



Dokumentacja techniczna projektu „LoRaptor”

Autorzy

Karol Duda
Kamil Kwiatkowski
Krzysztof Duda

Zespół Szkół Elektryczno-Mechanicznych nr 7 w Nowym Sączu

Spis treści

| | |
|---|-----------|
| 1 Wprowadzenie | 1 |
| 1.1 Czym jest LoRaptor? | 2 |
| 1.2 Cele, założenia i zastosowania | 3 |
| 1.3 Off the grid, on the mesh - co to znaczy? | 4 |
| 1.4 Przebieg pracy | 5 |
| 2 Część sprzętowa | 7 |
| 2.1 Projekt PCB i montaż komponentów | 8 |
| 2.2 Schemat elektroniczny | 9 |
| 2.3 Produkcja PCB | 12 |
| 2.4 Lista elementów (BOM) | 15 |
| 2.5 Obudowa urządzenia | 16 |
| 2.6 Złożona jednostka | 25 |
| 2.7 Szacunkowy koszt jednej jednostki | 28 |
| 3 Część programowa | 29 |
| 3.1 Biblioteka LoRaMesher | 29 |
| 3.1.1 Czym jest LoRaMesher? | 29 |
| 3.1.2 Struktura wiadomości i komunikacja | 30 |
| 3.1.3 Architektura zadań i kolejek | 31 |
| 3.1.4 Abstrakcja danych i integracja w LoRaptorze | 32 |
| 3.1.5 Format danych Payload | 32 |
| 3.1.6 Wysyłanie danych przez LoRaMesher | 33 |
| 3.1.7 Dekodowanie po stronie odbiorcy | 33 |
| 3.1.8 Korzyści projektowe | 34 |
| 3.1.9 Podsumowanie | 34 |
| 3.2 Biblioteka RaptorCLI | 35 |
| 3.2.1 Architektura i integracja | 35 |
| 3.2.2 Połączenia — czym one są? | 36 |
| 3.2.3 Przykładowe polecenia | 37 |
| 3.2.4 Definiowanie komend w kodzie | 39 |
| 3.2.5 Argumenty poleceń | 40 |
| 3.2.6 Argumenty o nieograniczonej liczbie | 41 |
| 3.2.7 Zarządzanie wyjściem i wejściem | 42 |
| 3.2.8 Podsumowanie | 42 |
| 3.3 Aplikacja RaptChat | 43 |
| 3.3.1 Architektura działania | 44 |
| 3.3.2 Komunikacja z urządzeniem | 45 |
| 3.3.3 Warstwa BLE | 45 |
| 3.3.4 Automatyzacja konfiguracji | 46 |
| 3.3.5 Warstwa wiadomości | 46 |
| 3.3.6 Interfejs użytkownika | 46 |

| | | |
|----------|--|-----------|
| 3.3.7 | Podsumowanie | 54 |
| 4 | Zalety i korzyści naszego rozwiązania | 55 |
| 5 | Konkurencyjność | 56 |
| 6 | Innowacyjność | 57 |
| 7 | Plany na przyszłość | 58 |
| 8 | Gdzie nas można znaleźć? | 59 |

1 Wprowadzenie

Współczesny świat oferuje nam niezliczone możliwości rozwoju — od łatwego dostępu do wiedzy i technologii, po liczne udogodnienia, które wspierają nas w codziennym życiu. Jednocześnie jednak, tempo tego życia nieustannie przyspiesza, stawiając przed nami nowe wyzwania. Dlatego coraz częściej odczuwamy potrzebę unowocześniania naszej rzeczywistości: czynienia jej bardziej punktualną, intuicyjną, bezpieczną i niezawodną — tak, by wspierała nas w codziennych obowiązkach i pozwalała lepiej zarządzać tym, co najcenniejsze.

„Najlepszym sposobem przewidywania przyszłości jest jej tworzenie.”

— Alan Kay

Właśnie z tej potrzeby narodził się nasz pomysł: **LoRaptor** — innowacyjny system komunikacji dalekiego zasięgu, pozwalający wspieranym urządzeniom (głównie mobilnym, m.in. smartfonom) na bezprzewodową wymianę danych bez użycia Wi-Fi czy sieci komórkowych. To odpowiedź na współczesne realia, w których niezależność, zasięg, i szybkość mają kluczowe znaczenie. Zaprojektowaliśmy rozwiązanie, które nie tylko spełnia te wymagania, ale czyni to w sposób elegancki, lekki i dostępny.

1.1 Czym jest LoRaptor?

LoRaptor to innowacyjny system komunikacji bezprzewodowej, który umożliwia wymianę danych pomiędzy urządzeniami mobilnymi — takimi jak smartfony, tablety czy komputery — bez potrzeby korzystania z Wi-Fi, Internetu ani sieci komórkowych. To rozwiązanie skierowane jest przede wszystkim do osób i środowisk, które potrzebują niezawodnej łączności na dużą odległość, bez zależności od zewnętrznej infrastruktury. System składa się z trzech głównych komponentów:

- Moduł elektroniczny, który można podłączyć bezpośrednio do urządzenia użytkownika (np. przez USB-C);
- Aplikacja mobilna, umożliwiająca intuicyjne sterowanie, wysyłanie i odbieranie wiadomości;
- Biblioteka programistyczna, pozwalająca w prosty sposób definiować i obsługiwać komendy systemowe;

Dzięki tym elementom, LoRaptor tworzy ekosystem, który działa niezależnie od infrastruktury zewnętrznej.

„To narzędzie, które pozwala pozostać w kontakcie — zawsze, wszędzie i bez ograniczeń.”

1.2 Cele, założenia i zastosowania

LoRaptor to projekt powstały z potrzeby zapewnienia niezależnej komunikacji bezprzewodowej w sytuacjach, gdzie tradycyjne sieci zawodzą — w terenie, w warunkach kryzysowych, podczas biwaków, wypraw czy gier terenowych. Naszym celem było stworzenie lekkiego, energooszczędnego urządzenia, które:

- Umożliwia komunikację bez potrzeby dostępu do Wi-Fi, GSM czy Internetu;
- Tworzy dynamiczną sieć mesh opartą na technologii LoRa;
- Działa intuicyjnie i bez konfiguracji sieciowej;
- Współpracuje z aplikacją mobilną oraz terminaliem CLI;
- Może pełnić rolę płytka deweloperskiej;

Główne zastosowania **LoRaptora**:

- Komunikacja awaryjna i terenowa;
- Działania survivalowe i gry outdoorowe;
- Projekty edukacyjne, hobystyczne i IoT;
- Eksperymenty z sieciami bezprzewodowymi i protokołem LoRa;

Dzięki prostej budowie, łatwej integracji oraz dużej elastyczności, **LoRaptor** stanowi solidne narzędzie zarówno dla użytkowników końcowych, jak i dla twórców rozwiązań technologicznych.

1.3 Off the grid, on the mesh - co to znaczy?

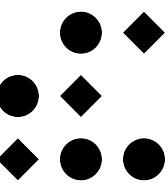
Hasło „**Off the grid**” odnosi się do działania poza tradycyjną infrastrukturą sieciową — bez dostępu do Wi-Fi, sieci komórkowej, internetu czy zasilania z sieci energetycznej. To stan całkowitej niezależności od zewnętrznych systemów.

Z kolei „**On the mesh**” wskazuje na działanie w ramach sieci mesh, czyli samodzielnie organizującej się sieci urządzeń, które komunikują się między sobą bez centralnego punktu. Topologia siatki (mesh) oznacza, że każde urządzenie może przekazywać dane do wielu innych urządzeń, tworząc elastyczną strukturę o wielu możliwych ścieżkach komunikacji, co zwiększa odporność sieci na awarie poszczególnych węzłów oraz zwiększa jej zasięg.

„Bez internetu, bez zasięgu, bez problemu — bo nasza sieć działa sama.”

To idealne podsumowanie idei **LoRaptora**:

- Urządzenia działają tam, gdzie nie ma infrastruktury;
- Tworzą własną, inteligentną i odporną sieć mesh;
- Pozwalają na komunikację prawie wszędzie — niezależnie od warunków;

OFF THE GRID 
ON THE MESH 

1.4 Przebieg pracy

Pomysł stworzenia systemu niezależnej komunikacji LoRa pojawił się na początku 2024 roku jako odpowiedź na rosnące zapotrzebowanie na niezależne, bezpieczne i funkcjonalne rozwiązania komunikacyjne. Inspiracją były zarówno projekty open-source, jak i własne doświadczenia z pracy w terenie oraz zamiłowanie do mikrokontrolerów ESP32.

Początkowo projekt istniał wyłącznie jako koncepcja. W połowie 2024 roku rozpoczęły się pierwsze konkretne działania:

Faza I – Eksperymenty i testy podstawowe

- Testy zasięgu modułów LoRa (RA-02) i wybór parametrów urządzenia.
- Pierwsze połączenia ESP32 z LoRa za pomocą interfejsu SPI.
- Próby z bibliotekami Arduino i eksperymenty z transmisją tekstu.

Faza II – Budowa sprzętu

- Projekt i montaż płytki PCB z modułem ESP32-S3-MINI-1.
- Zintegrowanie diody RGB i złącz USB-C.
- Testy zasilania i eliminacja zakłóceń w komunikacji.

Faza III – Rozwój oprogramowania

- Stworzenie własnej biblioteki do obsługi wiersza poleceń — RaptorCLI.
- Wykorzystanie biblioteki LoRaMesher jako podstawy komunikacji mesh.
- Opracowanie kompresji wiadomości za pomocą algorytmu SMAZ i szyfrowania AES (sprzętowe).
- Integracja komunikacji BLE oraz USB CDC.

Faza IV – Aplikacja mobilna

- Zaprojektowanie aplikacji RaptChat w języku Dart (Flutter).
- Obsługa połączeń przez BLE i USB z widokiem czatu.
- Testy interfejsu użytkownika i poprawa responsywności.

Faza V – Integracja i testy końcowe

- Połączenie wszystkich modułów sprzętowych i programowych w spójną całość.
- Testy funkcjonalne w warunkach zbliżonych do rzeczywistych.
- Projekt i wydruk obudowy 3D z litofanem (oknem sygnalizacyjnym).

Od pomysłu do wykonania minął ponad rok — od pierwszych szkiców i prób z przewodami na płytce stykowej, aż po działające urządzenie w obudowie, gotowe do pracy w terenie. Dziś, w marcu 2025 roku, **LoRaptor** stanowi w pełni działający prototyp, który spełnia założone cele i może być dalej rozwijany.

