

Jan Zobniów, Adrian Grzechnik, Sofia Levchenko, Aleksandra Harasimik,
Mateusz Izbicki

Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych Politechniki
Warszawskiej w Warszawie

INCZ

-

Bezprzewodowy monitoring stanu bydła

Dokumentacja projektu

12 lipca 2022

Spis treści

1. Wstęp	2
1.1. Zdefiniowanie problemu	2
2. Organizacja projektu	3
2.1. Organizacja współpracy	3
2.2. Podział prac	3
2.3. Wykres Ganta	4
2.4. Fazy projektu	4
3. Zdefiniowanie wymagań	5
3.1. Stworzenie persony użytkownika	5
3.2. Określenie wymagań	6
3.3. Kluczowa funkcjonalność	6
4. Analiza rynku	7
4.1. AnTrack	7
4.1.1. Moduł śledzący oparty na GPS	8
4.1.2. Moduł do transmisji danych	8
4.1.3. Moduł panelu słonecznego	9
5. Propozycja rozwiązania	10
6. Implementacja	12
6.1. Część hardware'owa	12
6.1.1. Założenia, implementowana funkcjonalność	12
6.1.2. Mikrokontroler STM32	12
6.1.3. Moduł radiowy LoRa	13
6.1.4. Moduł GPS	14
6.1.5. Czujnik temperatury	15
6.1.6. Bateria	16
6.2. Część software'owa	21
6.2.1. Założenia, implementowana funkcjonalność aplikacji mobilnej	21
6.2.2. Aplikacja mobilna	22
6.2.3. Konfiguracja bazy danych	24
6.2.4. Połączenie aplikacji mobilnej z bazą danych	25
6.2.5. Połączenie bazy danych z TTN'em	26
6.3. Kod źródłowy	27
7. Prototyp	28
8. Podsumowanie	29
Literatura	30

1. Wstęp

W ramach projektu z przedmiotu INCZ naszym zadaniem było zdefiniowanie, a następnie rozwiązanie wybranego problemu przy pomocy sieci inteligentnych czujników.

Zaproponowanie rozwiązania zawiera w sobie analizę istniejących rozwiązań, zaprojektowanie odpowiedniej sieci inteligentnych czujników oraz implementację prototypu.

1.1. Zdefiniowanie problemu

Nasz zespół zdecydował, że będziemy zajmowali się problemem z obszaru gospodarstwa domowego.

W obecnych czasach, większość farm musi monitorować lokalizację bydła samodzielnie, przez co pewna liczba zwierząt hodowlanych gubi się na terenie otwartym. Skutkuje to w większej ilości poświęcanego czasu przez farmerów na odnajdywanie zwierząt, co może być niewykonalne przy odpowiednio licznym stadzie.

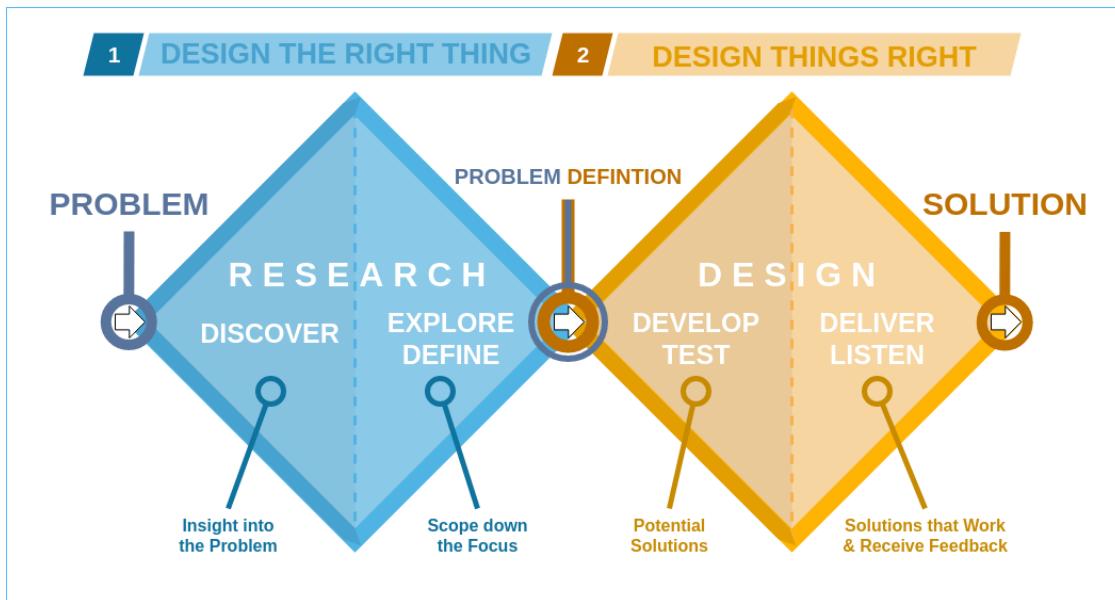
Jednym z rozwiązań jest ograniczenie pastwiska, jednak wpływa to negatywnie na stan zdrowotny wypasanych zwierząt (czuje się ono wtedy bardziej zestresowane), co z kolei wpływa na jakość produktów, które możemy z nich czerpać np. mleka

Zdecydowaliśmy się na wybór rozwiązania tego problemu i zaproponowaliśmy rozwiązanie w postaci systemu do monitoringu stanu bydła, które pozwoli na łatwiejsze monitorowanie hodowanych zwierząt. Farmerzy zostaną odciążeni (nie będą musieli już pilnować zwierząt przez cały czas) oraz zostanie polepszony stan zdrowotny bydła (jako że już nie musi znajdywać się wyłącznie w ramach terenu zamkniętego).

2. Organizacja projektu

2.1. Organizacja współpracy

Współpraca w projekcie opierała się na popularnym w tych czasach modelu tworzenia **Double Diamond**, w którym ważną rolę gra research oraz wdrażanie nowych wersji w życie projektu.



Rysunek 1. Wizualizacja modelu Double diamond

Idąc zgodnie z modelem który przyjeliśmy, postanowiliśmy że część związaną z researchem rozwiążemy poprzez cotygodniowe spotkania na serwerze w aplikacji Discord. Na tych spotkaniach wymienialiśmy się zdobytą wiedzą oraz planowaliśmy kolejne cele w projekcie.

Czas tygodnia pomiędzy spotkaniami został przeznaczony na część modelu związaną z projektowaniem i tworzeniu części potrzebnych do realizacji celów w danym tygodniu.

2.2. Podział prac

W ramach naszego zespołu podzieliliśmy się na dwie grupy przewodnie, dzięki którym chcieliśmy zachować równowagę pracy, ale również zoptymalizować rozwój:

- grupa Software - 3 osoby
- grupa Hardware - 2 osoby

Dzięki takiemu podejściu chcieliśmy być bardziej elastyczni na zmiany.

Mimo podziału na grupy, każdy z nas poprzez cotygodniowe spotkania wiedział co "dzieje się w innej grupie".

2.3. Wykres Ganta



Rysunek 2. Wykres Gantta

2.4. Fazy projektu

Zgodnie z przedstawionym wykresem Gantta [2] projekt podzieliliśmy na 3 główne fazy:

- Research
- Development
- Testing

W pierwszym etapie miała zostać podjęta decyzja o wyborze tematu projektu oraz sposobu jego realizacji. W tej fazie każdy miał samodzielnie szukać lub wymyślać możliwe tematy a następnie implementacje wybranego przez nas temat. Etap ten zakończył się pod koniec marca.

Kolejny etap, czyli development. Jest to najdłuższa faza w projekcie i zawierają się w niej dwie "gałęzie rozwoju" - rozwój software oraz hardware. W ramach zrealizowania równoległych prac nad oba gałęziami postanowiono, jak zostało wspomniane we wcześniejszym punkcie, dwie grupy, która każda z nich realizuje jedną z gałęzi.

Grupa Hardware, wykonująca gałąź hardware, miała na celu zaprojektowanie oraz zbudowanie czujnika, który byłby montowany na krowie i wysyłałby niezbędne dane do naszych serwerów.

Równolegle pracowała grupa Software, której zadaniem było stworzenie warstwy klienckiej (aplikacji mobilnej) oraz warstwy logicznej, która będzie zarządzała informacjami, które zostaną przesyłane przez zaprojektowany przez grupę Hardware czujnik.

Na ten etap zostało zaplanowane ponad półtora miesiąca.

Ostatnią fazą było testowanie, w której badaliśmy zachowanie się czujnika oraz sprawdzaliśmy czy wszystkie integracje systemu działają poprawnie (o nich będzie dalej w dokumentacji).

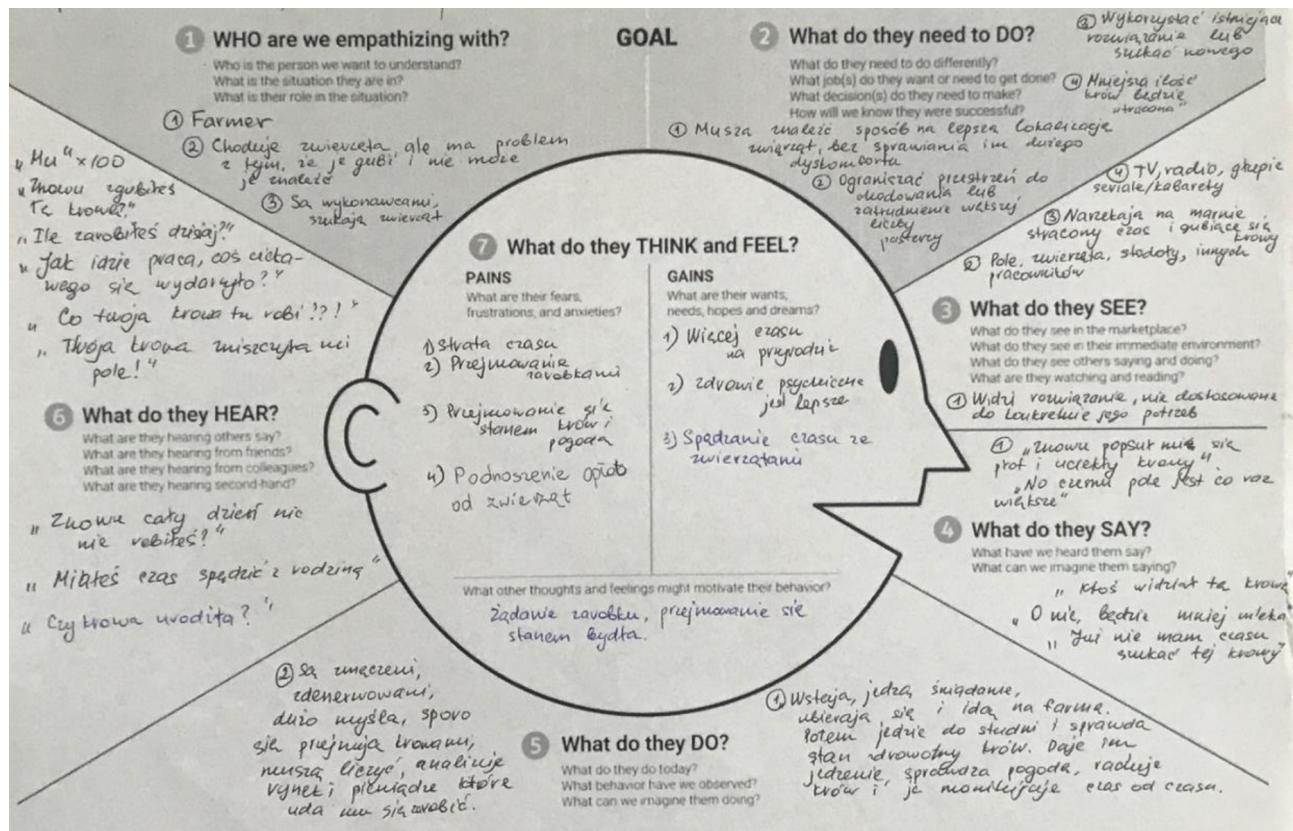
3. Zdefiniowanie wymagań

Podczas **fazy przeprowadzenia research'u**, między innymi, zastanowiliśmy się, jakie wymagania nasze rozwiązanie musi spełniać, aby w odpowiedni sposób go zaprojektować.

