESTAMP (6)	No	(null)	6 (null)	
7 BLAD_POMIARU	NUMBER (38, 2)	Yes	(null)	7 (null)	
8 CZY_DODANE_RECZNIE	NUMBER (1, 0)	Yes	(null)	8 (null)	

Rysunek 7: Tabela Pomiary

6.3 Konwersja imienia i nazwiska SpeechToText

Ponieważ w projekcie postanowiliśmy uwzględnić przechowywanie wykonanych pomiarów we wcześniej stworzonej bazie danych Oracle, należało dodać możliwość podania, przez pacjenta swojego imienia i nazwiska. Głównym założeniem bramki jest, że jest bezkontaktowa, więc postawiliśmy na przekazywanie danych drogą verbalną. W metodach, o które rozbudowaliśmy kod korzystaliśmy z biblioteki `SpeechRecognition`, która wspiera wiele silników rozpoznawania mowy. My użyliśmy silnika Google.

Po zmierzeniu temperatury na terminalu (docelowo ekranie LCD) wyświetla się prośba o podanie imienia i nazwiska za co odpowiada funkcja `speech_to_text()`. Użytkownik ma na to 8 sekund. Istnieje możliwość, że użytkownik się pomylił lub program źle zrozumiał jego mowę, więc na terminalu wyświetla się zapytanie czy mowa została poprawnie przekonwertowana. Pacjent ma dwie próby, jeżeli po tym imię i nazwisko nadal nie będzie poprawnie zrozumiane, to pojawia się prośba o przeliterowanie imienia oraz nazwiska. Następnie program ponownie pyta czy to co zrozumiał to dane użytkownika, jeżeli tak, dane zostają zapisane do pliku CSV wraz z pomiarem, jeżeli nie, wyświetla się prośba o zwrócenie się do personelu, który przy użyciu aplikacji opisanej w sekcji "Aplikacja do kontaktu z bazą danych" wprowadzi personalia do bazy danych ręcznie.

Bramka nie musi znajdować się w pomieszczeniu, w którym pasuje bezwzględna cisza, ponieważ zauważaliśmy w kodzie funkcję `adjust_for_ambient_noise()` która po uruchomieniu mikrofonu dostosowuje poziom szumów otoczenia przez 6 sekund. Podczas tego procesu użytkownik jest proszony o poczekanie.

6.4 Aplikacja do kontaktu z bazą danych

6.4.1 Backend

Ważnym dla nas aspektem projektu jest prostota użytkowania bramki, zarówno przez badanych pacjentów jak i personel, który powinien w dowolnym momencie mieć możliwość wglądu do danych pomiarowych. Do realizacji tego celu wykorzystaliśmy popularny framework w języku **Java - Spring Boot**, w którym stworzyliśmy aplikację webową do komunikacji z naszą bazą danych.

Sama aplikacja prezentuje użytkownikowi tylko wybrane kolumny z relacją. Są to: temperatura pomiaru, imię, nazwisko oraz data pomiaru. Uprawniony użytkownik może również zalogować się jako administrator - wtedy aplikacja prezentuje wszystkie kolumny relacji bez konieczności ich manualnego sprawdzania w bazie danych. Jest to wygodnie rozwiązywanie w sytuacji np. ewentualnego błędu w systemie, gdzie dostęp do całości danych często jest wymagany do efektywnego zlokalizowania usterki oraz postawienia poprawnej diagnozy. Dodatkowo, użytkownik ma możliwość ręcznego wprowadzania pomiarów oraz usuwania rekordów z bazy danych za pomocą aplikacji.

Oprócz odczytywania danych pomiarowych z bazy danych oraz wyświetlania ich użytkownikowi personelu nasza aplikacja pełni jeszcze jedną równie istotną funkcję. Służy ona do automatycznego zapisu rekordów do bazy danych. W tym celu napisaliśmy prostą metodę odczytującą dane z plików pomiarowych pochodzących z Raspberry Pi, którą oznaczyliśmy adnotacją `@Scheduled` z wartością `fixedRate` ustawioną na **4000**. Oznacza to, że metoda co 4 sekundy będzie odczytywać oraz, za pomocą obiektu DAO, zapisywać dane do bazy danych.