2 Część sprzętowa

Projekt **LoRaptor** wymagał stworzenia dedykowanej, kompaktowej i energooszczędnej platformy sprzętowej, która nie tylko umożliwi bezprzewodową komunikację dalekiego zasięgu, ale będzie również łatwa do integracji z urządzeniami końcowymi — smartfonami, komputerami czy innymi mikrokontrolerami.

Kluczowym elementem konstrukcji stała się zaprojektowana od podstaw płytka drukowana (PCB), zawierająca dwa główne moduły:

- **ESP32-S3-MINI-1-N8** — nowoczesny mikrokontroler firmy Espressif z obsługą BLE, Wi-Fi oraz USB-OTG, wyposażony w dużą moc obliczeniową i niskie zużycie energii.
- **RA-02 LoRa** (AIThinker) — wydajny moduł komunikacyjny oparty na technologii LoRa, pracujący na częstotliwości 433 MHz.

Oba moduły zostały połączone na jednej płytce, zaprojektowanej w środowisku **KiCad**, z myślą o maksymalnym wykorzystaniu przestrzeni i uproszczeniu produkcji.

2.1 Projekt PCB i montaż komponentów

Projekt płytki przewiduje:

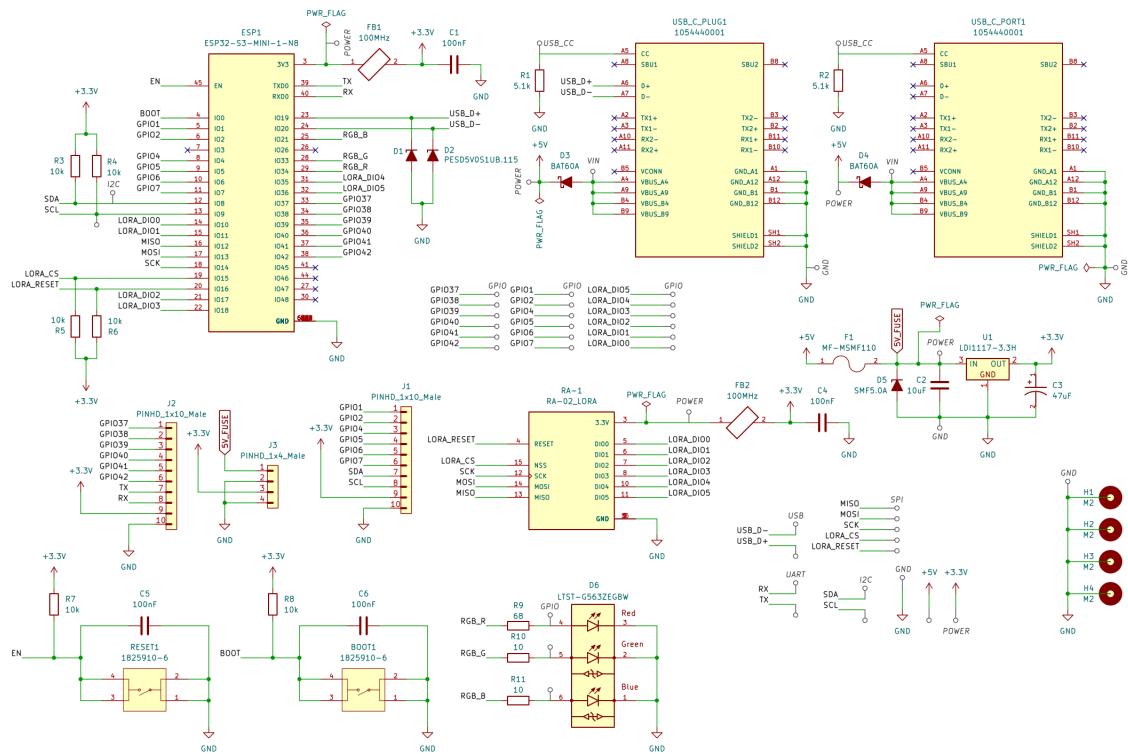
- Port USB-C (męski i żeński) z obsługą komunikacji i zasilania;
- Komunikację LoRa poprzez moduł RA-02;
- Sygnalizację stanu pracy za pomocą diody RGB;
- Przyciski BOOT i RESET dla konfiguracji systemu;
- Zabezpieczenia przed ESD i przepięciami;
- Filtrację zasilania z wykorzystaniem koralików ferrytowych i kondensatorów;
- Do stabilizacji napięcia zastosowano układ LDI1117-3.3H, zapewniający niezawodne zasilanie 3.3V dla wszystkich komponentów.

Na płytce umieszczono również złącza szpilkowe (raster 1.27 mm) przeznaczone do programowania, debugowania oraz rozszerzeń funkcjonalnych (wyprowadzono niewykorzystane piny GPIO). Płytkę ma grubość 0.8 mm, co pozwala na zastosowanie jej w obudowach o niewielkiej głębokości.

Dokumentacja techniczna projektu "LoRaptor"

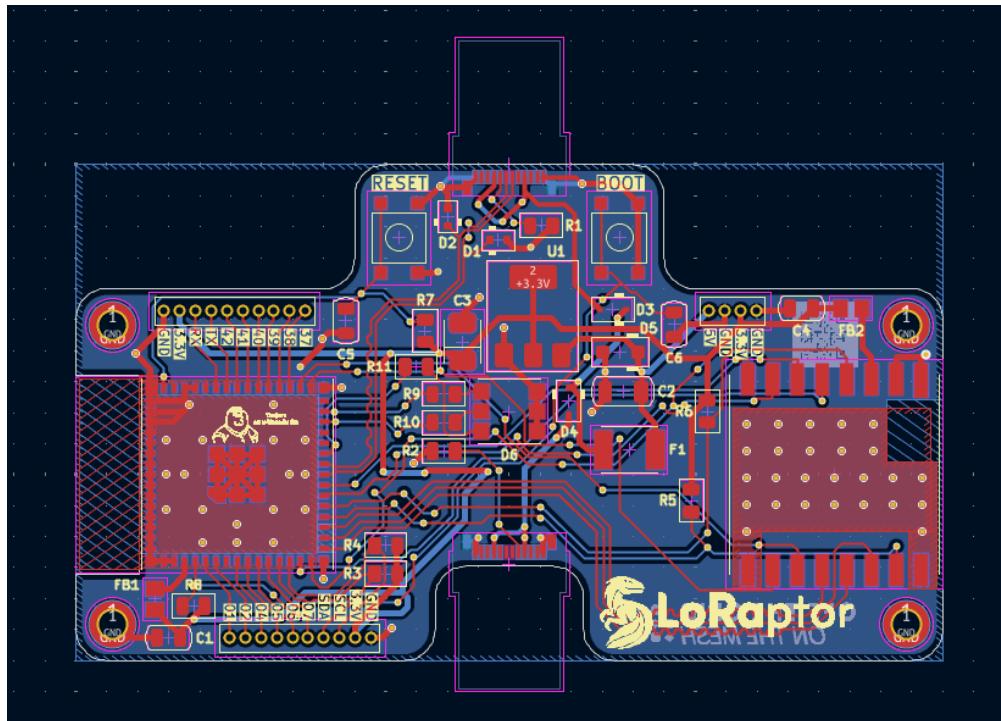
2.2 Schemat elektryczny

Poniżej znajduje się schemat elektryczny w wersji V1.0.0, przedstawiający pełne połączenia między komponentami, pinout ESP32, zasilanie i logikę sterowania.

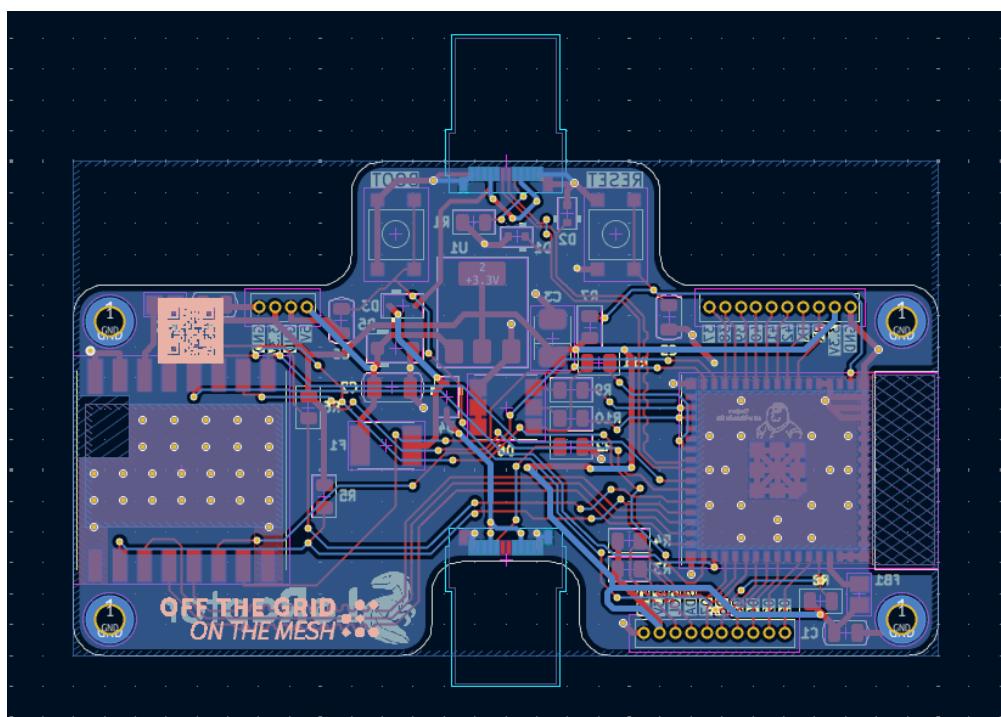


Rysunek 1: Schemat elektryczny płytki

Dokumentacja techniczna projektu "LoRaptor"