Zaczęliśmy od stworzenia persony, czyli reprezentacji typowego użytkownika, systemu, który mamy na celu stworzyć.

3.1. Stworzenie persony użytkownika

Najpierw określiliśmy typowe środowisko naszej persony, poprzez zdefiniowanie tego, **co ona robi na co dzień, w jakim środowisku znajduje się, co czuje, co mówi, co słyszy oraz o czym myśli**. Takie określenie przedstawiliśmy w formie Rysunku 1.



Rysunek 3. Określenie środowiska otaczającego personę

W kolejnym kroku, na podstawie już zdefiniowanego środowiska, otaczającego naszą personę, również stworzyliśmy jej bardziej konkretną reprezentację, abyśmy mogli się zastanowić, **z czego naszemu użytkownikowi byłoby najwygodniej korzystać** oraz **czego naszej personie nie starczy** w codziennym życiu (abyśmy mogli częściowo załatwiać te problemy za pomocą naszego produktu). Taką reprezentację możemy zaobserwować na Rysunku 2.

Zespół 5		
Autorzy: Tębićki, Lewandowski, Małasznik, Zubrowski Brychnicki		Data: 3.9.2022
Historia <small>a. Ostatni rok życia, obecnie: studentka szkółki rolniczej. b. Chciała kreatywnie, opisać, który nowoczesny dla problemu, usługa, który przyczyniłby się i w jaki sposób zwiększać swoje życie mieszka na wsi. Skoncentrował się na technikum z średnim wynikiem. W technikum poznaje swoja rola. Irena. Przejąć gospodarstwo od swojego ojca, bo nie może zatrudnić się o tym od inego. Jego życie składa się z krow, farma i rodziny. Jest postulatem związanym z prostymi problemami: chcieli by się wszystkie problemy szybko rozwiązać!</small>	Cele <small>Oznacza, że jestem kreatywną jako piękno oraz moje doświadczenie chce zwiększyć jak najwięcej pieniężne choć utrzymać gospodarstwo chce być najlepszym Farmerem chce utrzymać rodzinę</small>	Potrzeby <small>Najważniejsze, najpotrzebniejsze rzeczy, wydajenia które muszą zaspakiać aby działać wyżej trwały dobrej jakości, zatrudnienia zapotrzebowanie na dystrybucję produktu Wystarczająco prostsze do chodzenia zwierząt</small>
Obawy <small>Wyszczególniając problemy, utrudniające życie. Co jak wie karmy rady samemu? Seron sie nie uda, krowy zatrzymają się jak moj produkt nie będzie atrakcyjny?</small>	Przydatne <small>Rozumiem, że kreatywne nie napiszysz. Nie mamy też żadnych aby celów przedstawionych, ani postulatów kreatywności. Dofinansowanie dodatkowe rach do pracy Dodatkowe narzędzia do monitorowania cypla na farmie</small>	
Imię i nazwisko: Zdzisław Nowak Stanowisko: senior farmer Branża: gospodarstwo rolnicze Wiek: 46	Oprogramowanie / usługi / rozwiązania <small>Używa często Radio, Google Maps, WhatsApp, Viber, Facebook, TV, Drony</small>	Używa rzadko <small>Laptop, smartphone,</small>

Autorem karty jest Piotr Marszałkowski - ux.marszalkowski.org

Rysunek 4. Skonkretyzowana reprezentacja persony

3.2. Określenie wymagań

Po przeanalizowaniu stworzonej wcześniej persony, doszliśmy wniosku, że nasz użytkownik:

- częściej korzysta z komórki niż z komputera stacjonarnego,
- jest zapoznany ze sposobem korzystania z takich aplikacji jak Google Maps i powszechnie używanych mesendżerów
- nie ma zapewnionej przewodowej łączności z siecią Internet (tylko bezprzewodowa)

I że potrzebuje:

- pomocy z prowadzeniem gospodarstwa (przykładowo, z monitorowaniem bydła na farmie)
- aby jego produkt (w tym przypadku mleko, które mu daje krowa) był najlepszy jak się da
- co wpływa na zdrowie jego zwierząt (w tym przypadku krów)

3.3. Kluczowa funkcjonalność

W taki sposób, udało nam się wyodrębnić kluczowe funkcje, które by nasz użytkownik potrzebował najbardziej i byłyby dla niego wygodne:

- zautomatyzowane monitorowanie lokalizacji bydła w czasie rzeczywistym
- zautomatyzowane monitorowanie stanu środowiska, otaczającego bydło (przykładowo, czujnik temperatury)
- dostęp do informacji o lokalizacji bydła poprzez aplikację mobilną
- wyszukiwanie ścieżki, prowadzącej do jednostki bydła, w sposób podobny do tego, jak to robi aplikacja Google Maps
- zapewnienie interfejsu aplikacji mobilnej, podobnego do interfejsu typowego messenger'a
- przetwarzanie danych poza farmą (w najlepszym przypadku w chmurze)
- niezbędność jedynie połączenia bezprzewodowego z Internetem

4. Analiza rynku

Przed rozpoczęciem prac nad naszym projektem zajęliśmy się przeglądem istniejących już na rynku rozwiązań. Jako efekt końcowy projektów największą popularnością cieszyła się obroża. Niektóre z rozwiązań zamiast jednego produktu końcowego proponowały dwa, obrożę i klips na ucho, wtedy obroża mogła mieć mniejszy rozmiar. Naszą uwagę przykuło jedno rozwiązanie, które było najbardziej zbliżone do naszej wizji. Rozwiązanie te nosi nazwę **AnTrack** [1] i zostało ono dokładniej opisane w rozdziale 4.1.

4.1. AnTrack

Urządzenie te zostało stworzone ze względu na to, że na terenach Południowej Afryki i Kenii pojawił się duży problem związany z coraz częściej obserwowanymi zdarzeniami kradzieży bydła. Powodowało to duże straty dla farmerów zajmujących się nimi. Jednym ze sposobów ograniczenia takich zdarzeń było prowadzenie częstej ewidencji bydła w gospodarstwie. Rozwiązanie te zapobiegało większej eskalacji kradzieży, jednak było bardzo pracochłonne. Z tego względu wymyślono bezprzewodowe rozwiązanie odpowiedzialne za śledzenie zwierząt gospodarskich w czasie rzeczywistym.

Rozwiązaniem jest inteligentna obroża do noszenia przez zwierzęta. Śledzi ona zwierzęta w środowisku zewnętrznym za pomocą technologii częstotliwości radiowej. Obroża ta jest wodoszczelna i posiada panele słoneczne, które są w stanie wygenerować wystarczającą ilość energii, aby zasilić urządzenie nawet wtedy, kiedy zachmurzenie utrzymywałoby się przez tydzień.

Rozwiązanie te składa się z trzech głównych elementów:

- **czujniki** - obroże jako Bezprzewodowa Sieć Sensoryczna
- **serwer** - chmura
- **klient** - komputery lub smartfony użytkowników końcowych

Zwierzęta są wyposażone w obroże bezprzewodowe posiadające interfejsy GPS, dzięki którym możliwe jest określenie lokalizacji zwierzęcia. Z powodu dużego zużywania energii przez czujniki GPS, nie są one aktywne przez cały czas, lecz pracują w interwałach przełączając się między trybem aktywnym, a autonomicznym. Odstęp czasu są zależne od typu ruchu danego zwierzęcia. Lokalizacja GPS jest przesyłana do chmury w czasie rzeczywistym. Do tego celu została wykorzystana komunikacja bezprzewodowa. Dane te mogą być przesyłane za pomocą różnych metod np. GSM, czy też protokołów komunikacyjnych dalekiego zasięgu jak LoRaWan.

W tym scenariuszu nie wszystkie zwierzęta muszą być wyposażone w takie moduły. Może być tak, że dane GPS od poszczególnych zwierząt będą wysyłane do sąsiednich zwierząt posiadających moduł do bezprzewodowej komunikacji. Do komunikacji między zwierzętami mogą zostać wykorzystane protokoły takie, jak Zigbee, Z-Wave lub BLE. W celu oszczędzania energii, niektóre zwierzęta, a dokładniej ich obroże, mogą stosować algorytmy do uzyskiwania informacji o swojej lokalizacji wykorzystując w tym celu dane GPS od sąsiadnego zwierzęcia. Do tego stosuje się różnego rodzaju techniki, od trilateracji, aż po złożone algorytmy optymalizacyjne.

Centralną częścią architektury jest chmura, która zajmuje się gromadzeniem wszystkich danych i wykonywaniem złożonych zadań dotyczących ich analizy. Ze względu na szybki rozwój dronów, możliwe jest również wykorzystanie ich do gromadzenia danych. Obroża może wysyłać bezpośrednio do drona dane (za pomocą tanich protokołów), podczas gdy on odpowiedzialny byłby za przesłanie ich do chmury. W takim przypadku dron działałby jako hub lub brama między obrożami, a chmurą. Klienci otrzymują informacje zwrotne z chmury za pomocą standardowych interfejsów takich, jak aplikacje

mobilne lub webowe.

AnTrack posiada trzy kluczowe moduły:

- lokalizator oparty na GPS
- panel słoneczny używany do doładowywania baterii
- transmitter

Kompletny system jest wykonany w postaci obroży. Na jej pasku umieszczona zostanie część elastycznych ogniw słonecznych, które zostaną podłączone do urządzenia znajdującego się pod szyją zwierzęcia na pasku. Na obroży znajduje się również wodoszczelna obudowa, w której umieszczono układ elektroniczny i akumulator. Jednostka mikrokontrolera (MCU) odpowiedzialna jest za sterowanie wszystkimi urządzeniami podłączonymi do systemu. System zawiera dwa źródła energii, akumulator i panel słoneczny. Gdy panel słoneczny dostarcza więcej energii, niż potrzebuje system, wtedy podsystem zarządzania zasilaniem ładuje akumulator aż do jego pełnego zasilenia. Jeśli panel słoneczny nie dostarcza wystarczającej ilości energii, system będzie polegał tylko i wyłącznie na akumulatorze. MCU steruje zasilaniem modułu GPS i transceivera RF. Aby ograniczyć zużycie energii do minimum, MCU wykorzystuje swoje wewnętrzne oscylatory i jest w trybie uśpienia, jeśli system nie rejestruje pozycji lub nie nadaje. Moduł GPS ma stałe zasilanie do jego pinu V Backup. Aby upewnić się, że SRAM będzie zawsze zasilany, napięcie będzie obniżane do określonego zakresu. MCU ma podłączoną zewnętrzną pamięć, w której zapisywane są dane o lokalizacji, kiedy stacja bazowa jest poza zasięgiem.

Do wykonania takiego systemu potrzebne jest użycie dwóch technologii bezprzewodowych. Jedna z nich służy do lokalizowania położenia zwierzęcia, druga zajmuje się przesyłaniem danych do serwera w celu ich analizy.

4.1.1. Moduł śledzący oparty na GPS

W systemie AnTrack GPS jest wykorzystywany do śledzenia zwierzęcia w jego naturalnym środowisku. Pozycja zwierzęcia jest rejestrowana przez mikrokontroler w regulowanych odstępach czasu. Używany jest moduł GPS **GPTA010**. Działa on ze źródłami napięcia od 3,0 V do 4,3 V (zwykle 3,3 V). Przy szybkości aktualizacji wynoszącej 1 Hz, typowy pobór prądu przez urządzenie wynosi około 25 mA, co przekłada się na zużycie energii na poziomie 82 mW. Przy śledzeniu pozycji, urządzenie zużywa średnio 20 mA prądu. Na potrzeby tego zastosowania, śledzenie nie będzie potrzebne, ponieważ lokalizacja będzie pobierana co 15 minut.

Moduł ten wykorzystuje do komunikacji uniwersalny asynchroniczny odbiornik/nadajnik - **UART**. GPS przekazuje dane do urządzenia podłączonego przez UART za pomocą protokołu NMEA. Zdania NMEA są przesyłane z częstotliwością 1 Hz, ale może być ona regulowana. W każdym takim zdaniu wyświetlane są różne informacje o położeniu, prędkości, czasie i statusie satelitów.