6.4.2 Frontend

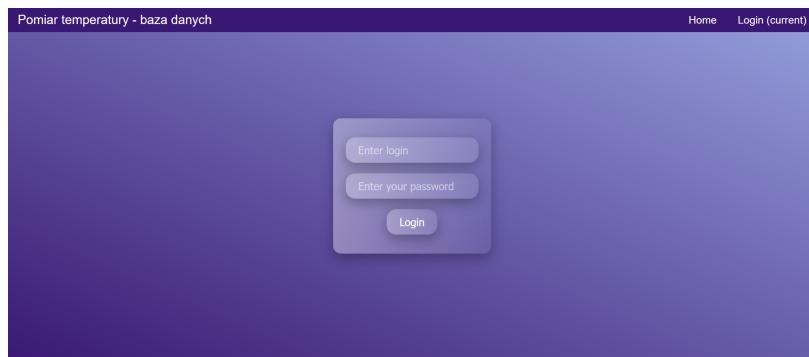
W części prezentowanej użytkownikowi skupiliśmy się przede wszystkim na czytelności danych oraz przejrzystości strony. Do przygotowania stron użyliśmy HTML, a w celu zapewnienia estetycznej prezentacji pozytkowaliśmy się CSS, dzięki któremu mogliśmy zdefiniować kolory przewodnie strony, wymiary poszczególnych elementów oraz akcje wykonywane na nich przy przemieszczaniu się użytkownika. Tak właśnie udało nam się osiągnąć między innymi "efekt skupienia" po zaznaczeniu danego pola lub efektywnie podświetlenie przycisku **Login** towarzyszące najechaniu na niego. Ponadto zdecydowaliśmy się na umieszczenie w górnej części strony paska nawigacyjnego z przyciskami umożliwiającymi akcje w obrębie aplikacji webowej.

Użytkownik witany jest stroną powitalną z informacją o możliwości zalogowania się poprzez kliknięcie na wyświetlany komunikat, bądź też na przycisk na pasku nawigacyjnym. Na wspomnianym pasku znajduje się również tytuł strony oraz hiperlink przenoszący do aktualnej strony powitalnej.



Rysunek 8: Widok Welcome Page

Po wybraniu opcji zalogowania się, użytkownik zostaje przekierowany na odpowiednią stronę, która umożliwia wpisanie loginu oraz hasła, natomiast kliknięcie na przycisk **Login** pozwala na potwierdzenie wprowadzonych danych.



Rysunek 9: Widok Login Page

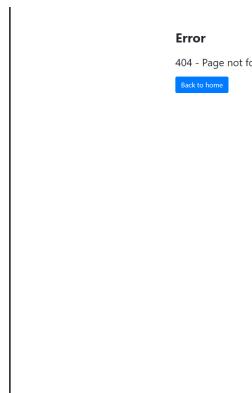
Po zalogowaniu się do strony, w zależności od tego, czy zalogowany jest admin, czy użytkownik, wyświetla się tabelka z danymi według wzoru stylistycznego zaprezentowanego poniżej.

ID	Temperatura Ciała	Temperatura Otoloczenia	Imię	Nazwisko	Data Pomiaru	Błąd Pomiaru	Action
13830.72	23.81	Piotr	koscisz		2024-06-12T17:55:15.5		Edytuj Usuń
13533.68	23.17	Wioletta	Zochowska		2024-06-12T17:47:09.5		Edytuj Usuń
13634.01	23.35	Blażej	Borowy		2024-06-12T17:48:27.5		Edytuj Usuń
13732.33	23.63	Patryk	Kalbarczyk		2024-06-12T17:49:59.5		Edytuj Usuń
13933.55	22.71	bajo	jajo		2024-06-12T18:01:14.5		Edytuj Usuń
14032.22	20.61	Oliwia	Borkowska		2024-06-12T18:15:48.5		Edytuj Usuń
14133.48	20.97	pj	do		2024-06-12T18:17:36.5		Edytuj Usuń
14234.0	21.15	Agata	Zatorska		2024-06-12T18:18:26.5		Edytuj Usuń
14331.94	20.59	Michał	Krzyżanowski		2024-06-12T18:28:10.5		Edytuj Usuń
14551.6	21.45	Michał	sokala		2024-06-12T18:35:34.0		Edytuj Usuń
14633.2	21.03	Zenon	damon		2024-06-12T18:39:41.0		Edytuj Usuń
12336.65	23.57	Filip	Wajda		2024-06-12T18:54:48:18.0		Edytuj Usuń
13235.03	22.31	Piotr	kosiara		2024-06-12T18:40:42.0		Edytuj Usuń
13335.24	22.53	Natalia	łopczewska		2024-06-12T18:42:48.0		Edytuj Usuń
13433.36	21.81	Piotr	chudzikiewicz		2024-06-12T18:58:09.5		Edytuj Usuń