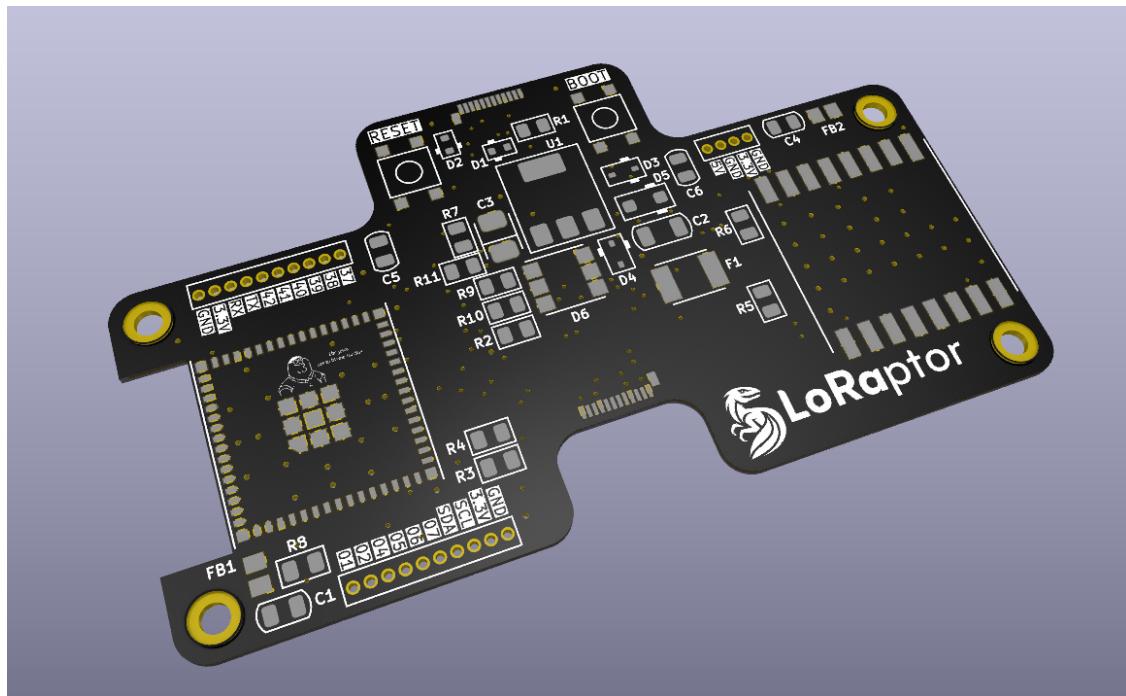


Rysunek 2: Układ płytki

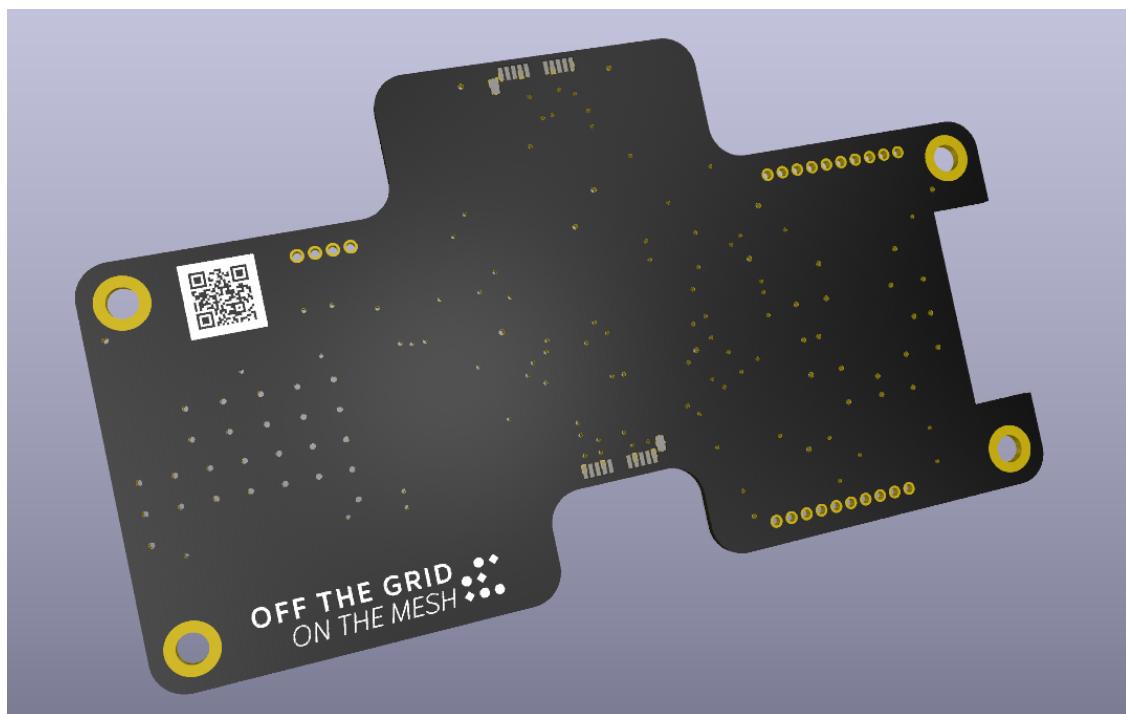


Rysunek 3: Układ płytki od spodu

Dokumentacja techniczna projektu "LoRaptor"



Rysunek 4: Widok 3D góry płytki



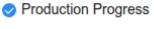
Rysunek 5: Widok 3D spodu płytki

2.3 Produkcja PCB

Płytki została wykonana w lutym 2025 roku w chińskiej firmie **JLCPCB** jako seria prototypowa. Zamówienie obejmowało:

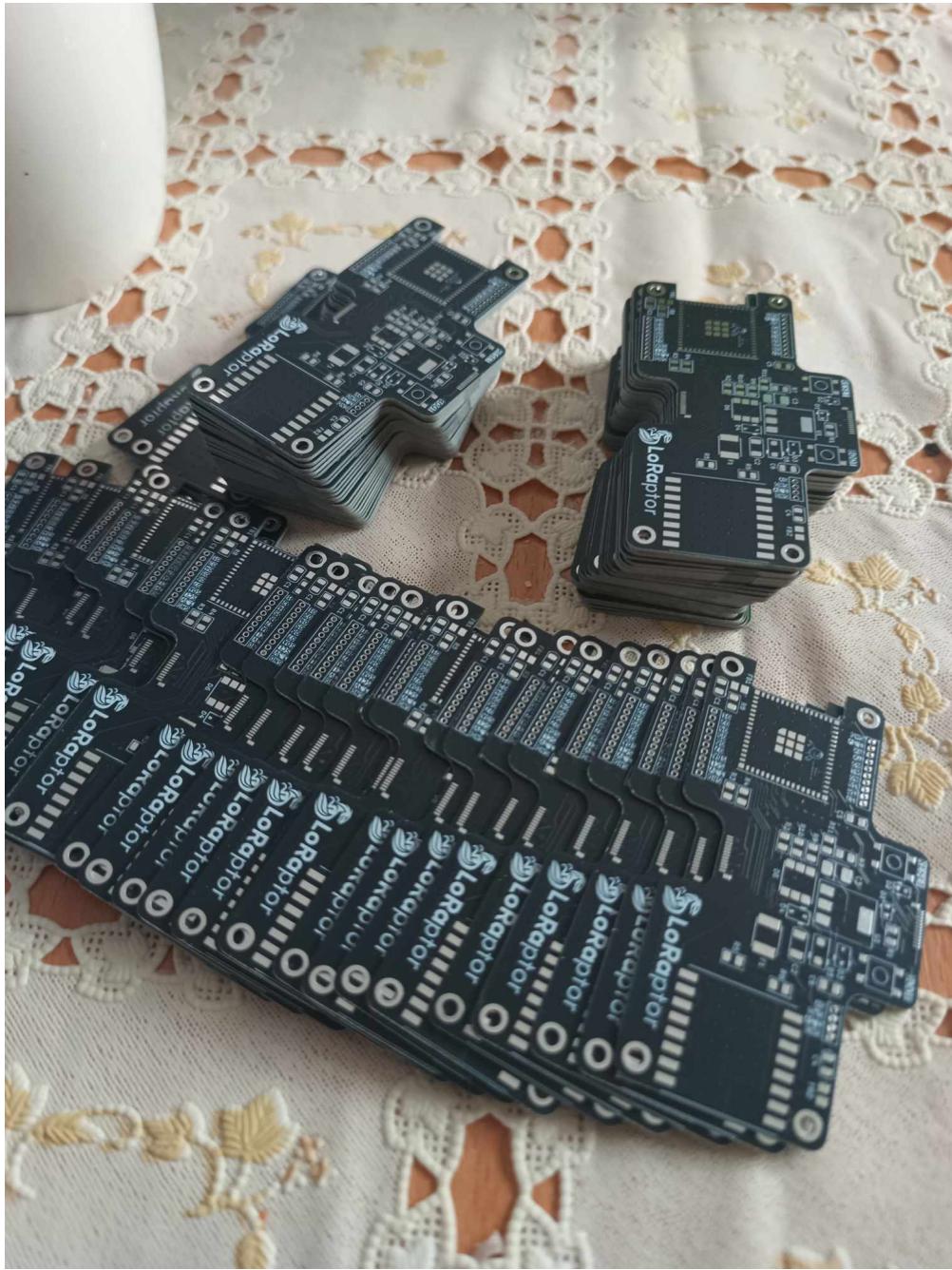
- 75 sztuk płyt (mała seria),
- 1 szablon SMT do nakładania pasty lutowniczej,
- całkowity koszt z wysyłką i opłatami: **\$59.77**.

Czas realizacji produkcji wynosił około 5-6 dni + dostawa UPS Express Saver.

| 2025-02-07 00:32:06 W202502070732343 PCB/Stencil | | | |
|--|---|---|--|
|  | Merchandise Total: \$25.10 Shipping Charge: \$23.49 Customs duties & taxes: \$11.18 Order Total: \$59.77 |  Shipped UPS Express Saver Shipment Tracking | |
|  PCB  Stencil Example | Small-batch PCB Order #: P1-9302189A Build Time: 5-6 days Product Details | \$18.10 75pcs | LoRaptor_V1.0.0_P1  |
| | SMT Stencil Order #: SO12502066843-9302189A Build Time: 1 day Product Details | \$7.00 1pcs | LoRaptor_V1.0.0_P1   |

Rysunek 6: Zamówienie produkcyjne w JLCPCB (75 płyt + stencil SMT)

Dokumentacja techniczna projektu "LoRaptor"



Rysunek 7: Płytki drukowane po otrzymaniu z produkcji



Rysunek 8: Stencil SMT do nakładania pasty lutowniczej

2.4 Lista elementów (BOM)

Pełna lista elementów zastosowanych na płytce znajduje się poniżej. Wszystkie komponenty dobrano tak, aby zapewnić kompatybilność z montażem SMT oraz łatwość ręcznego lutowania.