4.1.2. Moduł do transmisji danych

AnTrack może korzystać z dwóch metod przesyłania danych o położeniu do serwera, oparte są one na technologiach GSM i RF. Transmisja oparta na GSM - moduł GSM jest w stanie przesyłać informacje o położeniu zwierzęcia wykorzystując dostępną już sieć GSM dostarczoną przez dostawcę usług komórkowych. Jednak rozwiązanie te ma swoje wady, ponieważ każde urządzenie będzie potrzebowało karty SIM do poprawnego działania. A karta ta musiała być ładowana za każdym razem, kiedy będzie się zbliżać do wyczerpania pakietu danych. W tym celu można zastosować interfejs API, który będzie monitorować każdą kartę SIM, a serwer będzie mógł określić która karta potrzebuje doładowania i odpowiednio wykonać te doładowanie. Istotne jest to, że moduły te są w stanie pobierać prądy szczytowe o natężeniu 2A, a to wymaga odpowiednio naładowanego akumulatora. Moduł ten wykorzystuje

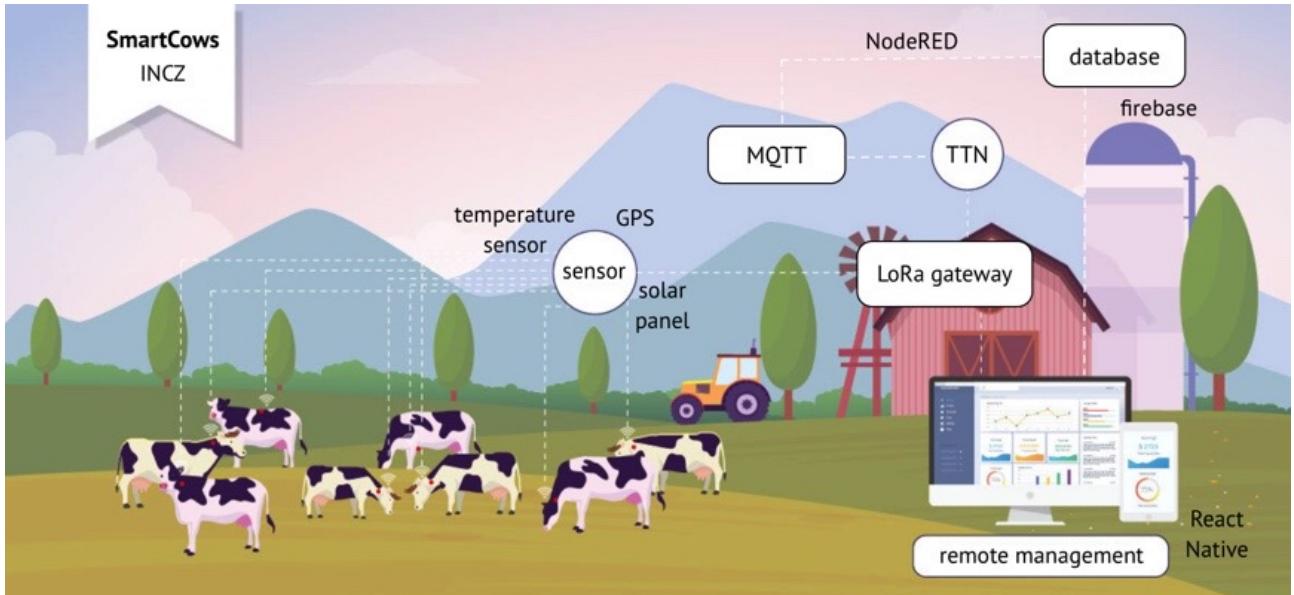
do komunikacji UART. Używa on poleceń AT do komunikacji z urządzeniami zewnętrznymi. Moduł GPRS jest odpowiedzialny za ustanowienie łączności z siecią komórkową. Kiedy GPRS jest używany i transmisuje dane, to pobiera prąd w zakresie 156 mA - 408 mA, zależy on od liczby używanych kanałów RX/TX. Niemożliwe jest dokładne przewidzenie pobieranego prądu, dlatego zawsze zakłada się najgorszy przypadek.

4.1.3. Moduł panelu słonecznego

Elastyczne panele słoneczne są nowym rozwiązaniem w tej branży. W tym zastosowaniu użyte zostały **PowerFilm PT15-75** o szerokości 90 mm i długości 270 mm. Są one zabezpieczone i uszczelnione przed wodą. Panel ten będzie generować średnio 2740 mWh dziennie. Obliczono, że dzienne zapotrzebowanie na energię będzie wynosiło 23,99 mWh (przy nRF24L01). Do zapewnienia niezawodnego działania AnTrack, starczy to na zasilenie urządzenia przez 5 dni (jest to przybliżona wartość szacunkowa). W przypadku użycia rozwiązania z kartą SIM (SIM800), dzienne zapotrzebowanie na moc zwiększy się do 2,75 Wh. Opcja ta nie jest wystarczająca, aby zapewnić niezawodność systemu. Aby rozwiązać ten problem, należy zwiększyć odstęp czasu wysyłania danych lokalizacji do 1h. Spowoduje to 4-krotne zmniejszenie zużycia energii.

5. Propozycja rozwiązania

Zatem, na podstawie zdefiniowanych wymagań oraz biorąc pod uwagę już istniejące rozwiązania, zdefiniowaliśmy architekturę rozwiązania, która wygląda w sposób, przedstawiony na Rysunku 3.



Rysunek 5. Architektura proponowanego rozwiązania

W tak zdefiniowanej architekturze, każde obserwowane zwierze posiada własny **czujnik** wyposażony w **moduł GPS**, czujnik temperatury, **panel słoneczny** oraz **moduł LoRa**. W ten o to sposób każdy z czujników jest w stanie komunikować się z bramą LoRa i przesyłać informację o obecnej lokalizacji bydła, temperaturze środowiska otaczającego bydło oraz o poziomie naładowania baterii takiego czujnika.

Aby taka komunikacja była możliwa, farma musi posiadać własną **bramkę LoRa** lub być umiejscowiona w zasięgu takiej bramki.

Przy pomocy bramki LoRa, dane sczytywane z czujnika są wysyłane do **The Things Network** (TTN, czyli globalny ekosystem Internetu Rzeczy przeznaczony dla współpracy, który jest wykorzystywany do utworzenia sieci, urządzeń i rozwiązań z wykorzystaniem LoRaWAN). TTN przetwarza otrzymane dane (przykładowo, konwertuje dane z postaci bitowej na postać w postaci obiektu JSON, czytelnej dla człowieka) oraz udostępnia je przez własnego MQTT brokera dla wszystkich chętnych (którzy posiadają klucz dostępu).

Aby móc, odbierane dane w jakiś sposób udostępniać naszemu użytkownikowi, musimy nasze dane gdzieś przechowywać. Zdecydowaliśmy zatem wybrać platformę **firebase** (utrzymywana przez Google) i jej usługę **real-time-database** do przechowywania danych, wysyłanych "na żywo" i następnie połączyć naszą usługę z aplikacją przy pomocy odpowiedniego API.

Jednym z ostatnich elementów, potrzebnych dla umożliwienia komunikacji między czujnikiem, a aplikacją użytkownika jest połączenie TTN'u z firebase'em. Tutaj skorzystaliśmy z narzędzia znanego jako **NodeRED** do pobierania danych udostępnianych przez TTN (w tym przypadku łączymy się przy pomocy **protokołu MQTT** do naszej sieci i wychwytywaliśmy wszystkie wiadomości, przesyłane

przez sieć), ich przefiltrowania oraz przekierowywania do bazy danych firebase'u.

Aby móc zwizualizować wyniki w odpowiedni sposób, skorzystaliśmy z języka javascript i framework'u **React Native**, umożliwiającego tworzenie **aplikacji natywnych** (czyli takich, które mogą być wykorzystywane na urządzeniach każdego typu, od urządzeń mobilnych do przeglądarek i komputerów stacjonarnych) oraz bibliotek pomocniczych, umożliwiających pobieranie danych z bazy danych, pokazywanie lokalizacji czujników, tworzenie ścieżek oraz rozpoznawanie lokalizacji użytkownika.

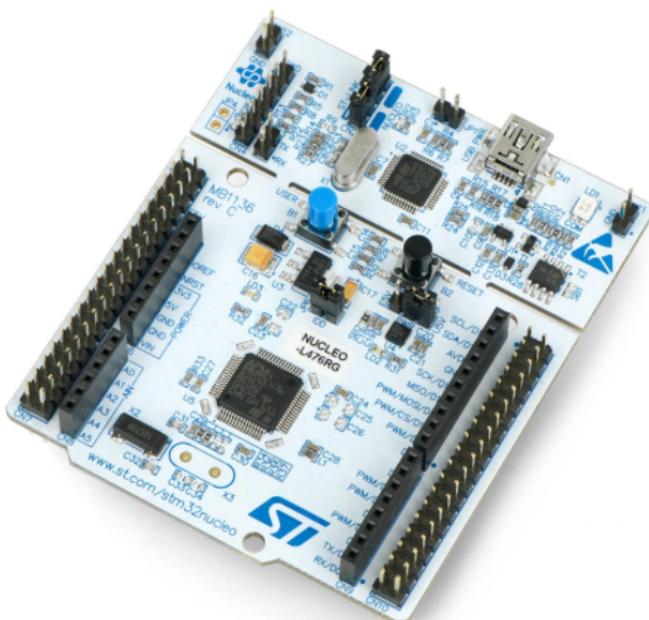
6. Implementacja

6.1. Część hardware'owa

6.1.1. Założenia, implementowana funkcjonalność

Na podstawie wcześniejszej przeprowadzonej analizy, zdecydowaliśmy się na wykonanie czujnika o niewielkich rozmiarach, działającego na zasilaniu baterijnym. Czujnik okresowo będzie wykonywał lokalizowanie za pomocą GPS oraz pomiar temperatury, a następnie będzie wysyłał te dane korzystając z sieci LoRa. Dodatkowo zasilanie czujnika będzie wspomagane panelem słonecznym.

6.1.2. Mikrokontroler STM32



Rysunek 6. Moduł Nucleo

Sercem czujnika jest moduł Nucleo z 32-bitowym mikrokontrolerem STM32L476RG, który wyposażony jest w jeden rdzeń ARM Cortex M4 o maksymalnym taktowaniu 80 MHz, pamięci Flash 1 MB, 128 kB pamięci SRAM. Według dokumentacji [2] mikrokontroler ten w trybie Standby złączonym zegarem RTC powinien pobierać prąd o wartości $1,4 \mu A$, a podczas normalnej pracy pobiera $100 \mu A/MHz$. Układ obsługuje też 20 interfejsów komunikacyjnych, m.in. 5x USART, 3x I2C czy 3xSPI, co pozwala na podpięcie wielu czujników.

6.1.3. Moduł radiowy LoRa

Jako sposób komunikacji wykorzystaliśmy LoRe. Jest to technologia bezprzewodowej transmisji danych, cechująca się transmisją na duże odległości przy bardzo małym poborze prądu, na czym nam zależało.



Rysunek 7. Moduł Lora-E5

Dlatego jako modułu radiowego użyliśmy LoRa-E5 STM32WLE5JC. Według dokumentacji [3] podczas operacji wysyłania zużywa 111mA, a podczas oczekiwania i odbierania danych zużywa 6,7 mA. W trybie low-power zużywa jedynie $2 \mu\text{A}$. Komunikuje się z kontrolerem przez interfejs USART za pomocą komend AT.