Rysunek 10: Widok strony z danymi

W odpowiednich miejscach tabelki znajdują się przyciski akcji służące do edycji lub usunięcia danego rekordu, a dodatkowo podczas korzystania ze strony po zalogowaniu się, na pasku nawigacyjnym znajduje się również przycisk **Logout**, umożliwiający wygodne wylogowanie się ze strony.

W przypadku odczytania przez aplikację kodu statusu błędu HTTP, użytkownik przekierowany jest na stronę z odpowiednim komunikatem, gdzie znajduje się również przycisk, umożliwiający powrót do strony głównej. Rozpatrzzone błędy to: 403 - forbidden page; 404 - page not found; 500 - internal server error; 504 - gateway timeout. Przykładowy widok został przedstawiony poniżej.



Rysunek 11: Widok strony z komunikatem o błędzie

6.5 Zapis pomiarów do bazy danych

Dane pomiarowe zapisywane na minikomputerze Raspberry Pi zapisywane są ciągiem w jednym pliku CSV. Aby zapisać dane z pomiarów do bazy danych, należało rozbić zadanie na dwie części. Pierwszą z nich było wysłanie danych z Raspberry Pi do zewnętrznego serwera, którym w naszym projekcie będzie laptop podłączony do tej samej sieci co Raspberry Pi. Umożliwia nam to połączenie obu maszyn poprzez protokół **SSH** (Secure Shell). Dzięki temu, jesteśmy w stanie skorzystać z protokołu sieciowego **SCP** (Secure Copy Protocol), który służy przesyłowi danych pomiędzy dwoma urządzeniami połączonymi przez SSH. SCP działa jako komenda w terminalu i można skorzystać z niej na dwa sposoby - albo przesyłać pliki z urządzenia, na którym wywoływana jest komenda, do drugiego urządzenia; albo pobierać pliki z innego urządzenia na to, na którym wywołano komendę. W pierwszym podejściu nasz zespół spróbował napisać kod w języku python, który działałby na Raspberry Pi i wysyłał pliki do laptopa za każdym razem, gdy pomiar został zrobiony. Niestety po kilku próbach okazało się, że Raspberry Pi nie jest w stanie wykonywać tego kodu niezawodnie w szybkim tempie, jeśli działa również kod odpowiadający za działanie termometru. Z tego powodu spróbowano drugiej opcji - laptop pobierałby w takim scenariuszu cyklicznie dane pomiarowe z Raspberry Pi. Napisano do tego funkcję **data_download**:

6.6 Zapis pomiarów do bazy danych

Dane pomiarowe zapisywane na minikomputerze Raspberry Pi zapisywane są ciągiem w jednym pliku CSV. Aby zapisać dane z pomiarów do bazy danych, należało rozbić zadanie na dwie części. Pierwszą z nich było wysłanie danych z Raspberry Pi do zewnętrznego serwera, którym w naszym projekcie będzie laptop podłączony do tej samej sieci co Raspberry Pi. Umożliwia nam to połączenie obu maszyn poprzez protokół **SSH** (Secure Shell). Dzięki temu, jesteśmy w stanie skorzystać z protokołu sieciowego **SCP** (Secure Copy Protocol), który służy przesyłowi danych pomiędzy dwoma urządzeniami połączonymi przez SSH. SCP działa jako komenda w terminalu i można skorzystać z niej na dwa sposoby - albo przesyłać pliki z urządzenia, na którym wywoływana jest komenda, do drugiego urządzenia; albo pobierać pliki z innego urządzenia na to, na którym wywołano komendę. W pierwszym podejściu nasz zespół spróbował napisać kod w języku python, który działałby na Raspberry Pi i wysyłał pliki do laptopa za każdym razem, gdy pomiar został zrobiony. Niestety po kilku próbach okazało się, że Raspberry Pi nie jest w stanie wykonywać tego kodu niezawodnie w szybkim tempie, jeśli działa również kod odpowiadający za działanie termometru. Z tego powodu spróbowano drugiej opcji - laptop pobierałby w takim scenariuszu cyklicznie dane pomiarowe z Raspberry Pi. Napisano do tego funkcję **data_download**:

```
1 import subprocess
2 import os
3 from time import sleep
4
5
6 def send_file(file_source: str, destination: str):
7     scp_command = f"scp {file_source} {destination}"
8     try:
9         subprocess.run(scp_command, shell=True, check=True)
10        print(f"File {file_source} transferred successfully.")
11    except subprocess.CalledProcessError as e:
12        print(f"Error occurred while transferring file: {e}")
13
14 if __name__ == "__main__":
15
16     dane = []
17     new_dane = []
18     number_of_files = 0
19
20     flag_work = True
21     while flag_work:
22         send_file("team15@192.168.0.91:~/Desktop/bramka_pomiarowa/dane.csv",
23                   "C:\\\\PW\\\\Proj_gr\\\\Python_dane\\\\dane_combined")
24         try:
25             with open("C:\\\\PW\\\\Proj_gr\\\\Python_dane\\\\dane_combined\\\\dane.csv", "r") as file:
26                 new_dane = [el.strip() for el in file]
27                 file.close()
28         except:
29             print("nie odnaleziono pliku :(")
30
31         new_files_number = 0
32         new_files_number = len(new_dane) - len(dane)
33         flag_found = False
34         if new_files_number != 0:
35             flag_found = True
36             print(new_files_number)
37             ##dane = new_dane[:]
38             # print(len(new_dane))
39             # print(new_files_number)
40
41             for i in range(new_files_number):
42                 # for i in range(5):
43                 with open(f"C:\\\\PW\\\\Proj_gr\\\\Python_dane\\\\dane\\\\plik_{number_of_files}.csv", "w") as file:
44                     file.write(new_dane[len(dane) - 1 + i + 1])
```

```

46     file . close ()
47     number_of_files += 1
48
49     dane = new_dane [:]
50
51     if flag_found :
52         sleep (5)
53         print ("dzialam w stanie: active")
54     else :
55         sleep (10)
56         print ("dzialam w stanie: idle")

```

Kod zaczynamy od zainportowania potrzebnych bibliotek - **subprocess**, **os** oraz **time**, z którego wykorzystamy funkcję **sleep**.

Następnie definiujemy funkcję **send_file**, która jako parametry przyjmuje dwa Stringi - **file_source**, który reprezentuje miejsce pliku, który chcemy pobrać z docelowego urządzenia, oraz **destination**, czyli miejsce do którego chcemy pobrać dany plik. **scp_command** to String przechowujący polecenie SCP. Korzystamy z "try-except", aby spróbować wykonać polecenie **subprocess.run(scp_command, shell=True, check=True)**. Wykonuje ona nasze polecenie SCP w "shellu", czyli tak, jakby było one ręcznie wywołane w terminalu. W przypadku, gdy wystąpi jakikolwiek błąd, np. gdy wprowadzono złe miejsce docelowe, "except" złapie błąd i wypisze go na ekranie. Jeśli wszystko wyszło pomyślnie, pliki zostały pobrane, a na konsoli wypisze się komunikat o pomyślnym zadziaływaniu funkcji.

W głównej części kodu definiujemy 3 zmienne - **dane**, reprezentujące wszystkie dotychczasowe pobrane dane, **new_dane**, reprezentujące wszystkie dane pobrane w danej iteracji działania pętli oraz **number_of_files**, który trzyma liczbę wszystkich dotychczasowych pojedynczych zapisów danych. W nieskończonej pętli zaczynamy od użycia funkcji **send_file** do lokalnego serwera (laptopa), na którym wykonywany jest kod. Następnie zapisujemy wszystkie linijki z pobranego CSV jako pojedyncze elementy w liście **new_dane**; jeżeli pliku nie odnaleziono, "exception" wypisuje nam o tym komunikat. Następnie sprawdzamy ile jest nowych wpisów pomiarowych od ostatniego ściągnięcia danych - jeżeli liczba ta jest różna od 0, ustawiamy flagę **flag_found** na True. W pętli "for" zapisujemy poszczególne wpisy pomiarowe do oddzielnych plików tekstowych, każdy z plików ma swoją unikatową liczbę odpowiadającą liczbie plików w danej chwili. Na samym końcu lista dane przyjmuje wartości listy **new_dane**.