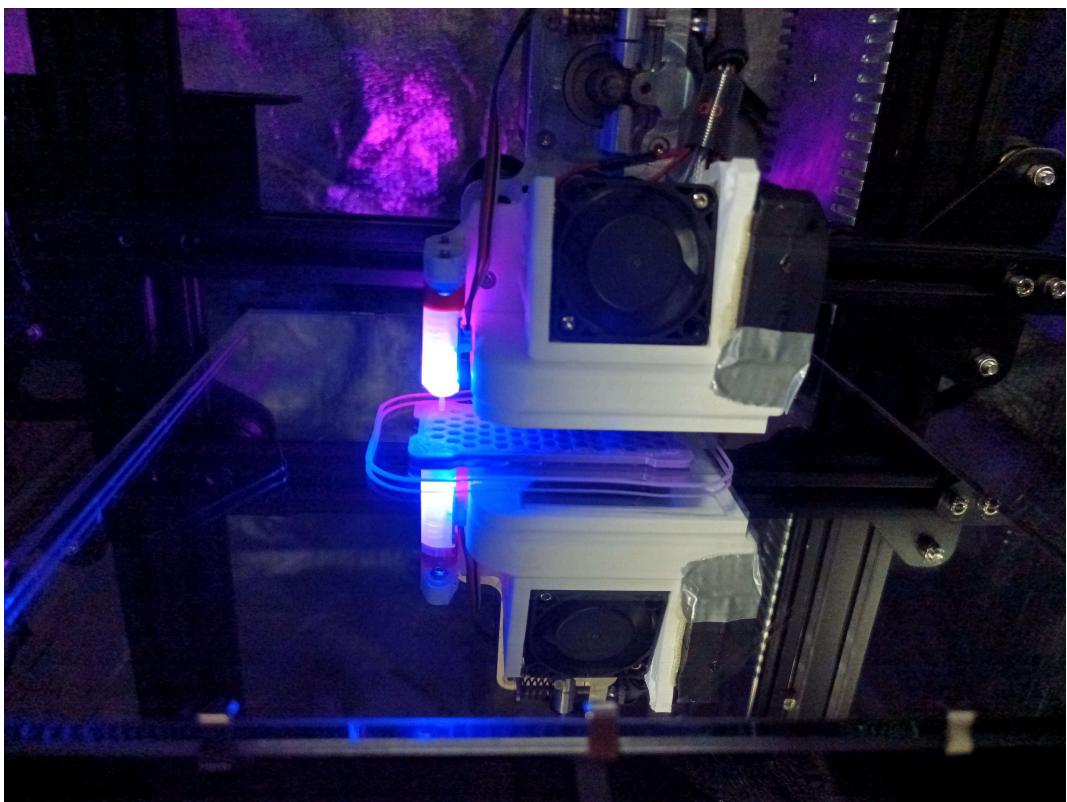
Zastosowano popularne obudowy SMD (0805 / 1206) i sprawdzone układy o dobrej dostępności.

- **ESP32-S3-MINI-1-N8** — główny mikrokontroler z BLE, Wi-Fi i USB;
- **RA-02 LoRa** — moduł komunikacji 433 MHz z anteną (Aliexpress, ok. 15 zł / szt.);
- **LTST-G563ZEBW** — dioda RGB LED;
- **1825910-6** — przyciski BOOT i RESET;
- **LDI1117-3.3H** — stabilizator napięcia 3.3V;
- **SMF5.0A, PESD5V0S1UB.115, BAT60A** — zabezpieczenia ESD i przeciwięciowe;
- **MF-MSMF110** — bezpiecznik polimerowy;
- **USB-C (żeńskie)** — gateryczne złącza zakupione na Aliexpress (nierekomendowane do finalnej produkcji, ale wystarczające do prototypów);
- Rezystory i kondensatory (5.1kΩ, 10kΩ, 68Ω, 10Ω, 100nF, 10uF, 47uF) – standardowe elementy pasywne SMD;

2.5 Obudowa urządzenia

Aby LoRaptor mógł funkcjonować jako w pełni przenośne urządzenie terenowe, konieczne było zaprojektowanie dedykowanej obudowy, chroniącej elektronikę przed uszkodzeniami mechanicznymi, a zarazem umożliwiającej łatwą obsługę i estetyczną prezentację projektu.

Obudowa została wykonana we własnym zakresie — zaprojektowana od podstaw w środowisku **Autodesk Inventor** (licencja studencka), a następnie wydrukowana na zmodyfikowanej drukarce **Creatality Ender 3**, przystosowanej do pracy z filamentami technicznymi.



Rysunek 9: Góra obudowy w trakcie drukowania na drukarce 3D

Do druku wykorzystano filament **PETG** polskiej firmy **FilHub**, w dwóch kolorach:

- **Fioletowy** — jako kolor bazowy, stanowiący główny korpus obudowy,
- **Mleczny biały** — zastosowany w górnej części obudowy jako element dekoracyjno-funkcjonalny.

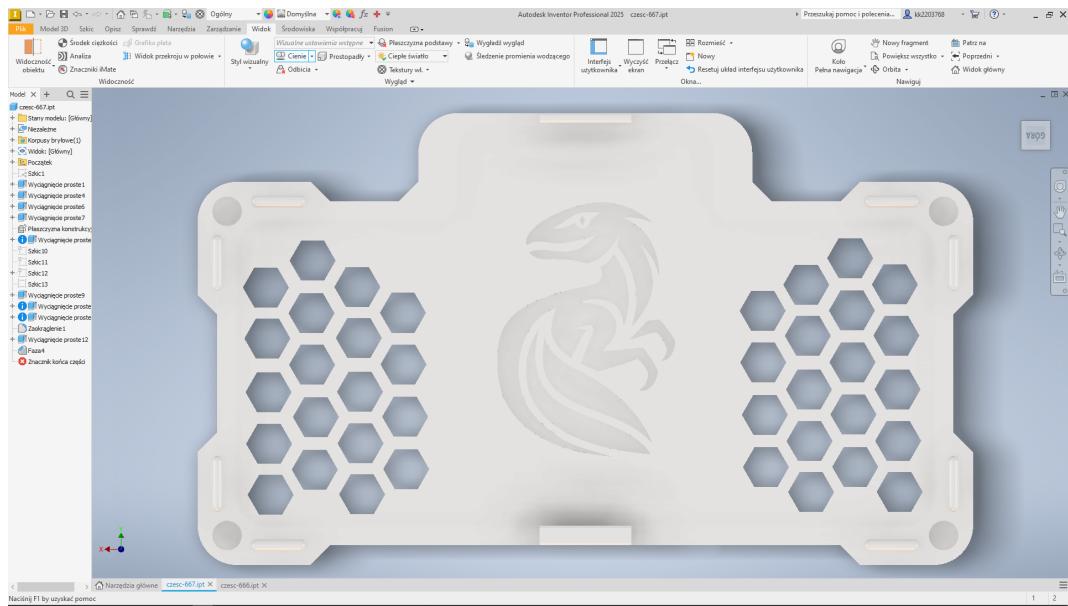
Materiał PETG został wybrany nieprzypadkowo — charakteryzuje się on doskonałym połączeniem wytrzymałości mechanicznej oraz odporności termicznej, znacznie przewyższając w tym zakresie standardowy filament PLA. Dodatkowo, PETG cechuje się zwiększoną odpornością na promieniowanie UV, co ma kluczowe znaczenie dla urządzenia przeznaczonego do pracy w terenie.

Elementy obudowy zostały wydrukowane na szklanym stole, co zapewniło pierwszej warstwie idealnie gładką powierzchnię. Jest to szczególnie istotne w przypadku górnej pokrywy, gdzie zastosowano efekt **litofanu** (ang. *lithophane*) — trójwymiarowej grafiki wygenerowanej w sposób umożliwiający podświetlenie jej od tyłu. Wykorzystując przezroczystość mlecznego PETG, w górnej pokrywie obudowy umieszczono logo projektu, które podświetlane jest od wewnętrz przez diodę LED RGB. Dioda ta pełni funkcję wskaźnika stanu urządzenia, przez co efekt litofanu jest nie tylko dekoracyjny, ale i funkcjonalny.

Całkowity czas wydruku kompletnej obudowy wyniósł około 3-4 godzin. Warto podkreślić, że dzięki zastosowaniu szklanego stołu uzyskano niemal lustrzaną powierzchnię materiału, co dodatkowo podnosi walory estetyczne całej konstrukcji.

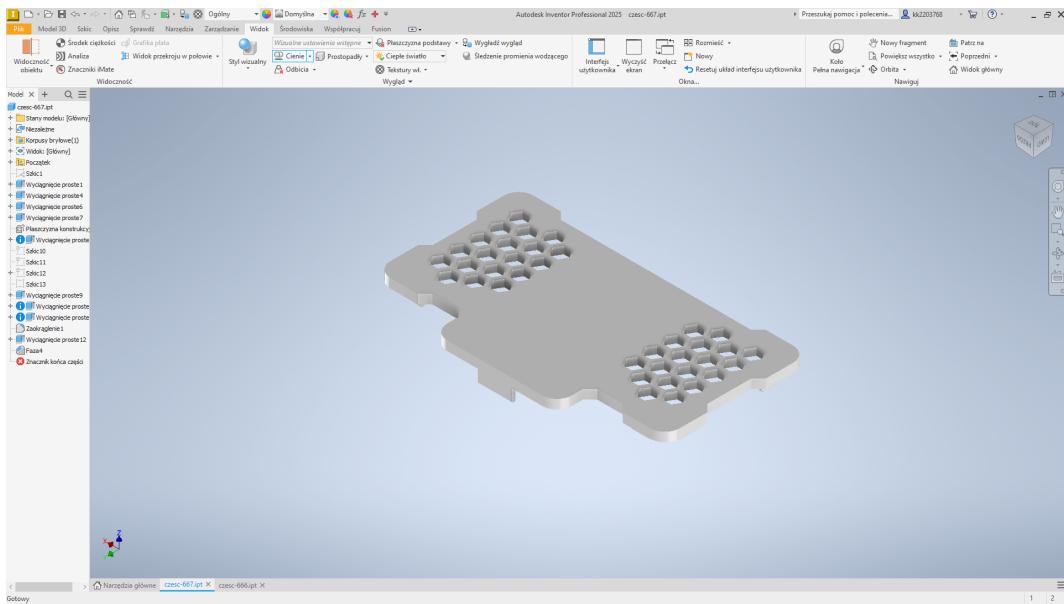
Dokumentacja techniczna projektu "LoRaptor"

Całość została zaprojektowana tak, aby możliwy był bezproblemowy dostęp do portów USB-C oraz zapewniona cyrkulacja powietrza w razie dłuższej pracy urządzenia. W strukturze obudowy zastosowano charakterystyczny wzór heksagonalny (plaster miodu), który nie tylko podniósł walory estetyczne, ale przede wszystkim zwiększył sztywność konstrukcji przy jednoczesnym ograniczeniu zużycia materiału. Taka struktura komórkowa pozwala na optymalne rozłożenie naprężeń mechanicznych, zwiększając odporność obudowy na odkształcenia, przy zachowaniu lekkości całej konstrukcji oraz zmniejszonej ilości wykorzystanego filamentu.