Po odpowiednim napisaniu algorytmu łączenia się z siecią LoRa, udało nam się uzyskać możliwe krótkie czasy konfiguracji modułu oraz nawiązywania połączenia z siecią. Zamiast użycia zwykłego opóźnienia HAL_DELAY w celu oczekiwania na wiadomość potwierdzającą z modułu LoRa, wykorzystaliśmy funkcje HAL_UART_RxCpltCallback oraz przerwania systemowe. Dzięki czemu po otrzymaniu odpowiedniej wiadomości potwierdzającej, możemy od razu wysłać kolejne polecenie AT. Pozwoliło nam to uzyskać łączny czas pierwszego uruchomienia urządzenia, nawiązania połączenia oraz wysłania wiadomości - ok. 10 s, a każda kolejna próba trwała już tylko ok. 2 s.

```
19:08:20.242 -> +LOWPOWER: AUTOOFF
19:08:20.337 -> +JOIN: Start
19:08:20.337 -> +JOIN: NORMAL
19:08:29.631 -> +JOIN: Network joined
19:08:29.677 -> +JOIN: NetID 000013 DevAddr 26:0B:E6:B3
19:08:29.723 -> +JOIN: Done
19:08:29.817 -> +MSG: Start
19:08:30.800 -> Going to sleep now...
19:08:30.800 -> !!!!!+LOWPOWER: AUTOON
```

Rysunek 8. Wysyłanie wiadomości po pierwszym uruchomieniu urządzenia

```

19:07:01.779 -> +LOWPOWER: AUTOOFF
19:07:01.826 -> +JOIN: Joined already
19:07:01.920 -> +MSG: Start
19:07:02.904 -> Going to sleep now...
19:07:02.951 -> !!!!!+LOWPOWER: AUTOON

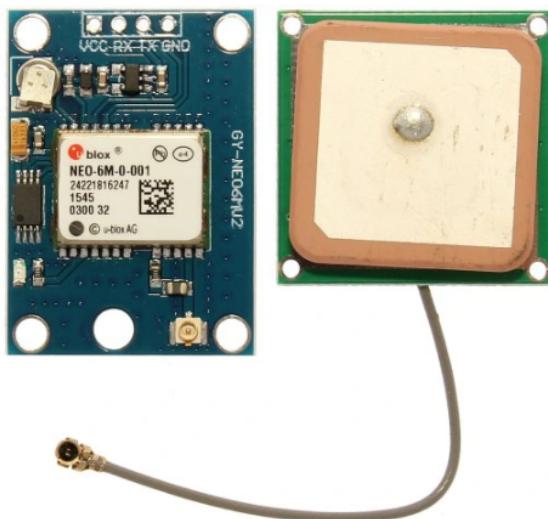
```

Rysunek 9. Wysyłanie wiadomości po wybudzeniu urządzenia

Podczas usypiania mikrokontrolera moduł LoRa przełączany jest w tryb *extremely-low-power*.

6.1.4. Moduł GPS

Główną cechą naszego urządzenia jest możliwość śledzenia pozycji bydła. W tym celu posłużyliśmy się ogólnodostępnym systemem satelitarnym - GPS. Z racji, iż postawiliśmy na energooszczędność, zdecydowaliśmy się na wybór małego, ekonomicznego modułu **GY-NEO6MV2**.



Rysunek 10. Zdjęcie modułu **GY-NEO6MV2**

Dzięki regulatorowi napięcia jest on w stanie pracować na napięciach z zakresu 3.3-5V. Dodatkowo posiada on wbudowaną baterię litową **MS621FE-FL11E** oraz pamięć wewnętrzną zapisującą parametry konfiguracji jak i ostatnio odebrane współrzędne [4]. Stąd ciepły start zajmuje mniej niż 3 sekundy. Pierwszy zimny start trwa ok. 27 sekund. Moduł komunikuje się za pomocą interfejsu UART, przesyłając wiadomości protokołu *NMEA*

```

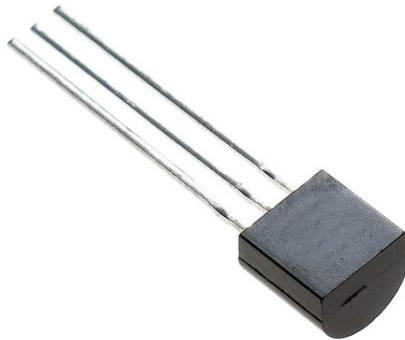
23:27:39.807 -> GPRMC,212740.00,A,5212.51370,N,02100.64687,E,2.303,,260522,,,A*7D
23:27:39.901 -> $GPVTG,,T,,M,2.303,N,4.266,K,A*27
23:27:39.948 -> $GPGGA,212740.00,5212.51370,N,02100.64687,E,1,07,1.25,117.4,M,34.3,M,,*54
23:27:39.995 -> $GPGSA,A,3,09,04,06,03,11,07,16,,,,,2.32,1.25,1.96*01
23:27:40.088 -> $GPGSV,3,1,12,02,32,298,15,03,17,131,10,04,47,075,15,06,41,239,15*75
23:27:40.135 -> $GPGSV,3,2,12,07,42,191,25,09,83,059,11,11,36,295,09,16,22,071,10*78
23:27:40.229 -> $GPGSV,3,3,12,20,21,306,15,26,16,040,,29,05,352,,30,16,205,*74
23:27:40.276 -> $GPGLL,5212.51370,N,02100.64687,E,212740.00,A,A*67

```

Rysunek 11. Przykładowa wiadomość odebrana przez nasz czujnik

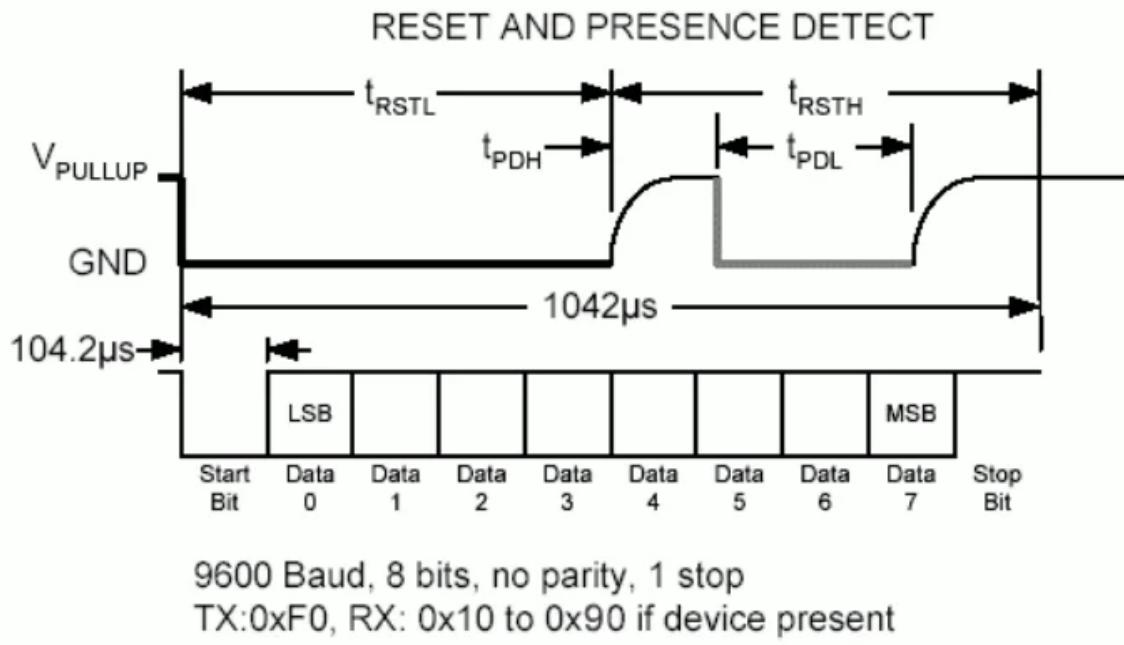
6.1.5. Czujnik temperatury

W celu sprawdzenia stanu bydła jak i potwierdzeniu, iż na zwierzętach znajdują się nasze czujniki - postanowiliśmy zamontować czujnik temperatury. Zdecydowaliśmy się na powszechny, cyfrowy model **DS18B20**.



Rysunek 12. Zdjęcie czujnika temperatury DS18B20

Jest on zasilany napięciem z zakresu od 3 do 5V. Dokładność pomiaru temperatury opisana w dokumentacji wskazuje na 0.5°C [5]. Domyślnym sposobem komunikacji jest wykorzystanie protokołu *1-wire*, jednakże wymaga on dokładnej, nieprzerwanej synchronizacji czasowej z naszym mikrokontrolerem. Z tego względu postanowiliśmy wykorzystać interfejs **UART**, tak aby symulował on komunikację poprzez *1-wire*. Jest to możliwe poprzez odpowiednie sterowanie *baud rate* oraz wysyłanie wiadomości z odpowiednimi znakami w konkretnej szczelinie czasowej.

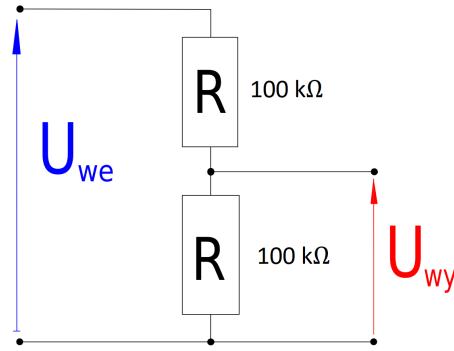


Rysunek 13. Przykładowe przedstawienie symulacji wiadomości reset poprzez UART

6.1.6. Bateria

Nasz czujnik wykorzystuje zasilanie akumulatorowe typu LiPo o pojemności 1100 mAh, dlatego do pomiarów wykonywanych przez urządzenie został zaimplementowany również pomiar napięcia baterii. Na podstawie tego napięcia możemy przesyłać informacje o poziomie naładowania baterii i ostrzec użytkownika, kiedy bateria jest bliska wyczerpaniu.

Wiedząc, że wejście przetwornika analogowo-cyfrowego w układzie przyjmuje napięcie z zakresu od 0 V do 3,3 V, a wykorzystywane ogniwo może dostarczyć napięcie nawet do 4,2 V, wykorzystaliśmy dzielnik napięcia:



Rysunek 14. Dzielnik napięcia

Stąd maksymalne napięcie jakie dostajemy na wejściu wynosi teraz ok. 2,1 V. Wykorzystując rezystory o dość dużej rezystancji uzyskaliśmy niewielkie straty wynikające z prawa Ohma: $I = \frac{U}{R} = \frac{4,2V}{200k\Omega} = 21\mu A$.

Aby zwiększyć dokładność pomiaru napięcia [6] ustawiliśmy największy czas próbkowania (640,5 cykłów) oraz włączliśmy Oversampling, czyli wykonywanie wielu pomiarów oraz wyliczenie z nich średniej wartości. Taktowanie zegara ADC zostało ustawione na 64 MHz.

ADC_Regular_ConversionMode	
Enable Regular Conversions	Enable
Enable Regular Oversampling	Enable
Oversampling Right Shift	8 bit shift for oversampling
Oversampling Ratio	Oversampling ratio 256x
Regular Oversampling Mode	Oversampling Continued Mode
Triggered Regular Oversampling	Single trigger for all oversampled conversio...
Number Of Conversion	1
External Trigger Conversion Sour...	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 1
Sampling Time	640.5 Cycles
Offset Number	No offset