Drugie zadanie polegało na cyklicznym pobieraniu danych z lokalnego serwera (laptopa), aby przenieść je do bazy danych. Zostało to zrealizowane w "Javie" we frameworku **Spring Boot**, co zostało opisane w sekcji **Backend**.

6.7 Wyświetlanie wyników i komunikacja z użytkownikiem

W ciągu rozwijania projektu rozważono kilka opcji komunikacji bramki z użytkownikiem. Celem takiej komunikacji było przekazywanie informacji użytkownikowi takich jak: zmierzona temperatura i ewentualne komentarze do tego, instrukcje podawania danych osobowych i postęp przebiegu całego procesu.

Pierwszym pomysłem było użycie ekranu RC0802A - jest to dwuliniowy ośmioznakowy wyświetlacz LCD. Zamysłem było zamontowanie takiego małego ekranu na fizycznej bramce tak, aby użytkownik widział wszystkie wyświetcone komendy. Niestety pomysł zawiódł z powodu słabego wykonania samego ekranu - kilka razy kable po wpięciu do pinów na Raspberry Pi zostały rozerwane, co zmniejszało ich niezawodność w realnym działaniu.

Następnie spróbowano wykorzystać technologię text-to-speech (TTS), aby werbalnie przekazywać użytkownikowi wszystkie polecenia i komunikaty. Wykorzystano by wtedy jedną z wielu dostępnych bibliotek do TTS w Python'ie i wersja testowa została stworzona, jednak okazało się, że takie rozwiązanie również sprawia problemy. Implementacja TTS działała poprawnie, ale działała z wielkim opóźnieniem, tzn. zanim komunikat byłby usłyszany, program chciał już iść do następnej części kodu, co powodowało problem z synchronizacją programu z komunikatami głosowymi. Powodami tego mogło być dużo rzeczy - wydajność procesora Raspberry Pi, jego brak chłodzenia (Raspberry Pi często był wyczuwalnie ciepły), bądź połączenie internetowe, które nie było wystarczająco szybkie, aby program korzystający z biblioteki TTS mógł dostać (skompilowane w chmurze danej biblioteki) plik dźwiękowy do odegrania. Z tych przyczyn porzucono również tą opcję.

Finalnie zdecydowano się na sprawdzoną i prostą dla użytkownika metodę pokazywania tekstu na monitorze, w przypadku pokazu końcowych projektów - na ekranie laptopa (w rzeczywistym zastosowaniu takiej bramki podłączono ją do jakiegoś większego monitora bądź telewizora). Teksty były "printowane" na ekran terminalu w czytelny sposób dla osoby mierzącej swoją temperaturę. W pierwszej kolejności pokazywany był komunikat informujący o wystartowaniu aplikacji. Po podłożeniu ręki pod termometr pokazywał on ile sekund mierzy już temperaturę - jeżeli w trakcie pomiaru użytkownik odsunął rękę przerywając tym samym pomiar, wyświetlał się komunikat o tym, żeby użytkownik przywrócił swoją rękę pod termometr. Po pomyślnym pomiarze temperatury jest ona wyświetlana na ekranie, a potem pokazywane są instrukcje dotyczące zapisu danych osobowych. Na ekranie pokaże się komunikat aby podać swoje imię i nazwisko, a po chwili żeby potwierdzić, czy aplikacja poprawnie usłyszała te dane. Jeżeli nie są to poprawne dane, wyświetli się tekst, aby ponownie podać imię i nazwisko. Jeżeli dalej aplikacja źle zrozumie imię i nazwisko użytkownika poprosi go o przeliterowanie po kolej swoich danych. Jeżeli coś pójdzie nie tak (to znaczy wyskoczy błąd w kodzie), na ekran terminala będzie komunikat o przechwyceniu takiego błędu. Jeżeli wszystko poszło pomyślnie, na końcu będzie komunikat o pomyślnym przetworzeniu i zapisaniu danych, pokaże się również data i godzina zapisu oraz ponownie - zmierzona temperatura użytkownika.