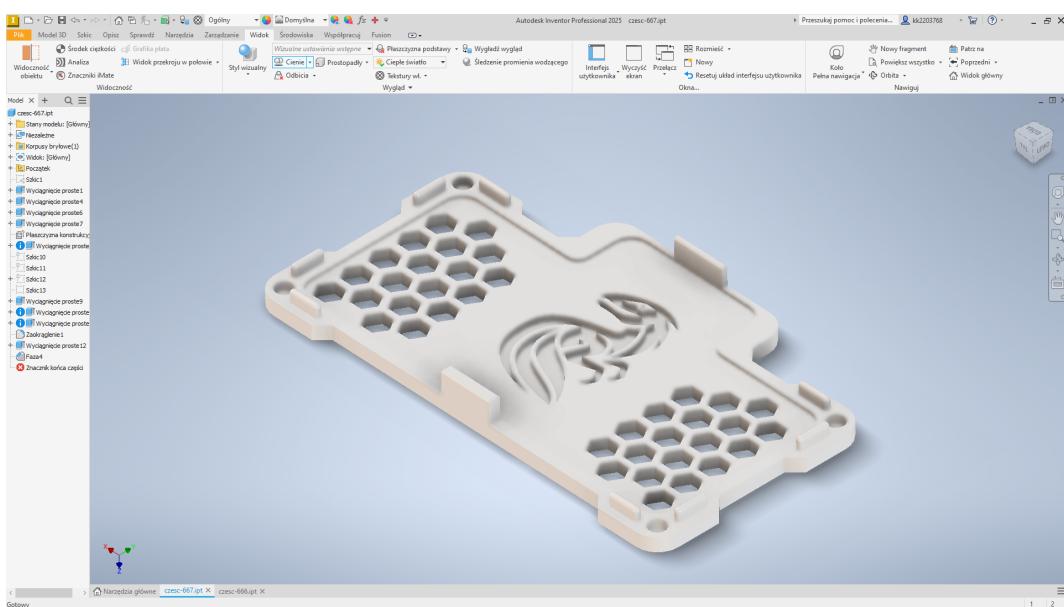


Rysunek 10: Górną część obudowy z funkcjonalnymi otworami wentylacyjnymi w strukturze heksagonalnej

Dokumentacja techniczna projektu "LoRaptor"

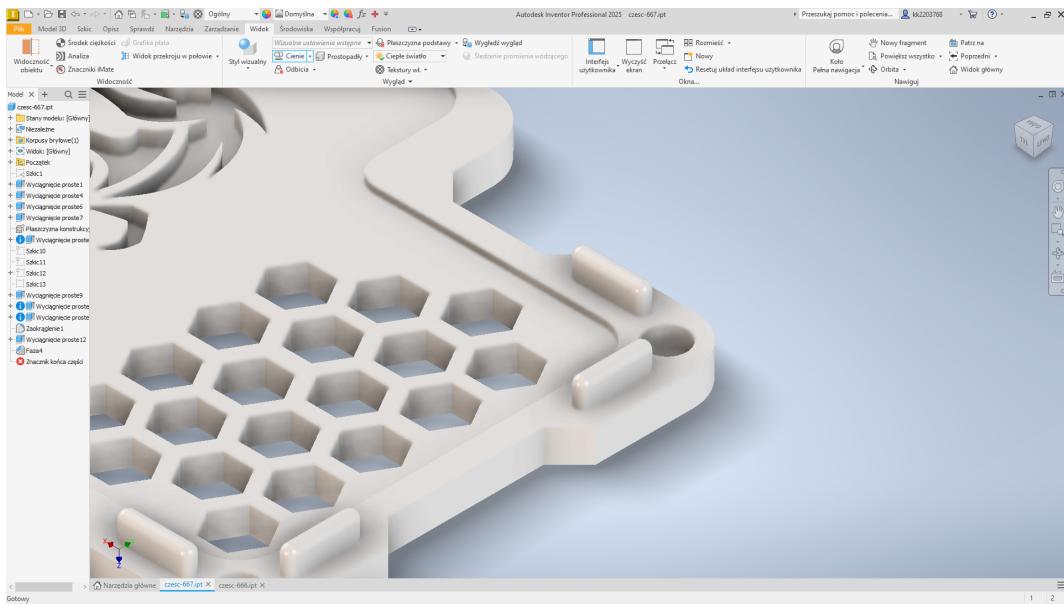


Rysunek 11: Widok z góry na górną część obudowy

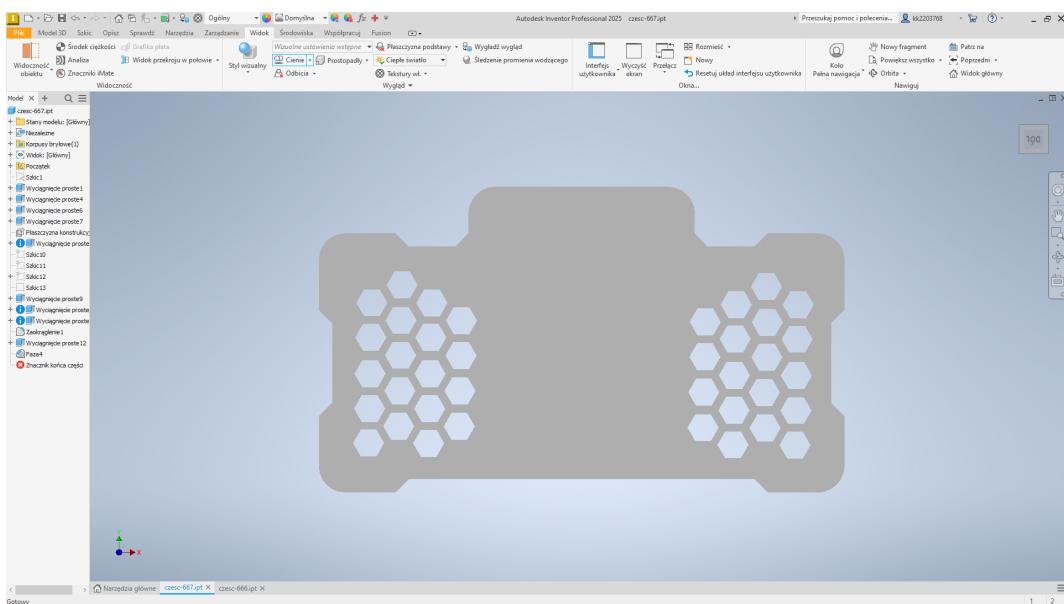


Rysunek 12: Perspektywiczne ujęcie górnej części obudowy ukazujące proporcje i kształt elementów

Dokumentacja techniczna projektu "LoRaptor"

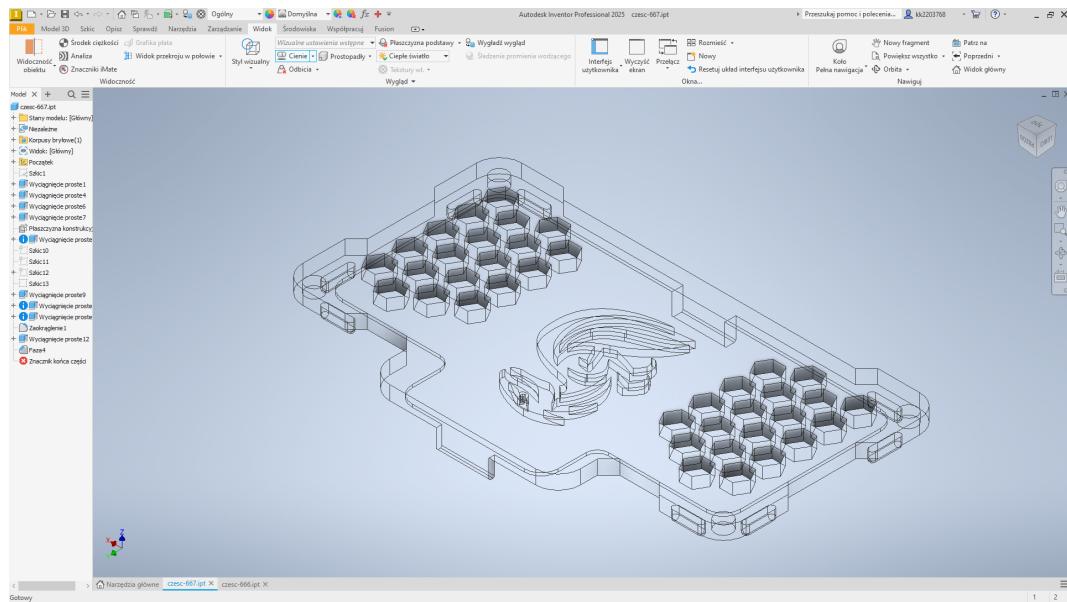


Rysunek 13: Zbliżenie na tylną część obudowy prezentujące wykonanie elementów łączeniowych

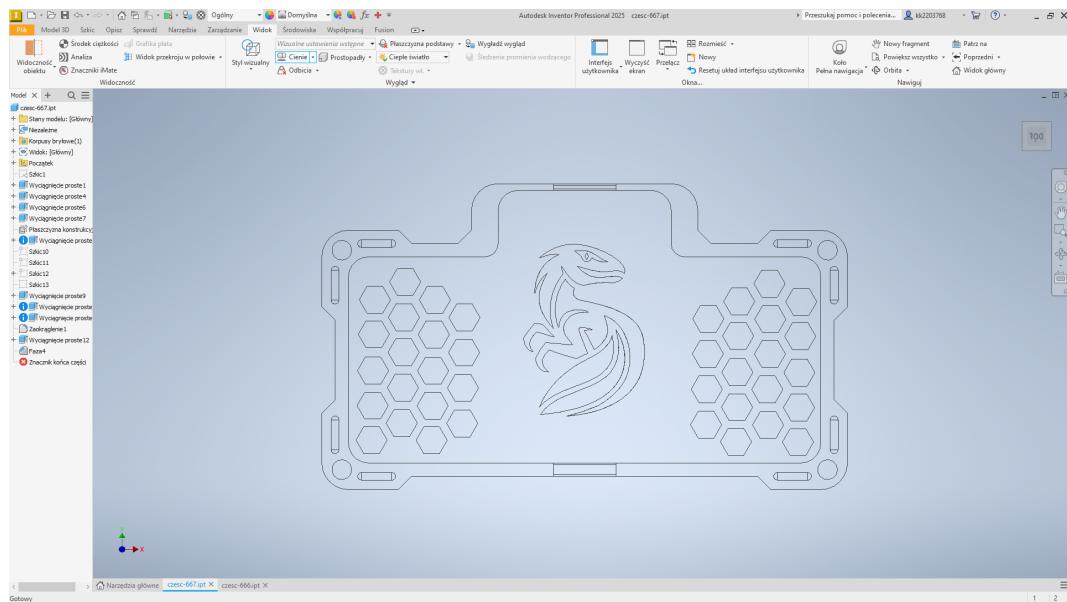


Rysunek 14: Front górnej części obudowy

Dokumentacja techniczna projektu "LoRaptor"