Rysunek 15. Ustawienia ADC1

Baterie Li-po czy Li-ion mogą przyjąć maksymalne napięcie ok. 4,2 V. Jako bezpieczne minimalne napięcie przyjęliśmy 3,2 V. Na podstawie tych informacji możemy oszacować aktualny poziom naładowania baterii. Jednak trzeba wiedzieć, że zależność napięcia od poziomu naładowania baterii nie jest liniowa. Stąd aktualny poziom baterii odczytujemy na podstawie poniższej tablicy:

```

_vs[0] = 3.200;
_vs[1] = 3.250; _vs[2] = 3.300; _vs[3] = 3.350; _vs[4] = 3.400; _vs[5] = 3.450;
_vs[6] = 3.500; _vs[7] = 3.550; _vs[8] = 3.600; _vs[9] = 3.650; _vs[10] = 3.700;
_vs[11] = 3.703; _vs[12] = 3.706; _vs[13] = 3.710; _vs[14] = 3.713; _vs[15] = 3.716;
_vs[16] = 3.719; _vs[17] = 3.723; _vs[18] = 3.726; _vs[19] = 3.729; _vs[20] = 3.732;
_vs[21] = 3.735; _vs[22] = 3.739; _vs[23] = 3.742; _vs[24] = 3.745; _vs[25] = 3.748;
_vs[26] = 3.752; _vs[27] = 3.755; _vs[28] = 3.758; _vs[29] = 3.761; _vs[30] = 3.765;
_vs[31] = 3.768; _vs[32] = 3.771; _vs[33] = 3.774; _vs[34] = 3.777; _vs[35] = 3.781;
_vs[36] = 3.784; _vs[37] = 3.787; _vs[38] = 3.790; _vs[39] = 3.794; _vs[40] = 3.797;
_vs[41] = 3.800; _vs[42] = 3.805; _vs[43] = 3.811; _vs[44] = 3.816; _vs[45] = 3.821;
_vs[46] = 3.826; _vs[47] = 3.832; _vs[48] = 3.837; _vs[49] = 3.842; _vs[50] = 3.847;
_vs[51] = 3.853; _vs[52] = 3.858; _vs[53] = 3.863; _vs[54] = 3.868; _vs[55] = 3.874;
_vs[56] = 3.879; _vs[57] = 3.884; _vs[58] = 3.889; _vs[59] = 3.895; _vs[60] = 3.900;
_vs[61] = 3.906; _vs[62] = 3.911; _vs[63] = 3.917; _vs[64] = 3.922; _vs[65] = 3.928;
_vs[66] = 3.933; _vs[67] = 3.939; _vs[68] = 3.944; _vs[69] = 3.950; _vs[70] = 3.956;
_vs[71] = 3.961; _vs[72] = 3.967; _vs[73] = 3.972; _vs[74] = 3.978; _vs[75] = 3.983;
_vs[76] = 3.989; _vs[77] = 3.994; _vs[78] = 4.000; _vs[79] = 4.008; _vs[80] = 4.015;
_vs[81] = 4.023; _vs[82] = 4.031; _vs[83] = 4.038; _vs[84] = 4.046; _vs[85] = 4.054;
_vs[86] = 4.062; _vs[87] = 4.069; _vs[88] = 4.077; _vs[89] = 4.085; _vs[90] = 4.092;
_vs[91] = 4.100; _vs[92] = 4.111; _vs[93] = 4.122; _vs[94] = 4.133; _vs[95] = 4.144;
_vs[96] = 4.156; _vs[97] = 4.167; _vs[98] = 4.178; _vs[99] = 4.189; _vs[100] = 4.200;

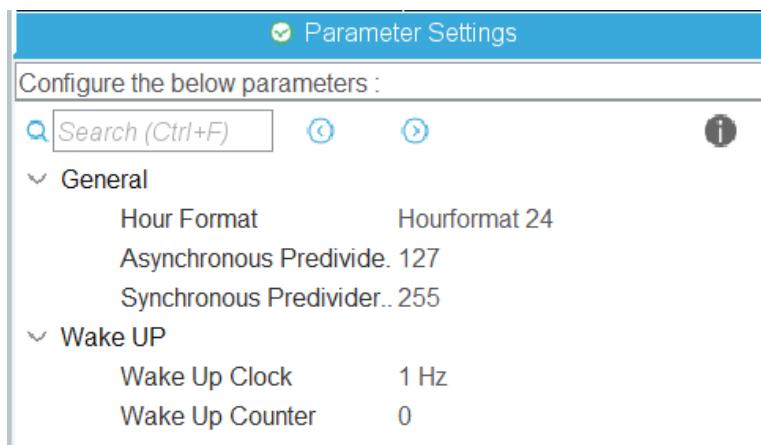
```

Rysunek 16. Zależność poziomu naładowania od napięcia baterii

Optymalizacja czasu działania na baterii

W celu maksymalnej optymalizacji czasu działania na baterii wprowadziliśmy następujące zmiany [7]:

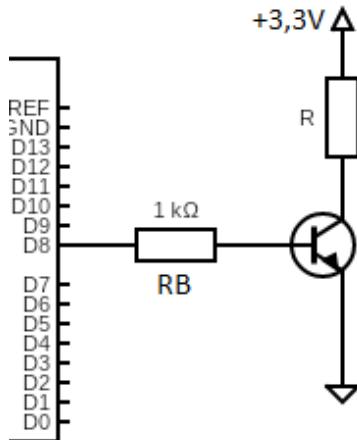
1. Obniżyliśmy częstotliwość taktowania mikrokontrolera STM32 (HCLK) na 4 MHz
2. Zmodyfikowaliśmy funkcje HAL_Delay, tak żeby mikrokontroler przechodził w tryb uśpienia i wybudzał się co 1 ms
3. Załączyczyliśmy tylko wykorzystywane peryferia
4. Zoptymalizowaliśmy czas pracy urządzenia, tak żeby trwał jak najkrócej. W naszym przypadku normalny cykl pracy trwa ok. 4s
5. Pomiary oraz wysyłanie danych wykonywane są okresowo co 5 minut



Rysunek 17. Ustawienia parametrów zegara RTC

Aby uśpić mikrokontroler na dłuższy czas (w naszym przypadku 5 min) wykorzystaliśmy zegar RTC o taktowaniu 1 Hz.

6. Po wykonaniu pomiarów i przesłaniu danych, mikrokontroler przechodzi w tryb Standby, a moduł lora w tryb extremal-low-power. Wybrany moduł GPS nie posiada żadnego trybu low-power, dlatego zdecydowaliśmy się wykorzystać klucz tranzystorowy (przełącznik).



Rysunek 18. Schemat przełącznika

Maksymalny prąd pobierany przez moduł GPS to 67 mA, dlatego tranzystor (2N2222A) sprawdzi się w naszym przypadku, ponieważ może przepuścić przez kolektor maksymalny prąd - 600 mA.

Według noty katalogowej [8] minimalny współczynnik wzmacnienia prądowego wynosi 100, stąd:

$$I_B = \frac{I_C}{\beta} = \frac{67mA}{100} = 0,67mA \quad | \quad R_B = \frac{U_H - U_{BE}}{I_B} = \frac{3,3V - 0,7V}{0,67mA} = 3,88k\Omega$$

Aby tranzystor nie wyszedł ze stanu nasycenia przyjąłem mniejszą wartość rezystancji bazy - 1 kΩ

Podając wysoki stan na wejście bazy, tranzystor przechodzi w stan nasycenia, zatem jest w pełni załączony (moduł GPS jest włączony). Po uśpieniu mikrokontrolera, do bazy tranzystora podawany jest niski stan, co wprowadza go w stan zatkania, tranzystor jest w pełni wyłączony (moduł GPS jest wyłączony, nie pobiera żadnego prądu).

Testy baterii

Aby zmierzyć całkowity prąd pobierany przez urządzenie w czasie normalnej pracy i uśpienia, wszystkie podłączone moduły oraz mikrokontroler STM32 zostały zmierzone oddzielnie. Budując czujnik na prototypowej płytce Nucleo-L47RG i zasilaniu jej z modułu DFR0559 napięciem 5V, nie mogliśmy zmierzyć na raz całego pobieranego prądu, ponieważ zauważaliśmy zwiększyły pobór prądu przez zastosowanie nieodpowiednich modułów. Płytki Nucleo posiada wbudowany programator ST-LINK, który jest zawsze podpięty pod zasilanie. Kolejnym problemem był moduł odpowiadający za ładowanie baterii (DFR0559), który miał wyjściowe napięcie 5V przez co występowały tu kolejne straty przez niepotrzebną konwersję napięcia. Najlepiej byłoby zasilić mikrokontroler bezpośrednio napięciem 3,3 V (odcinając również górną część płytki Nucleo z programatorem) oraz wykorzystać inny moduł ładowania baterii (np. TP4056) z regulatorem napięcia LDO 3,3 V (np. MCP1700-3302E/TO - idealny do działania z wykorzystaniem baterii, przez bardzo małą wartość prądu roboczego).

	Normalna praca	Obciążenie	Uśpienie
	Pobór prądu	Pobór prądu	Pobór prądu
Moduł LORA	7,65 [mA]	120 [mA]	48 [μ A]
Dzielnik napięcia	21 [μ A]	-	21 [μ A]
STM32	1 [mA]	-	0,8 [μ A]
Moduł GPS	55 [mA]	-	0
Termometr	1 [μ A]	1,5 [mA]	1 [μ A]
Przełącznik BJT	-	2 [mA]	0

Rysunek 19. Zmierzony pobór prądu przez poszczególne moduły

Lp.	Opis	Czas [s]	Średni pobór prądu [mA]
1	Dołączenie do sieci lora	0,1	8,67
2	Lokalizowanie	2	65,67
3	Pomiar baterii i temperatury	0,15	10,17
5	Wysłanie wiadomości i uśpienie	2	121,02
		Średnia	88,42
		Czas pracy [s]	4,25

Rysunek 20. Oszacowanie średniego poboru prądu podczas 2 i kolejnego cyklu pracy

Pojemność baterii [mAh]	1100
Średni prąd podczas uśpienia [mA]	0,0708
Średni prąd podczas pracy [mA]	88,42
Liczba cykłów pracy na godzinę	12
Czas pracy [s]	4,25
Łączny czas pracy na godzinę [s]	51
Łączny czas uśpienia na godzinę [s]	3549
Średni pobór prądu na godzinę [mA]	1,32
Przewidywany czas w dniach	34,7
Pobór prądu na rok [mA]	11584
Pobór prądu na dzień [mA]	32
Pobór prądu na dzień [Wh]	0,117

Rysunek 21. Szacowany czas pracy urządzenia

Doładowywanie baterii za pomocą panelu słonecznego

Z wykonanych obliczeń wychodzi nam ok. 34 dni pracy na samej baterii o pojemności 1100mAh. Dla farmerów o dużej liczbie zwierząt hodowlanych mogłoby być dość uciążliwe wymienianie baterii w każdym czujniku, dlatego postanowiliśmy rozbudować urządzenie o ładowanie za pomocą paneli słonecznych.



Rysunek 22. Panel słoneczny

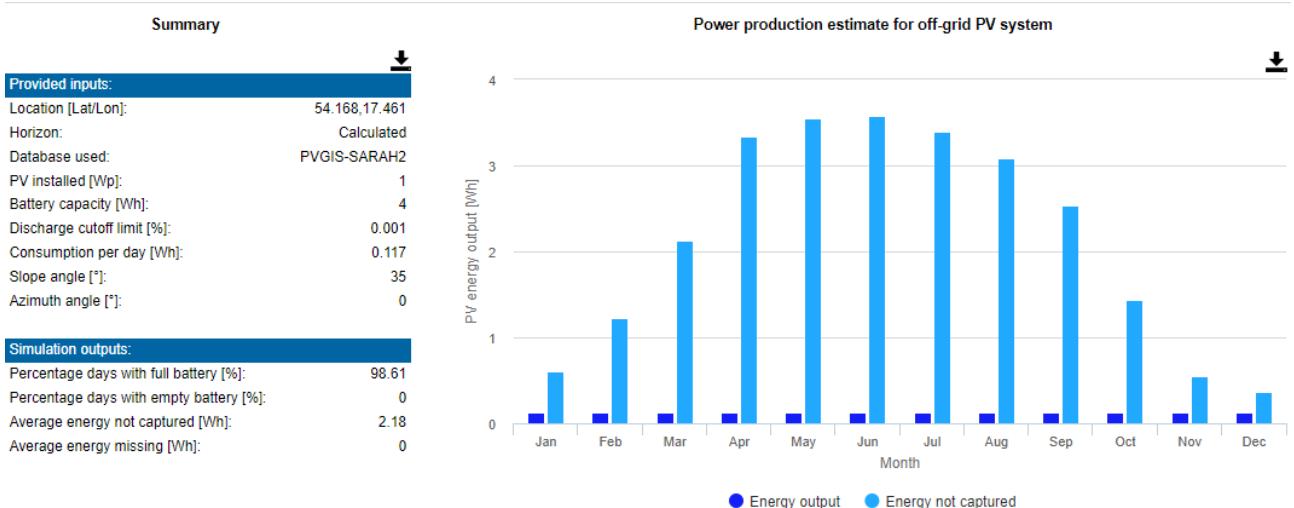
Wykorzystywany panel w naszym urządzeniu:

- Wymiary: 136 x 110 x 3 mm
- Moc: 1 W
- Napięcie wyjściowe: 6 V

Do oszacowania produkcji energii skorzystaliśmy ze strony PVGIS [9], która dostarcza informacje o promieniowaniu słonecznym i wydajności systemów fotowoltaicznych dla dowolnej lokalizacji w Europie i Afryce, a także dużej części Azji i Ameryki oraz umożliwia przeprowadzenie symulacji.