6.8 Konstrukcja bramki

Od początku projektu, koncepcja samej konstrukcji bramki, w jaki sposób fizyczny zostanie ona zrealizowana, mocno się zmieniała. Kluczową kwestią, która wpływała na końcowy wygląd bramki, była decyzja, gdzie nasz czujnik miałby mierzyć temperaturę.

Ostatecznie wybrano, aby miejscem pomiaru temperatury była dłoń lub okolice nadgarstka. Taki wybór uznany został za najbardziej stosowny, gdyż nie jest uwarunkowany innymi parametrami, które trzeba byłoby uwzględnić przy innej budowie bramki, np. wzrost przy próbie pomiaru temperatury z czoła.

Początkowo zaprojektowaliśmy koncepcyjnie naszą bramkę, która miała mieć kształt sześcianu, do którego środka wprowadzana byłaby ręka do pomiaru. Następnie przeszliśmy do realizacji fizycznego projektu, do którego użyliśmy rurek i trójkątów PVC oraz listewki. Po odpowiednim przycięciu rurek, powstała bramka o wymiarach 25cm x 25cm x 25cm. Do górnej "ściance" zostały przyłączone listewki, potrzebne do zamontowania na nich osprzętu, czyli Raspberry Pi oraz mikrofon, wykorzystywany do podawania imienia i nazwiska. Na środku została wywiercona mała dziura, w którą został włożony czujnik temperatury w kierunku środka bramki. Poniżej znajduje się zdjęcie bramki z prezentacji projektów.

Cała bramka z osprzętem była podczas pomiarów połączona z dwoma laptopami, jednym do obsługi pomiaru, a drugim ukazującym stronę internetową z bazą danych wykonanych pomiarów.



Rysunek 12: Widok fizycznej bramki

7 Lista „commitów”

Tryb naszej pracy nad projektem spowodował, że commity robiliśmy rzadko, z dużymi przerwami między nimi. Obejmowały one znaczące zmiany w projekcie. Początkowe commity skupiały się na dodawaniu podstawowych plików, następnie wprowadzano nowe funkcjonalności i poprawki, a na końcu dodano finalne wersje aplikacji.

Jun 16, 2024

 finalna wersja aplikacji webowej
azatorsk authored 4 hours ago

 finalna wersja kodu obsługi czujnika
azatorsk authored 4 hours ago

Mar 30, 2024

 Upload New File
azatorsk authored 2 months ago

 Update 64 files 
azatorsk authored 2 months ago

 Merge branch 'master' into 'main' 
azatorsk authored 2 months ago

 Aplikacja do kontaktu z bazą danych
azatorsk authored 2 months ago

 Update 8 files 
azatorsk authored 2 months ago

 Upload New File
azatorsk authored 2 months ago

 Upload New File
azatorsk authored 2 months ago

 Upload New File
azatorsk authored 2 months ago

 Upload New File
azatorsk authored 2 months ago

 Upload New File
azatorsk authored 2 months ago

 Add new directory
azatorsk authored 2 months ago

 Delete .gitkeep
azatorsk authored 2 months ago

 Add new directory
azatorsk authored 2 months ago

 Add new directory
azatorsk authored 2 months ago

 Replace config.cfg
azatorsk authored 2 months ago

 dodano speech to text oraz uwzględniono błędy pomiarowe
azatorsk authored 2 months ago

Mar 04, 2024

 Add sending_test.py 
ado1 authored 3 months ago

 Add config.cfg
ado1 authored 3 months ago

 Edit projekt.py
ado1 authored 3 months ago

Jan 09, 2024

 Add projekt.py
ado1 authored 5 months ago

Rysunek 13: Historia commitów

9 Stycznia 2024

1. **Add projekt.py** - Pierwszy commit dodający plik *projekt.py*, stanowiący początek projektu. Dodanie pierwszej wersji aplikacji obsługującej czujnik temperatury MLX.