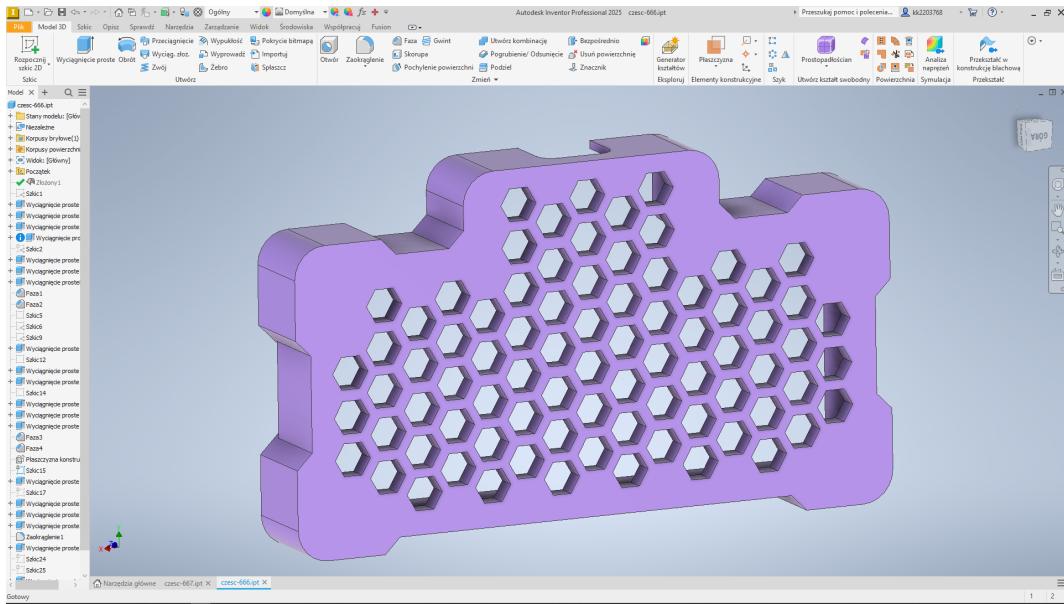


Rysunek 15: Wizualizacja szkieletowa górnej części obudowy

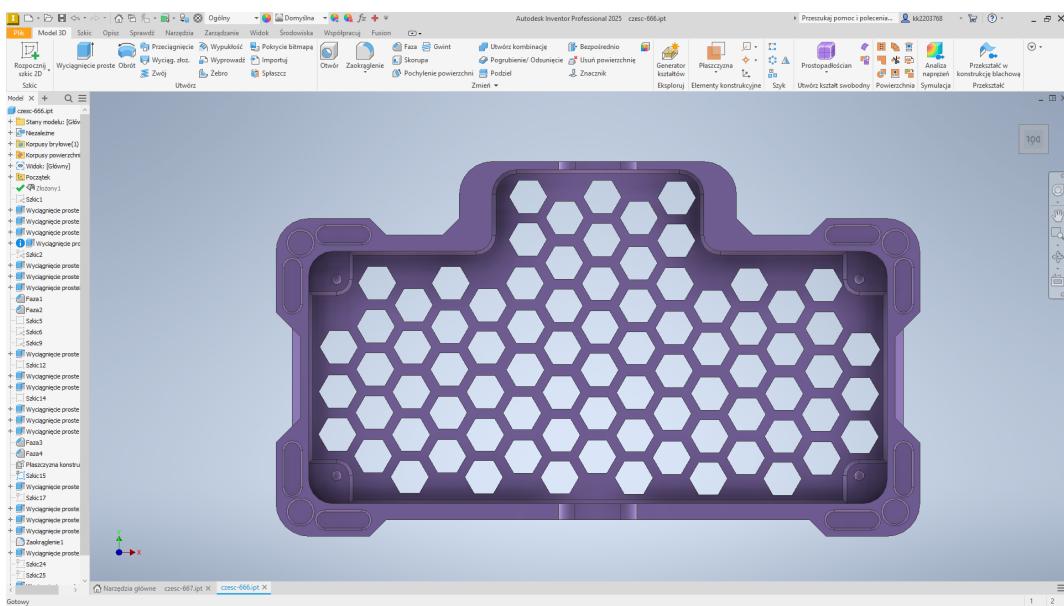


Rysunek 16: Frontowy widok szkieletowy górnej części obudowy

Dokumentacja techniczna projektu "LoRaptor"

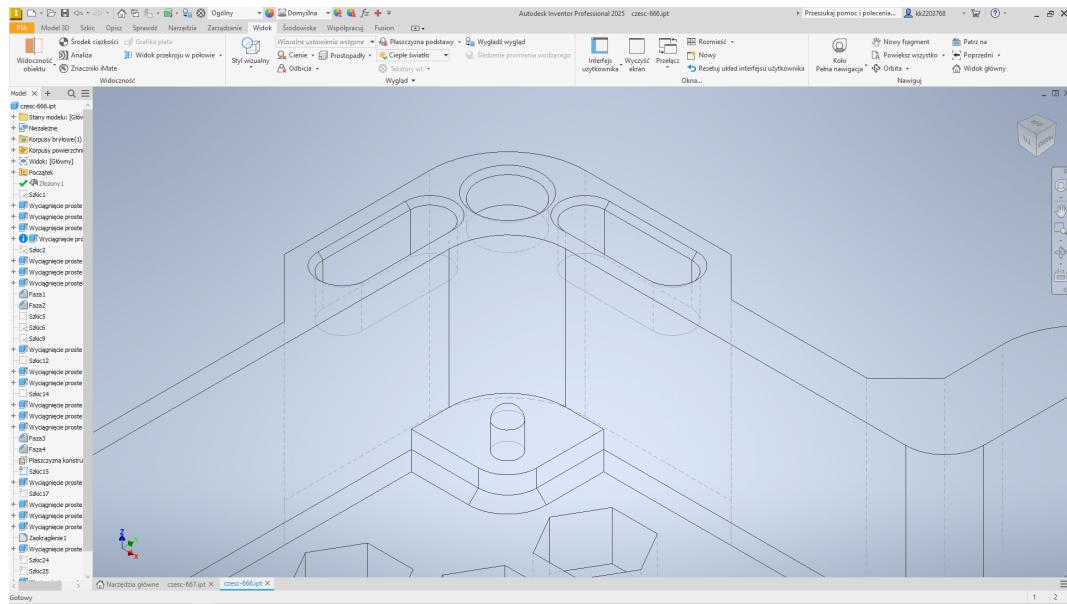


Rysunek 17: Widok tylnej strony dolnej obudowy z charakterystycznym wzorem wentylacyjnym

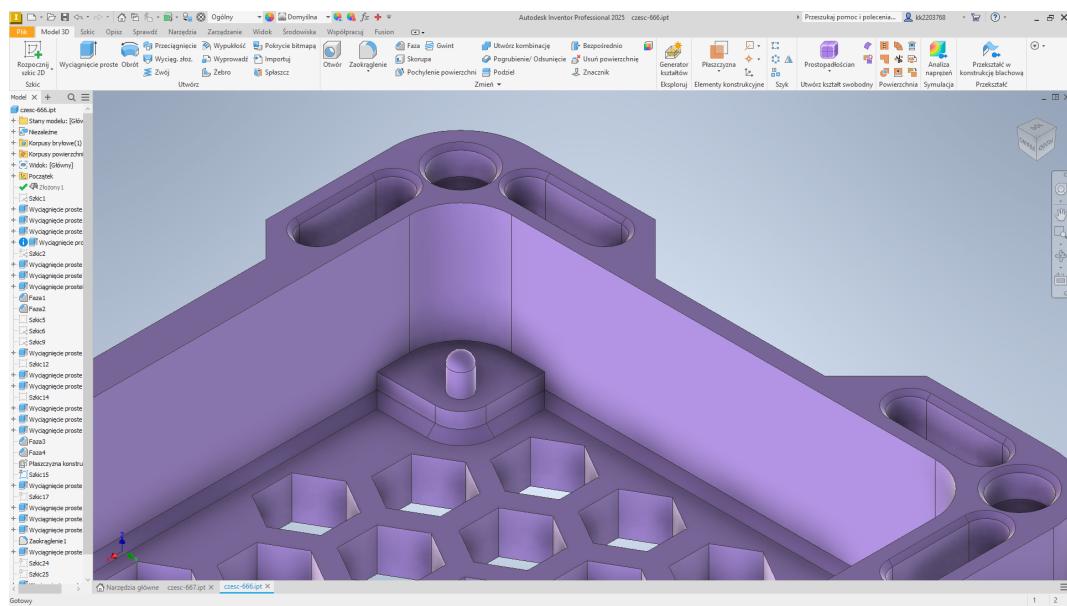


Rysunek 18: Widok frontalny dolnej części obudowy

Dokumentacja techniczna projektu "LoRaptor"

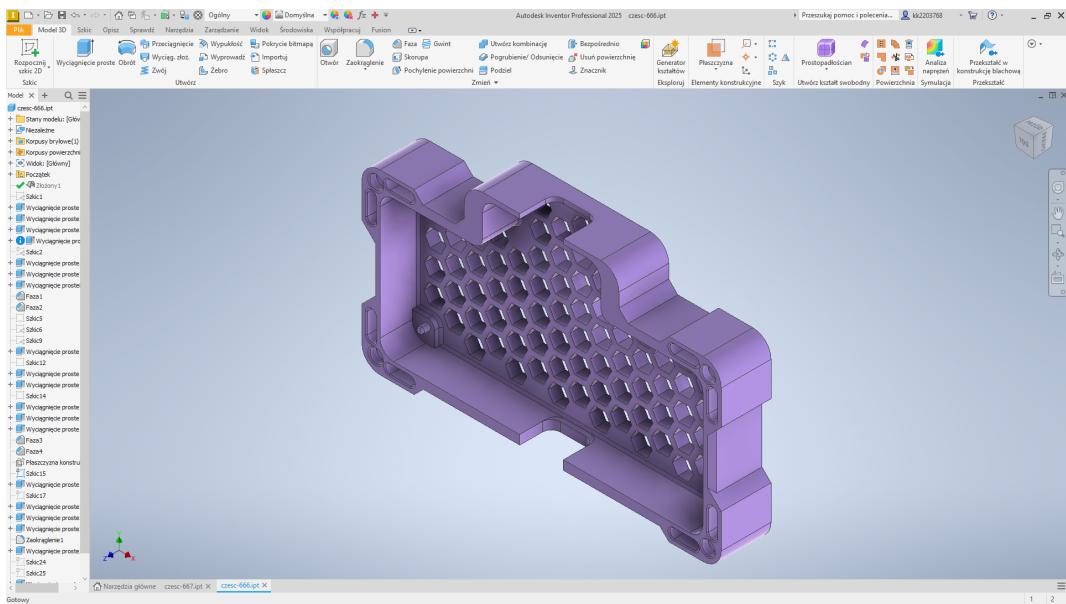


Rysunek 19: Zbliżenie na szkieletową strukturę dolnej części obudowy ukazujące detale mocowań



Rysunek 20: Zbliżenie na dolną część obudowy z widocznymi elementami mocującymi płytę drukowaną

Dokumentacja techniczna projektu "LoRaptor"



Rysunek 21: Profil boczny dolnej części obudowy

Obudowa stanowi ważne uzupełnienie projektu — łączy funkcję ochronną z estetyką i personalizacją, a efekt litofanu nadaje urządzeniu wyjątkowy charakter.