Przyjęte ustawienia w symulacji:

- Moc panelu słonecznego: 1 W
- Pojemność baterii: 4 Wh (3,7 V * 1100 mAh)
- Limit odcięcia rozładowania: 0.001 %
- Zużycie energii na dzień: 0,117 Wh (3,7 V * 24h * 1,32 mA)
- Lokalizacja : województwo pomorskie (najmniejsze natężenie promieniowania słonecznego w Polsce)



Rysunek 23. Oszacowana produkcja energii

Z powyższego wykresu można zauważyć, że wybrany panel o mocy 1 W nie przechwyci całej możliwej energii, ponieważ zabezpieczenie modułu ładowania nie będzie pozwalało przeładować ogniw. Według przeprowadzonej symulacji przez 98,61% dni bateria będzie w pełni naładowana oraz urządzenie zawsze będzie działało.

Symulacja opiera się na średnim zysku energii na miesiąc w wybranym miejscu, kiedy panel w ciągu dnia wystawiony jest na słońce na możliwie długi czas, pomija straty podczas ładowania oraz takie sytuacje, kiedy na przykład wiele dni jest deszczowych i pochmurnych, gdzie natężenie słoneczne jest dużo poniżej średniej. Jak wiadomo zwierzęta mogą być "pod dachem", więc czas na słońcu może być krótszy niż według tej symulacji. Wszystkie obliczenia są dość uproszczone, jednak nasze urządzenie będzie potrafiło przetrwać bez ładowania około 1 miesiąca, co jest w zupełni wystarczające. Stąd nasz czujnik nie będzie wymagał wymiany baterii czy jej doładowania.

Najlepiej by było jeszcze przeprowadzić testy praktyczne, które potwierdziłyby powyższe kalkulacje. Gdyby jednak bateria nie wystarczała w okresie zimowym (grudzień - luty), kiedy jest najmniej słońca, można byłoby spokojnie wymienić baterię na taką z większą pojemnością, np 2000-3000 mAh.

6.2. Część software'owa

6.2.1. Założenia, implementowana funkcjonalność aplikacji mobilnej

Na podstawie wcześniejszej przeprowadzonych analiz, zdecydowaliśmy się, iż aplikacja mobilna będzie składała się z 3 ekranów:

- **Map:** ekran z mapą, na której użytkownik będzie mógł zobaczyć swoją własną lokalizację, lokalizację obserwowanych zwierząt oraz będzie w stanie tworzyć ścieżki, prowadzone do każdego ze zwierząt
- **Animals:** ekran, na którym będzie możliwy podgląd stanu urządzenia, znajdującego się na zwierzęciu (poziom naładowania baterii), stan środowiska, otaczającego zwierzę (temperatura środowiska) oraz będzie możliwy podgląd trasy, którą dokonało zwierzę od czasu wejścia użytkownika do aplikacji
- **Account:** ekranu, na którym będzie możliwy podgląd imienia i nazwiska użytkownika oraz nadania uprawnień aplikacji do wykorzystania geolokalizacji użytkownika oraz wysyłania powiadomień.

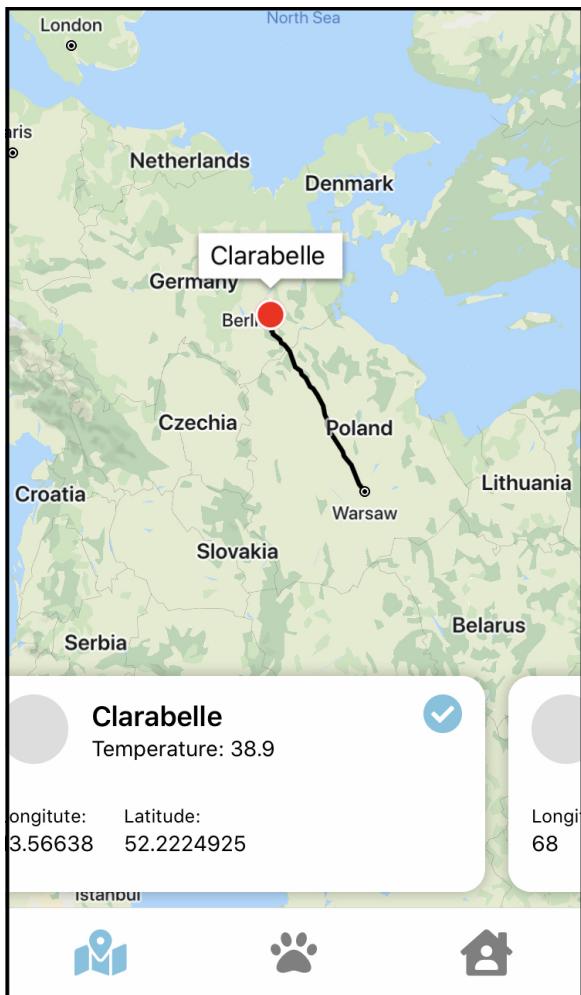
6.2.2. Aplikacja mobilna

Aplikacja mobilna została zaimplementowana przy pomocy framework'u React Native, charakteryzującego się tym, że umożliwia on tworzenie aplikacji natywnych, czyli takich, z których można korzystać na dowolnych urządzeniach (mobilnych i stacjonarnych) [10]. Framework React Native pozwala na tworzenie aplikacji przy wykorzystaniu języka funkcyjnego javascript, podobnie jak i w samym React.js opiera się na tworzeniu komponentów, w sposób podobny, jak to się robi przy użyciu HTML5 i CSS, i operowaniu na ich stanach w sposób dynamiczny.

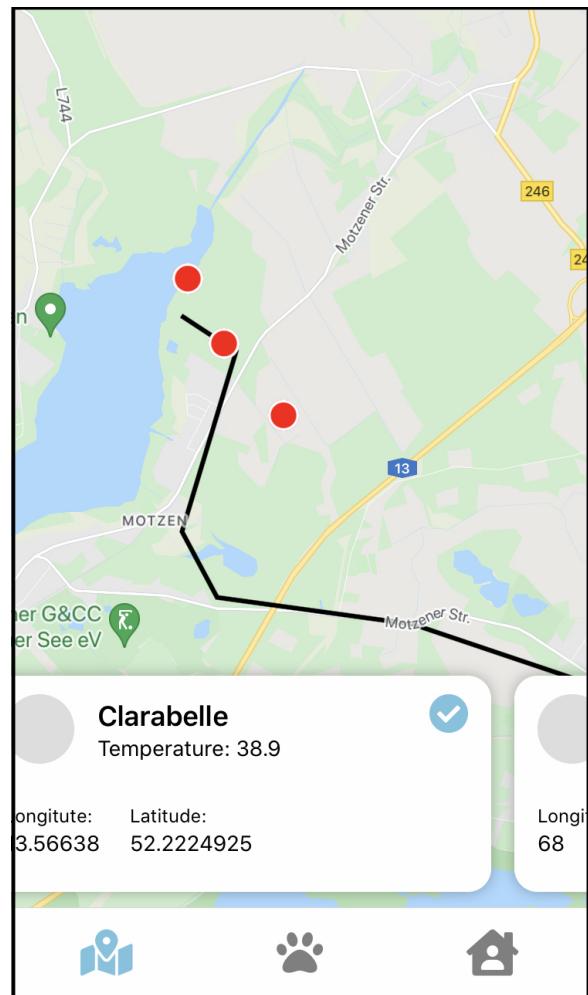
Aby się udało zaimplementować wszystkie wcześniej wspomniane funkcje, skorzystaliśmy między innymi z takich bibliotek pomocniczych, jak:

- **expo**, będącej zestawem narzędzi stworzonych dla React Native, które pomagają w bardzo szybkim uruchomieniu aplikacji, zawiera ona listę narzędzi, które upraszczają tworzenie i testowanie aplikacji React Native
- **react-redux**, będącej zestawem narzędzi, umożliwiających tworzenie, obserwowanie oraz zmianę danych, przechowywanych wewnętrz aplikacji i pozwalającej na ich wykorzystywanie na dowolnym poziomie zagłędzienia komponentów, z których się składa aplikacja
- **firebase**, będącej zestawem narzędzi umożliwiającym komunikację aplikacji React Native z takim narzędziem jak firebase (w naszej aplikacji wykorzystywanym do przechowywania danych, pobieranych z czujników)
- **react-native-maps**, pomagającej na integrację usługi Google Maps z naszą aplikacją, czyli pozwalającej wykorzystanie jej map i dopełnień do nich (przykładowo, używanie pinów)
- **react-native-maps-directions**, pozwalającej na tworzenie ścieżek między dwoma punktami, zgodnych z istniejącymi ścieżkami znany Google Maps
- **react-native-geolocation-service**, umożliwiającej pobieranie aktualnej geolokalizacji użytkownika (po jego wcześniejszej zgodzie na to)

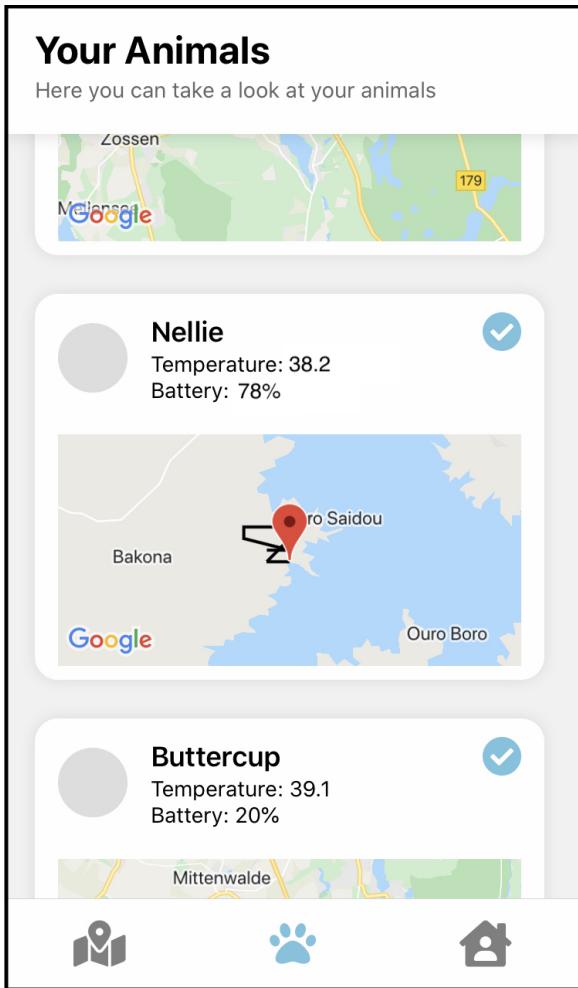
Po zaimplementowaniu wszystkich wspomnianych funkcji, nasza aplikacja wyglądała w sposób pokazany na Rysunkach 12-15. Przy budowaniu ścieżek wykorzystywane były lokalizacja docelowa (lokalizacja zwierzęcia) oraz lokalizacja użytkownika (udostępniana geolokalizacja).