4 Marca 2024

1. **Edit projekt.py** - Przygotowanie aplikacji obsługującej czujnik temperatury do współpracy z plikiem konfiguracyjnym określającym najważniejsze parametry pracy, czyli zakresy temperatur.
2. **Add config.cfg** - Dodanie pliku konfiguracyjnego, zawierającego zakresy temperatur.
3. **Add sending_test.py** - Dodanie prototypu programu wysyłającego pomiary będące poszczególnymi plikami na serwer. Ostatecznie nie skorzystaliśmy z tego rozwiązania.

30 Marca 2024

1. **dodano speech to text oraz uwzględniono błędy pomiarowe** - Dodanie funkcjonalności rozpoznawania mowy, wprowadzono poprawki związane z błędami pomiarowymi oraz zaktualizowano mechanizm odczytywania wartości z pliku konfiguracyjnego o nowe, wymagane wartości.
2. **pozostałe commity** - Pozostałe commity związane są z przygotowaniem pierwszej wersji aplikacji webowej oraz poprawkami literówek w istniejących już plikach.

16 Marca 2024

1. **finalna wersja kodu obsługi czujnika** - Dodanie ostatecznej wersji kodu obsługującej czujnik temperatury.
2. **finalna wersja aplikacji webowej** - Obszerny commit dodający finalną wersję aplikacji webowej zapewniający zarówno funkcjonalny back-end, jak i efektywny front-end.

8 Repozytorium

Ponieważ listing kodów zająłby bardzo dużą ilość miejsca załączamy link do repozytorium, w którym znajdują się wszystkie kodu źródłowe.

<https://gitlab-stud.elka.pw.edu.pl/azatorsk/bramka-do-bezkontaktowego-pomiaru-temperatury-ciala>

9 Pierwsza prezentacja



CZŁONKOWIE ZESPOŁU

- Agata Zatorska - Scrum master
- Michał Jeczmieniowski - Product owner
- Anh Quan Do - Developer
- Grzegorz Papaj - Developer
- Filip Wojda - Developer



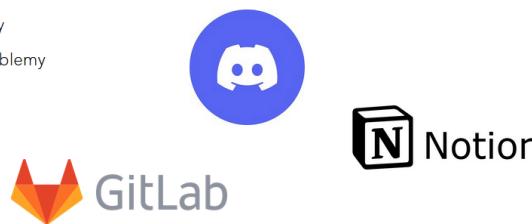
PRZYKŁADY UŻYCIA

- Termometr bezkontaktowy
- Bramka SE-1008



ORGANIZACJA PRACY

- Organizacja i przebieg spotkań
- Użyte programy
- Napotkane problemy



CO UDAŁO NAM SIĘ OSIĄGNĄĆ?

- Research
- Oprogramowanie obsługujące czujnik temperatury
- Sygnalizacja nieprawidłowości zmierzonej temperatury



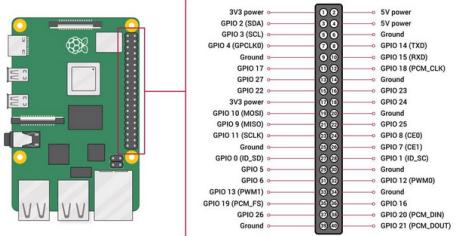
1/27/2024

Sample Footer Text

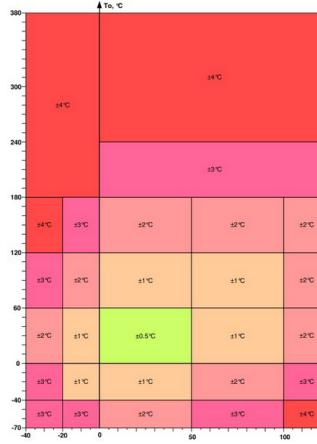
6

UŻYTE KOMPONENTY I OPROGRAMOWANIE

- Raspberry Pi 4 Model B 4GB
- MLX90614
- Python



DOKŁADNOŚĆ POMIARÓW DLA POSZCZEGÓLNYCH PRZEDZIAŁÓW TEMPERATUR



IMPLEMENTACJA KODU DO OBSŁUGI CZUJNIKA

- Użyte biblioteki

```
from smbus2 import SMBus
from mlx90614 import MLX90614
from time import sleep
import datetime
```

```
if __name__ == "__main__":
    bus = SMBus(1)
    sensor = MLX90614(bus, address=0x5A)
    i = 0
    temp_list = []
    if_meas = False
    print("Start urzadzenia")

    while True:
        amb = sensor.get_amb_temp()
        obj = sensor.get_obj_temp()
        ...
        else:
            sleep(1)
```