2.6 Złożona jednostka

Po zakończeniu montażu komponentów na PCB i przygotowaniu obudowy, powstała kompletna jednostka urządzenia **LoRaptor**. Całość została złożona ręcznie, z zachowaniem zasad montażu ESD oraz dokładnego dopasowania elementów mechanicznych.



Rysunek 22: Gotowa jednostka LoRaptora z zamontowaną obudową

Dzięki odpowiedniemu dopasowaniu oraz właściwościom materiału PETG, obudowa jest wytrzymała, odporna na wilgoć i promieniowanie UV, a jednocześnie lekka. Po montażu wszystkie elementy elektroniczne są dobrze chronione przed przypadkowym uszkodzeniem, co pozwala na bezpieczne użytkowanie w warunkach terenowych.



Rysunek 23: Efekt podświetlenia litofanu w górnej pokrywie obudowy (ciężko uchwycić na zdjęciu)

1209 / 2077 - LoRaptor

| | |
|----------------|---|
| VID | 0x1209 |
| PID | 0x2077 |
| Owner | Smuggr |
| License | MIT |
| Site | https://loraptor.smuggr.xyz/ |
| Source | http://github.com/smegg99/LoRaptor/ |

LoRaptor is a communication device leveraging the ESP32-S3-MINI-1 microcontroller and the RA-02 LoRa module. It facilitates seamless peer-to-peer communication over a mesh network topology, managed through an intuitive companion application.

Rysunek 24: Zrzut ekranu z przydzielonym kodem produktu na stronie pid.codes

Warto podkreślić, że LoRaptor otrzymał swój unikalny kod produktu USB (PID) przydzielony przez organizację pid.codes. Ta inicjatywa non-profit udogodnia identyfikatory PID dla projektów open-source, eliminując konieczność zakupu komercyjnych licencji USB-IF. Posiadanie dedykowanego kodu produktu pozwala na bezbłędną identyfikację urządzenia w systemach operacyjnych i zwiększa jego profesjonalizm. Jest to także potwierdzenie pełnej zgodności projektu z filozofią otwartego oprogramowania i sprzętu, co ułatwi społeczności rozwijanie LoRaptora w przyszłości.

2.7 Szacunkowy koszt jednej jednostki

Na podstawie faktur oraz kosztów zamówienia PCB oszacowano orientacyjny koszt produkcji jednej sztuki LoRaptora:

- **Płytkę PCB (JLCPCB)** — ok. 0,24 USD;
- **Wysyłka i cło** — ok. 0,45 USD;
- **Komponenty z TME** — ok. 6,50-7,00 USD;
- **RA-02 + antena** — ok. 3,50 USD;
- **Złącze USB-C (żeńskie)** — ok. 0,30-0,50 USD (Aliexpress);

Łączny koszt produkcji jednej sztuki: ok. 10-11 USD (około 40-45 złotych, w zależności od kursu walut i dostawców).

Zastosowanie komponentów z TME oraz tanich zamienników z Aliexpress pozwoliło utrzymać niski koszt jednostkowy, przy zachowaniu funkcjonalności niezbędnej w testach i prototypowaniu.

3 Część programowa

3.1 Biblioteka LoRaMesher

Aby zapewnić pełną funkcjonalność komunikacji typu mesh w projekcie **LoRaptor**, wykorzystano otwartoźródłową bibliotekę LoRaMesher. Biblioteka ta umożliwia tworzenie zdecentralizowanej sieci, w której urządzenia komunikują się bezpośrednio między sobą, bez potrzeby istnienia centralnego węzła, jakim w klasycznych systemach LoRaWAN jest bramka.

3.1.1 Czym jest LoRaMesher?

LoRaMesher to biblioteka w języku C++ implementująca proaktywny protokół routingu oparty na tablicach odległości (distance-vector). Pozwala ona urządzeniom tworzyć samodzielnie organizującą się sieć mesh, w której paczki danych są przekazywane między węzłami aż do miejsca docelowego.

Dzięki integracji z biblioteką **RadioLib**, **LoRaMesher** umożliwia obsługę komunikacji LoRa, wykrywanie pakietów przez przerwania oraz automatyczne zarządzanie czasem antenowym i unikaniem kolizji.

3.1.2 Struktura wiadomości i komunikacja

Każda wiadomość przesyłana w sieci składa się z nagłówka (header) i ładunku danych (payload). Nagłówek zawiera informacje takie jak adres źródłowy i docelowy, typ wiadomości oraz rozmiar danych. W sieci występują dwa rodzaje wiadomości:

- **Wiadomości routujące (routing messages)** — wysyłane cyklicznie przez każdy węzeł w celu aktualizacji lokalnych tablic routingu.
- **Wiadomości danych (data messages)** — zawierają właściwe dane użytkownika przesyłane do określonego węzła.

Routing w sieci odbywa się w pełni automatycznie. Węzły analizują wiadomości routujące sąsiadów i budują na ich podstawie optymalne trasy do pozostałych urządzeń. Gdy nadjejdzie wiadomość danych, węzeł może ją odebrać (jeśli jest adresatem), przekazać dalej (jeśli jest kolejnym ogniwem trasy), albo odrzucić (jeśli pakiet go nie dotyczy).

3.1.3 Architektura zadań i kolejek

LoRaMesher działa w środowisku **FreeRTOS** i opiera się na szeregu zadań (tasks) oraz kolejek pakietów (queues), które zarządzają cyklem życia wiadomości:

- **Q_RP (Received Packets)** — odbierane pakiety oczekujące na przetwarzanie.
- **Q_SP (Send Packets)** — kolejka pakietów do wysłania.
- **Q_UPR (User Received Packets)** — dane przeznaczone do aplikacji użytkownika.

Zadania odpowiedzialne za pracę sieci to m.in.:

- **Receive Task** — odbiera pakiety przez LoRa i przekazuje je do przetwarzania.
- **Send Task** — wysyła pakiety z kolejki Q_SP, z uwzględnieniem cyklu obowiązkowej przerwy (duty cycle) i sprawdzania dostępności kanału (CAD).
- **Routing Protocol Task** — cyklicznie wysyła wiadomości routujące do sąsiadów.
- **Process Task** — analizuje pakiety z Q_RP i decyduje, co dalej z nimi zrobić.
- **User Send Task** — interfejs programistyczny aplikacji użytkownika do wysyłania danych.
- **User Receive Task** — odbiór danych przez aplikację użytkownika.

3.1.4 Abstrakcja danych i integracja w LoRaptorze

LoRaptor, poza implementacją sieci typu mesh opartej na bibliotece LoRaMesher, wprowadza własną warstwę abstrakcji dla danych użytkownika. Każda wiadomość przesyłana przez siatkę LoRa nie jest przesyłana jako surowy tekst, lecz jako specjalnie przygotowany obiekt klasy `Payload`. Celem tego podejścia jest zapewnienie:

- struktury danych możliwej do rozbudowy,
- kompresji dla oszczędności przepustowości,
- szyfrowania dla bezpieczeństwa transmisji.

3.1.5 Format danych Payload

Każdy obiekt typu `Payload` zawiera następujące informacje:

- **publicWord** – hasło rozpoznawcze, np. identyfikator połączenia,
- **epoch** – znacznik czasu wysyłki (UNIX timestamp),
- **type** – typ danych,
- **content** – właściwa treść wiadomości.

Te dane są najpierw łączone w jeden ciąg znaków, a następnie:

1. **kompresowane** przy użyciu algorytmu Smaz2 — zoptymalizowanego pod krótkie wiadomości tekstowe,
2. **szyfrowane** przy użyciu AES-128 w trybie CBC,
3. **kodowane** w base64 w celu przesyłania przez kanały tekstowe.

3.1.6 Wysyłanie danych przez LoRaMesher

Tak przygotowany zaszyfrowany i skompresowany ciąg znaków jest przekazywany do biblioteki LoRaMesher jako zawartość wiadomości:

```
1 // Przyklad tworzenia wiadomosci
2 Payload payload("conn01", epochTime, "Hello world!", PayloadType::
3   MESSAGE);
4 std::string encoded;
5 if (payload.encode(connectionKey, encoded)) {
6   Message outgoingMsg(encoded, epoch, localNodeID, PayloadType::
7     MESSAGE);
8   // ... wysylka przez LoRaMesher
9 }
```

3.1.7 Dekodowanie po stronie odbiorcy

Po odebraniu wiadomości, urządzenie wykonuje operację odwrotną:

1. Odszyfrowanie danych przy pomocy klucza połączenia,
2. Dekompresję,
3. Parsowanie pól publicWord, epoch, type i content.

Wszystko to realizowane jest metodą:

```
1 Payload decoded;
2 if (Payload::decode(receivedEncryptedMsg, connectionKey, decoded)) {
3   std::string msgContent = decoded.getContent();
4   // obsługa wiadomości
5 }
```

3.1.8 Korzyści projektowe

Takie podejście zapewnia:

- **Rozdzielenie warstwy transmisji od warstwy danych**, co pozwala łatwo zmienić medium komunikacji (LoRa, Wi-Fi, BLE).
- **Bezpieczeństwo**, ponieważ dane są szyfrowane symetrycznie (AES-128).
- **Wydajność**, dzięki zastosowaniu lekkiego algorytmu kompresji.
- **Elastyczność**, ponieważ strukturę danych można łatwo rozszerzać o dodatkowe pola.

3.1.9 Podsumowanie

Klasa Payload działa jako inteligentna warstwa pośrednia pomiędzy logiką aplikacyjną a siecią LoRaMesher. To rozwiązanie nie tylko porządkuje przesyłane dane, ale również zabezpiecza je i optymalizuje pod kątem transmisji w sieci o niskiej przepustowości, jaką jest LoRa.