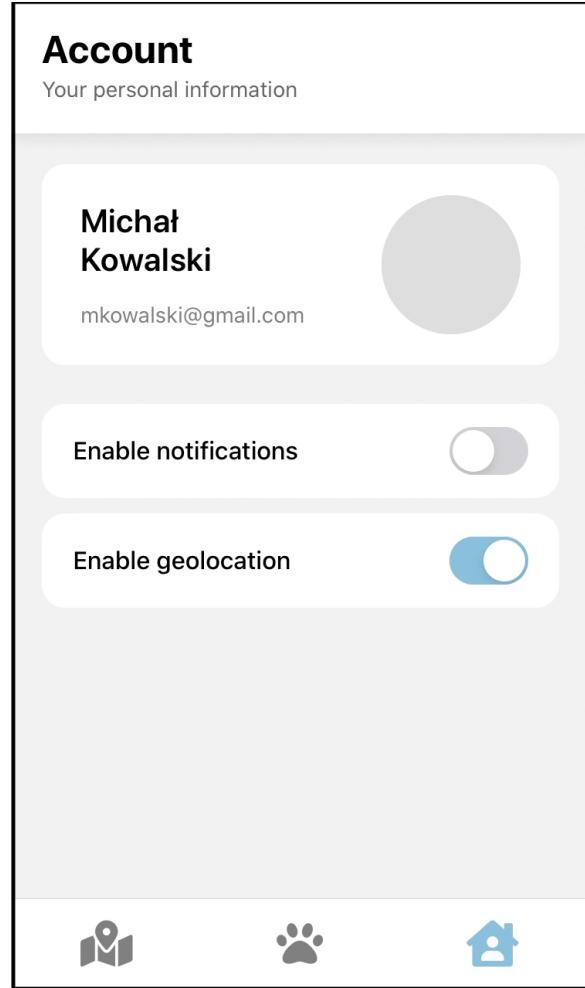
Rysunek 24. Ekran "Map", wersja oddalone



Rysunek 25. Ekran "Map", wersja przyblizona



Rysunek 26. Ekran "Animals"



Rysunek 27. Ekran "Account"

6.2.3. Konfiguracja bazy danych

Do przechowywania danych, wysyłanych w czasie rzeczywistym, skorzystaliśmy z dość popularnego narzędzia, czyli **firebase**. Ten system wybraliśmy z dwóch powodów [11]:

1. Firebase jest platformą, opracowaną przez Google, co jest dość ważnym aspektem w naszym przypadku, jako że również korzystamy z Google Maps. Dla obu tych narzędzi musimy posiadać konto Google i odpowiedni rodzaj subskrypcji do korzystania z usług Google Cloud Platform. Wykorzystując tylko i wyłącznie produkty wspierane przez Google pozbywamy się konieczności konfiguracji kluczy API i zarządzania narzędziami dla więcej niż jednej platformy.
2. Firebase oferuje możliwość do korzystania z real-time-database, gdzie dane mogą w czasie rzeczywistym nadpisywane, co idealnie pasuje do naszego use case'u.
3. Narzędzie NodeRED posiada moduły, przeznaczone specjalnie do komunikacji poprzez http, co jest wystarczającym, dla zapewnienia łączności tym narzędziem, a firebase-realtime-database.

Zatem najpierw po stronie platformy (dostępnej pod linkiem <https://console.firebaseio.google.com>), skonfigurowaliśmy naszą bazę danych i inne narzędzia (niezbędnie, zbyt udało się podłączyć bazę danych do naszej aplikacji mobilnej). Podgląd do skonfigurowanej bazy danych z wykorzystywaną strukturą danych, został umieszczony na Rysunku 16.

The screenshot shows the Firebase Realtime Database interface. At the top, there are navigation links for 'Data', 'Rules', 'Backups', and 'Usage'. On the right, there are links for 'Go to docs', a user profile icon, and a help icon. Below the header, the URL is https://fluent-limiter-349320-default-rtbd.firebaseio.europe-west1.firebaseio.app/. The main area displays a hierarchical database structure:

```

https://fluent-limiter-349320-default-rtbd.firebaseio.europe-west1.firebaseio.app/
  └── 0
    ├── battery: 30
    ├── latitude: 52.2224925
    ├── longitude: 13.56638
    ├── temperature: 38.9
    └── timestamp: 1651328585517
  └── 1
    ├── battery: 0
    ├── latitude: 8.98432
    ├── longitude: 13.56438
    └── temperature: 1.105371

```

A note at the bottom indicates the database location: Belgium (europe-west1).

Rysunek 28. Podgląd do zakładki konsoli platformy firebase, w której została skonfigurowana nasza baza danych

6.2.4. Połaczanie aplikacji mobilnej z bazą danych

Aby się udało połączyć naszą aplikację z bazą danych firebase'ową, musieliśmy wewnątrz naszego projektu z kodem źródłowym do aplikacji mobilnej stworzyć odpowiedni plik, zawierający wszystkie identyfikatory, pozwalające na uwierzytelnienie i rozpoznanie aplikacji przez Google Cloud Platform.

Zawartość takiego pliku została zwizualizowana na Rysunku 17 (wartości wskazanych pól zostały wymazane z powodów prawnych).

```

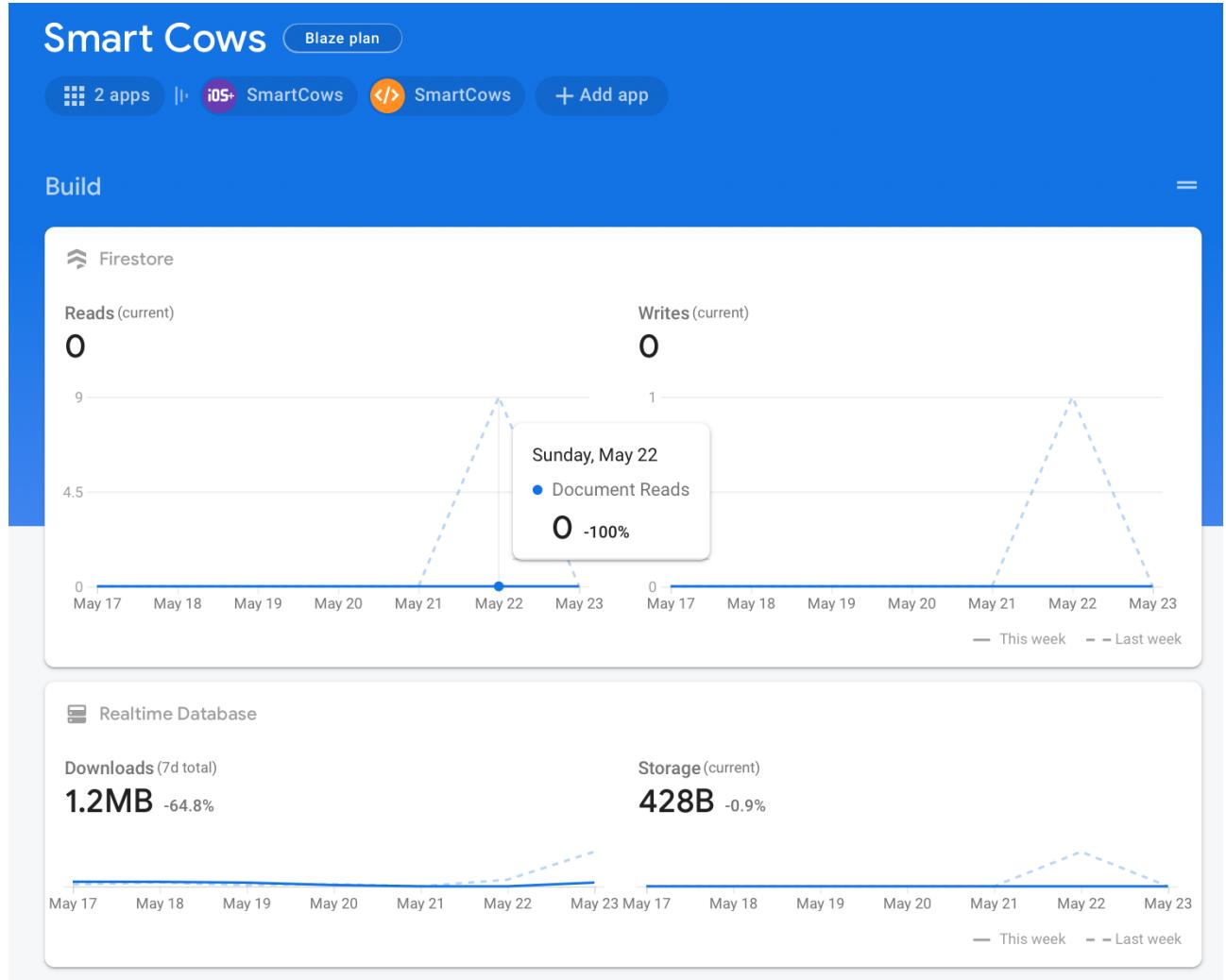
1 import { initializeApp } from "@firebase/app";
2
3 const firebaseConfig = {
4   apiKey: [REDACTED],
5   authDomain: [REDACTED],
6   databaseURL: [REDACTED],
7   projectId: [REDACTED],
8   storageBucket: [REDACTED],
9   messagingSenderId: [REDACTED],
10  appId: [REDACTED],
11  measurementId: [REDACTED]
12};
13
14 export const app = initializeApp(firebaseConfig);

```

Rysunek 29. Podgląd do zakładki konsoli platformy firebase, w której została skonfigurowana nasza baza danych

Po skonfigurowaniu wszystkiego, możemy podejrzeć na stronie ze statystykami, czy odbywa się

przesył danych i jeżeli tak, to jaki jest ich napływy. Przykładową statystykę można zobaczyć na Rysunku 18.



Rysunek 30. Podgląd do zakładki konsoli platformy firebase, w której możemy zaobserwować statystyki wykorzystywanych narzędzi

6.2.5. Połączenie bazy danych z TTN'em

Połączenie między bazą danych, a siecią The Things Network, do której zostały podłączone nasze czujniki, odbywało się przy wykorzystaniu narzędzia NodeRED.

Narzędzie NodeRED pozwala na tworzenie tzw. WebHooków w sposób uproszczony - poprzez wykorzystanie odpowiednich bloków funkcjonalnych. Blokami funkcjonalnymi, z których skorzystaliśmy w naszym projekcie, były:

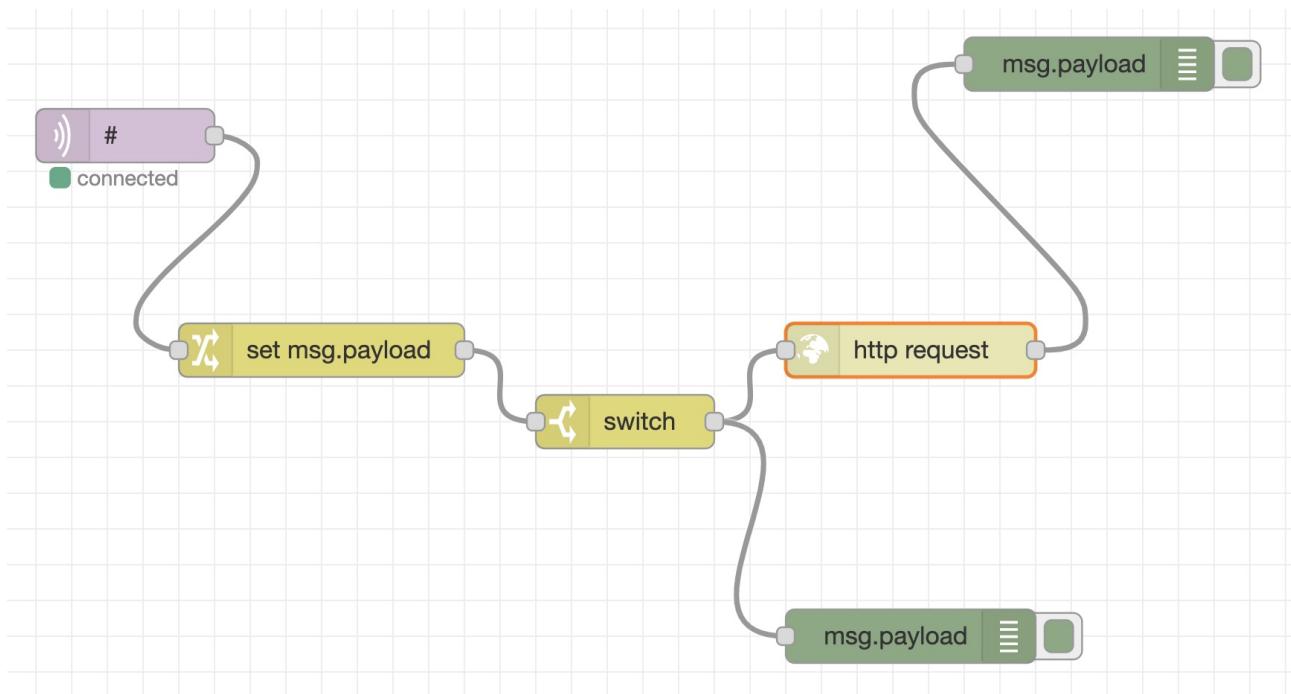
- **mqtt-in**, pełniącego funkcję klienta MQTT, poprzez subskrybowanie odpowiedniego topic'a udostępnionego przez blockera MQTT sieci TTN
- **change**, pozwalającego na modyfikację danych odbieranych w sposób, potrzebny użytkownikowi
- **switch**, do aplikowania warunków, przy spełnieniu którego dane są przepuszczane do odpowiednich wyjść tego bloku

- **http-request**, pozwalającego na wysyłanie http request'ów
- **debug**, pozwalającego na wypisywanie na potrzeby debug'u utworzonego systemu

Komunikacja odbywała się w następujący sposób:

1. Dane są pobierane poprzez korzystanie z bloku funkcjonalnego **mqtt-in**
2. Dane przekazywane z bloku **mqtt-in** do bloku **change** i podmieniane w taki sposób, aby zawierały one tylko potrzebny nam payload
3. Dane przekazywane z bloku **change** do bloku **switch**, gdzie jest sprawdzany warunek obecności pola timestamp w payload'zie (który świadczy o tym, że przechwytywana wiadomość nie jest wiadomością przeznaczoną do przekazywania dodatkowych parametrów, tylko zawiera oczekiwane przez nas dane, wysyłane przez czujnik)
4. Jeżeli warunek został spełniony, to dane przekazywane są do bloku **http-request**, a jeżeli nie spełnia - to je odrzucamy
5. Blok **http-request** wysyła dane na endpoint, przeznaczony do pobierania danych przez naszą bazę danych

W taki sposób jest zapewniana komunikacja między TTN'em, a naszą bazą danych, skonfigurowaną w konsoli firebase'u. Flow, wykorzystane w celu zapewnienia tej komunikacji, jest pokazany na Rysunku 19.