IMPLEMENTACJA KODU DO OBSŁUGI CZUJNIKA

- Ustalenie, czy pacjent jest przed czujnikiem

```
if abs(amb - obj) > 5.0 and if_meas == False:
    print("Pacjent wykryty!")
    print("Stój bez ruchu...")
    print("0s")
    if_meas = True
```

IMPLEMENTACJA KODU DO OBSŁUGI CZUJNIKA

- Pomiar

```
if if_meas:  
    if abs(amb - obj) > 5.0:  
        if i == 0:  
            i += 1  
            sleep(0.2)  
    ...  
    elif i == 50:  
        print("is")  
        i += 1  
        sleep(0.02)  
    elif i == 100:  
        print("2s")  
        i += 1  
        sleep(0.02)  
    else:  
        temp_list.append(obj)  
        i += 1  
        sleep(0.02)  
  
#If i == 150:  
avg = sum(temp_list)/(len(temp_list))  
print("%s")  
if avg < 34:  
    print("Temperatura wynosi: {avg:.2f}. Jest poniżej normy.")  
elif avg > 36 and avg < 39.5:  
    print("Temperatura wynosi: {avg:.2f}. Masz gorączkę.")  
elif avg > 39.5:  
    print("Temperatura wynosi: {avg:.2f}. Stan krytyczny.")  
else:  
    print(f"Body Temperature: {avg:.2f}")  
  
...  
  
print(datetime.datetime.now())  
print("-----")  
i = 0  
temp_list = []  
if_meas = False  
sleep(5)
```

IMPLEMENTACJA KODU DO OBSŁUGI CZUJNIKA

- Opuszczenie pola widzenia czujnika przez pacjenta

```
else:  
    print("Wracaj pacjencie, pomiar od nowa!")  
    if_meas = False  
    i = 0  
    temp_list = []  
    sleep(4)
```

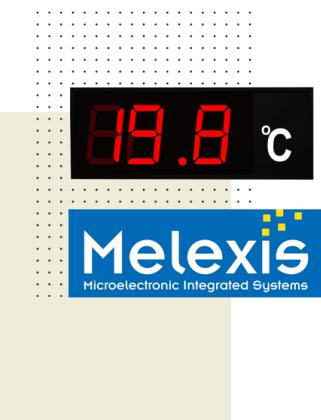
IMPLEMENTACJA KODU DO OBSŁUGI CZUJNIKA

- Zapis do pliku

```
with open("dane.csv", "a") as file:  
    acc = accuracy(amb, avg)  
    time = datetime.datetime.now().strftime("%d-%m-%Y %H:%M:%S")  
    temp=f"{avg:.2f},{amb:.2f},{time},{acc}\n"  
    file.write(temp)
```

- Funkcja opisująca dokładność pomiaru dla danego przedziału temperatur

```
def accuracy(amb,obj):  
    if(amb>=0 and amb<=50 and obj>=0 and obj <=60):  
        return 0.5  
    else:  
        return 1.0
```



PLAN Y NA PRZYSZŁOŚĆ

- Korekta błędów pomiarów:
 - Podawane napięcie 3.3V, zamiast 3V zalecanych przez producenta;
 - Uwzględnienie współczynnika emisyjności dla poszczególnych materiałów;
- Wyświetlacz do odczytu wyników:
 - Dogodna forma wskazania odczytanej temperatury po pomiarze;
 - Dodatkowa informacja o nieprawidłowej temperaturze;

PLAN Y NA PRZYSZŁOŚĆ cd.



- Czytelny interfejs użytkownika;
- Baza danych zapisująca pomiary;
- Estetyczna i funkcjonalna konstrukcja.

D Z IĘ K U J E M Y Z A
U W A G È !