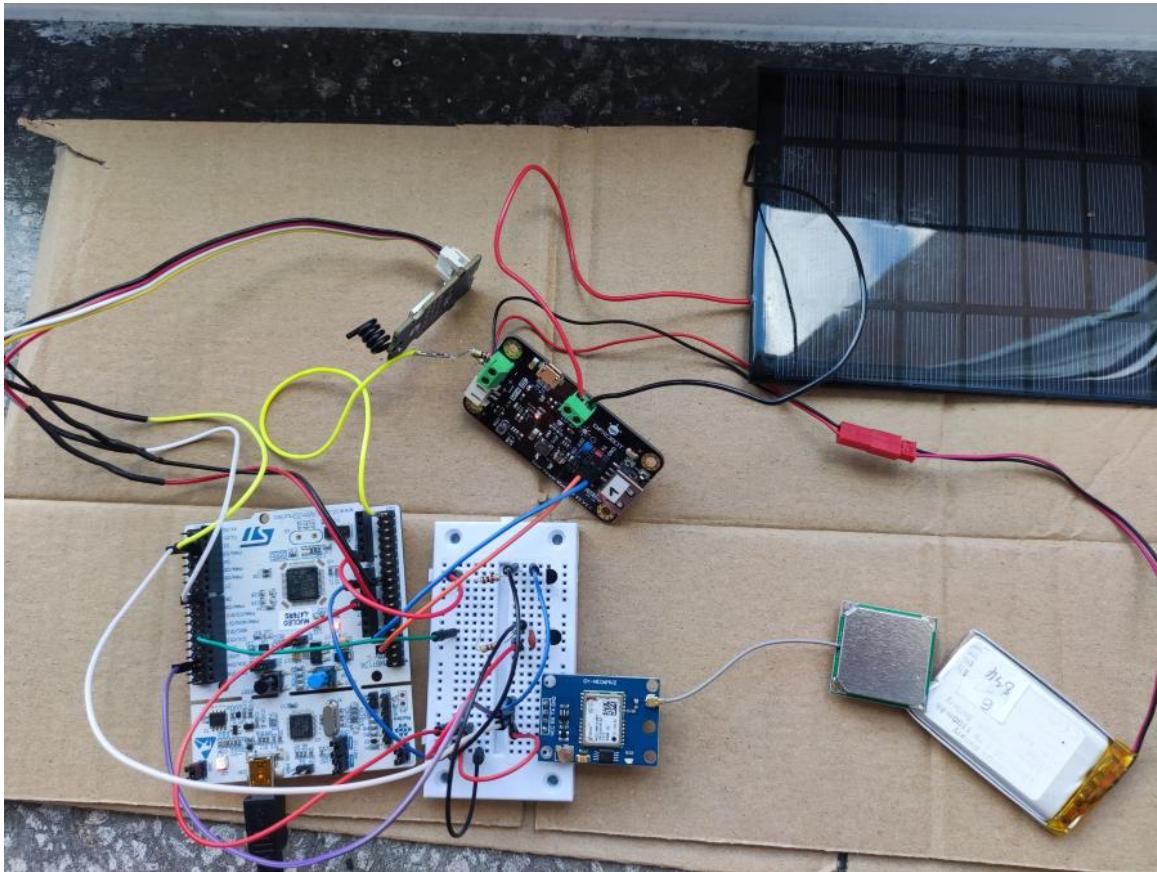
Rysunek 31. Wykorzystywany Flow w narzędziu NodeRED

6.3. Kod źródłowy

Link do kodu źródłowego: <https://github.com/maizyy/smart-cows>.

7. Prototyp

W efekcie końcowym mamy przetestowany, działający system umożliwiający odczytywanie położenia naszych krów, ich temperatury oraz stanu naładowania baterii każdego z czujników. Wszystko za pomocą aplikacji mobilnej.



Rysunek 32. Prototyp naszego produktu bez obudowy

```
23:27:39.807 -> +JOIN: Joined already
23:27:39.807 -> Start GPS
23:27:39.807 ->
23:27:39.807 -> GPRMC,212740.00,A,5212.51370,N,02100.64687,E,2.303,,260522,,,A*7D
23:27:39.901 -> $GPVTG,T,,M,2.303,N,4.266,K,A*27
23:27:39.948 -> $GPGGA,212740.00,5212.51370,N,02100.64687,E,1,07,1.25,117.4,M,34.3,M,,*54
23:27:39.995 -> $GPGSA,A,3,09,04,06,03,11,07,16,,,,,2.32,1.25,1.96*01
23:27:40.088 -> $GPGSV,3,1,12,02,32,298,15,03,17,131,10,04,47,075,15,06,41,239,15*75
23:27:40.135 -> $GPGSV,3,2,12,07,42,191,25,09,83,059,11,11,36,295,09,16,22,071,10*78
23:27:40.229 -> $GPGSV,3,3,12,20,21,306,15,26,16,040,,29,05,352,,30,16,205,*74
23:27:40.276 -> $GPGLL,5212.51370,N,02100.64687,E,212740.00,A,A*67
23:27:40.325 -> Koniec GPS
23:27:40.370 -> T1 = 23.9*C
23:27:40.464 -> +MSG: Start
23:27:41.963 -> !!!!!+LOWPOWER: AUTOON
23:27:41.963 -> STANDBY MODE is ON
```

Rysunek 33. Cykl działania naszego czujnika zaraz po wybudzeniu

8. Podsumowanie

Celem tego projektu było zaproponowanie rozwiązanie, które byłoby przydatne i użyteczne w rzeczywistych warunkach, panujących na świecie. Nasz zespół wybrał problem, związany z hodowaniem zwierząt i polegał na zaoferowaniu pomocy fermerom, które tym zajmują się.

Wdrożone przez nasz zespół rozwiązanie nie tylko jest w miarę tanie, ale też nie jest skomplikowane w obsłudze przez osoby trzecie. Aplikacja została zaimplementowana tak, aby korzystanie z niej było intuicyjne dla użytkownika, a sposób umieszczenia czujnika na bydle jest znany osobom go hodującego, jako że jest on tożsamy ze sposobem zakładania obroży na zwierzętach.

Proces realizacji tego projektu nie był prosty, musielibyśmy przeanalizować istniejące rozwiązania, możliwe "kombinacje" wykorzystywanych narzędzi, zdecydować się na jedno z takich i co jest najważniejsze, doprowadzić nasz produkt do takiego stanu, które umożliwiłoby przetestowanie całego systemu, a następnie zaprezentowanie jego działania. Ważnym też było podtrzymywanie komunikacji wewnętrz zespołu, aby być w stanie wykrywać możliwe problemu na wcześniejszych etapach realizacji projektu.

Jesteśmy świadomi tego, że niektóre elementy naszego systemu dałyby się zaimplementować z sposób wydajniejszy i efektywniejszy (w skrócie mówiąc "lepszy", przykładowo, zamiana wykorzystanych części czujnika na mniejsze, jako że jednak urządzenie przez nas skonstruowane jest nieco duże, niż by się chciało), jednak z powodu ograniczonego czasu poświęconego na realizację projektu, zdecydowaliśmy, że obecny stan naszego systemu jest zadowalająco dobry. Podczas realizacji tego projektu musielibyśmy niejednorazowo zagłębiać się w dokumentację wykorzystywanych narzędzi oraz spędzać czas debugując występujące błędy.

Nauczyliśmy się radzić sobie z takimi problemami jak niepoprawne połączenie kabli, brak komunikacji podzespołów czujnika, niepoprawna konfiguracja aplikacji mobilnej, braki uprawnień (w przypadku korzystanie z rozwiązań chmurowych), niedopasowanie sposobów zasilania podzespołów oraz wiele innych. Wiedza, uzyskana podczas realizacji tego projektu pewnie przyda się nam podczas realizacji innych, możliwe bardziej skomplikowanych systemów, i rozwiązania problemów które napotkamy na swojej ścieżce jako inżynierowie.

Literatura

- [1] G. S. R. M. Biljana Risteska Stojkoska, Dijana Capeska Bogatinoska, "Real-time Internet of Things Architecture for Wireless Livestock Tracking." https://journal.telfor.rs/Published/Vol10No2/Vol10No2_A2.pdf, 2018. Dostęp zdalny (21.03.2022 r.).
- [2] st.com, "Stm32l476rg - informacje ogólne i dokumentacja." <https://www.st.com/en/microcontrollers-microprocessors/stm32l476rg.html#overview>. Dostęp zdalny (12.06.2022 r.).
- [3] seedstudio.com, "Moduł radiowy lora-e5 stm32wle5jc - dokumentacja." https://files.seeedstudio.com/products/317990687/res/LoRa-E5%20module%20datasheet_V1.0.pdf. Dostęp zdalny (12.06.2022 r.).
- [4] terraelectronica.ru, "Moduł gps ne06m." https://www.terraelectronica.ru/pdf/show?pdf_file=%252Fz%252FDatasheet%252FU%252FUART%2BGPS%2BNE0-6M%2BUser%2BManual.pdf. Dostęp zdalny (12.06.2022 r.).
- [5] datasheets.maximintegrated.com, "Cyfrowy czujnik temperatury ds18b20." <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Dostęp zdalny (12.06.2022 r.).
- [6] forbot.pl, "Kurs stm32l4 #9 - przetworniki analogowo-cyfrowe (adc)." <https://forbot.pl/blog/kurs-stm32l4-przetworniki-analogowo-cyfrowe-adc-id46587>. Dostęp zdalny (12.06.2022 r.).
- [7] forbot.pl, "Kurs stm32l4 #6 - oszczędzanie energii." <https://forbot.pl/blog/kurs-stm32l4-oszczedzanie-energii-5-lat-na-baterii-id46581>. Dostęp zdalny (12.06.2022 r.).
- [8] st.com, "Tranzystor 2n2222a - nota katalogowa." <https://www.st.com/resource/en/datasheet/cd00003223.pdf>. Dostęp zdalny (12.06.2022 r.).
- [9] re.jrc.ec.europa.eu, "Pvgis." https://re.jrc.ec.europa.eu/pvg_tools/en/tools.html#api_5.2. Dostęp zdalny (12.06.2022 r.).
- [10] Facebook Open Source - Meta Platforms, Inc., "Dokumentacja do React Native." <https://pl.reactjs.org/docs/getting-started.html>. Dostęp zdalny (18.05.2022 r.).
- [11] Google Developers, "Firebase Documentation - Fundamentals." <https://firebase.google.com/docs/guides>. Dostęp zdalny (18.05.2022 r.).