3.2 Biblioteka RaptorCLI

RaptorCLI to autorska biblioteka służąca do definiowania, parsowania i wykonywania poleceń tekstowych w systemach wbudowanych. Powstała jako moduł firmware'u LoRaptor, umożliwiając użytkownikowi interakcję z urządzeniem poprzez USB (Serial) lub BLE (Bluetooth Low Energy), w stylu przypominającym klasyczne powłoki terminalowe, takie jak **Bash**.

3.2.1 Architektura i integracja

Biblioteka oparta jest na architekturze komend i dispatcherów. W firmware LoRaptor odpowiada za przyjmowanie poleceń (tekstowych stringów), parsowanie ich oraz wywoływanie odpowiednich callbacków z argumentami.

Polecenia przyjmowane są jednym z dwóch kanałów:

- **USB Serial** — z wykorzystaniem klasy **SerialComm**,
- **BLE NUS** — z wykorzystaniem klasy **BLEComm** (Nordic UART Service).

Każde polecenie trafia do centralnej kolejki **commandQueue** i jest przetwarzane w osobnym zadaniu systemu FreeRTOS:

```
1 xTaskCreate(commandProcessingTask, "CommandProcessingTask", 32768, NULL  
    , 1, NULL);
```

Dispatcher i **CLIOoutput** zapewniają przekierowanie komend jak i komunikatów zwrotnych do odpowiedniego kanału (BLE lub Serial).

3.2.2 Połączenia – czym one są?

Połączenia (ang. *connections*) w systemie LoRaptor stanowią abstrakcję kanałów komunikacyjnych pomiędzy węzłami sieci. Każde połączenie jest definiowane przez:

- **Unikalny identyfikator** — rozpoznawalny wewnątrz urządzenia string identyfikujący konkretne połączenie
- **Klucz kryptograficzny** — używany do szyfrowania i deszyfrowania transmisji
- **Lista odbiorców** — zbiór adresów węzłów sieci, do których kierowane są wiadomości

Z perspektywy użytkownika, połączenia można porównać do „kanałów komunikacyjnych” lub „czatów grupowych”. Każde połączenie ma swoją nazwę (identyfikator), własny klucz zabezpieczający wymianę informacji oraz listę uczestników, którzy mogą odbierać i wysyłać wiadomości. Dzięki interfejsowi **RaptorCLI**, użytkownik może w prosty sposób tworzyć nowe kanały komunikacyjne, dołączać do nich inne urządzenia oraz wysyłać i odbierać wiadomości — wszystko za pomocą prostych poleceń tekstowych, bez potrzeby zagłębiania się w techniczne szczegóły działania sieci.

3.2.3 Przykładowe polecenia

RaptorCLI obsługuje komendy z argumentami, aliasami oraz typowaniem argumentów. Oto przykłady: Utworzenie połączenia:

```
1 create connection -id "test" -k "secretKey123" -r [53052, 16888]
```

Wysłanie wiadomości do odbiorców:

```
1 send -id "test" -m "Czesc, jestescie tam?"
```

Odczyt odebranych wiadomości z bufora:

```
1 flush -id "test"
```

Pobranie identyfikatora węzła:

```
1 get nodeID
```

Ustawienie czasu RTC (epoch):

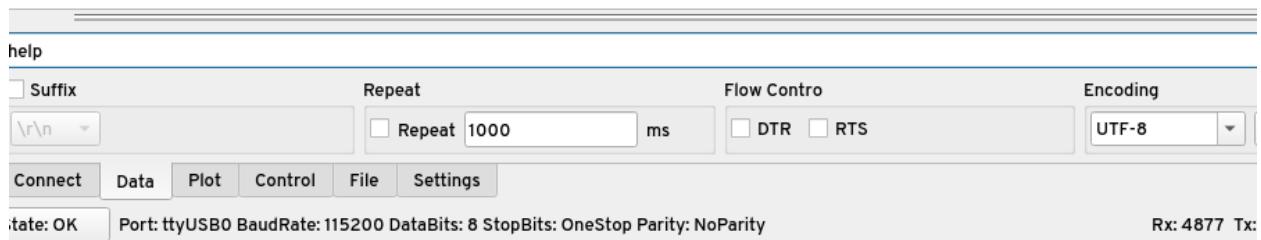
```
1 set rtc -t 1711241193
```

Wyświetlenie wszystkich komend:

```
1 help
```

Dokumentacja techniczna projektu "LoRaptor"

```
Received command: help
help (aliases: ?) - Displays help information for all commands.
send - Send a message on a connection
    Arguments:
        -id (string) required -- Connection ID
        -m (string) required -- Message
flush - Flushes the message queue
    Arguments:
        -id (string) required -- Connection ID
ping - Pings the system to check if it's responsive
create - Creates a new item
Subcommands:
connection - Creates a new connection
    Arguments:
        -id (string) required -- Connection ID
        -k (string) required -- Connection key
        -r (list) required -- Recipient IDs
connectionRecipient - Adds a recipient to a connection
    Arguments:
        -id (string) required -- Connection ID
        -r (int) required -- Recipient ID
update - Updates an item
delete - Deletes an item
Subcommands:
connection - Deletes a connection
    Arguments:
        -id (string) required -- Connection ID
connectionRecipient - Removes a recipient from a connection
    Arguments:
        -id (string) required -- Connection ID
        -r (int) required -- Recipient ID
list - Lists all items
Subcommands:
connections - Lists all connections
nodes - Lists all nodes
get - Gets a value
Subcommands:
nodeID - Gets the node ID
rtc - Gets the RTC time
set - Sets a value
Subcommands:
rtc - Sets the RTC time
    Arguments:
        -t (int) required -- Time in seconds since epoch
```



Rysunek 25: Przykładowy wynik komendy `help` z monitora portu szeregowego

3.2.4 Definiowanie komend w kodzie

Komendy są rejestrowane w funkcji registerCommands(), np.:

```
1 Command createConnCmd("connection", "Tworzy nowe polaczenie", output,
2   createConnCallback);
3
4 createConnCmd.addArgSpec(ArgSpec("id", VAL_STRING, true, "ID polaczenia
5   "));
6
7 createConnCmd.addArgSpec(ArgSpec("k", VAL_STRING, true, "Klucz"));
8
9 createConnCmd.addArgSpec(ArgSpec("r", VAL_LIST, true, "Lista odbiorcow"
10   ));
```

Funkcja createConnCallback zostanie wywołana, gdy komenda zostanie poprawnie wywołana z argumentami. W przypadku błędu, wyświetlony zostanie odpowiedni komunikat i zwrócony zostanie kod błędu.

3.2.5 Argumenty poleceń

Biblioteka RaptorCLI oferuje zaawansowany system obsługi argumentów poleceń. Każda komenda może definiować wymagane i opcjonalne argumenty różnych typów za pomocą metody addArgSpec(). Specyfikacja argumentu zawiera:

- **Nazwę** - identyfikator argumentu używany przy parsowaniu
- **Typ** - może być VAL_INT, VAL_STRING, VAL_LIST lub inny zdefiniowany typ
- **Wymagalność** - flaga określająca czy argument jest obowiązkowy
- **Opis** - tekst pomocy wyświetlany przy użyciu komendy help

Przykład definiowania argumentów:

```
1 command.addArgSpec(ArgSpec("name", VAL_STRING, true, "Nazwa elementu"))
  ;
2 command.addArgSpec(ArgSpec("count", VAL_INT, false, "Liczba elementów (
  opcjonalne)));
```

3.2.6 Argumenty o nieograniczonej liczbie

RaptorCLI obsługuje również komendy przyjmujące zmienną liczbę argumentów za pomocą funkcji `setVariadic()`. Gdy komenda jest oznaczona jako *variadiczna*, może przyjmować dowolną liczbę dodatkowych argumentów po zdefiniowanych parametrach.

```
1 Command echoCmd("echo", "Wypisuje wszystkie podane argumenty", output,
2   echoCmdCallback);
echoCmd.setVariadic(true);
```

Argumenty variadyczne są następnie dostępne w callbacku przez tablicę `arguments`:

```
1 void echoCmdCallback(const Command& cmd) {
2   CLIOutput* output = dispatcher.getOutput();
3   for (const auto& arg : cmd.arguments) {
4     output->println(arg.c_str());
5   }
6 }
```

Ta elastyczność umożliwia tworzenie poleceń podobnych do tych znanych z po-włok systemów operacyjnych, przyjmujących zmienną liczbę parametrów.

3.2.7 Zarządzanie wyjściem i wejściem

Wyjście i wejście CLI różni się w zależności od interfejsu:

- **ArduinoCLIOoutput** — dla USB Serial,
- **BLECLIOoutput** — dla komunikacji BLE.

W zależności od konfiguracji podczas komplikacji, wybierane jest odpowiednie wyjście:

```
1 #ifdef USE_SERIAL_COMM
2
3     ArduinoCLIOoutput serialOutput;
4
5     dispatcher.registerOutput(&serialOutput);
6
7 #else
8
9     BLECLIOoutput bleOutput((BLEComm*)commChannel);
10
11    dispatcher.registerOutput(&bleOutput);
12
13#endif
```

3.2.8 Podsumowanie

Biblioteka RaptorCLI stanowi kluczowy element interakcji użytkownika z systemem LoRaptor. Dzięki niej, użytkownik może dynamicznie zarządzać urządzeniem, wysyłać wiadomości, tworzyć połączenia, modyfikować parametry systemu i odbierać dane — wszystko z poziomu aplikacji RaptChat lub terminala.

3.3 Aplikacja RaptChat

RaptChat to mobilna aplikacja napisana w języku **Dart** z użyciem frameworka **Flutter**, przeznaczona do zarządzania urządzeniem LoRaptor i prowadzenia bezprzewodowej komunikacji tekstowej za pośrednictwem sieci mesh. Aplikacja umożliwia użytkownikowi:

- wyszukiwanie i łączenie się z urządzeniami LoRaptor przez BLE (Bluetooth Low Energy),
- konfigurację połączeń,
- wysyłanie oraz odbieranie wiadomości,
- wizualizację aktualnych węzłów w sieci,
- zarządzanie lokalnymi ustawieniami i urządzeniami.



Rysunek 26: Główne logo RaptChat



Rysunek 27: Alt. logo RaptChat

3.3.1 Architektura działania

Aplikacja łączy się z urządzeniem za pomocą profilu **NUS (Nordic UART Service)**, który zapewnia kanał tekstowej komunikacji BLE. Dane przesypane przez aplikację są komendami w formacie zdefiniowanym przez bibliotekę RaptorCLI. Aplikacja posiada własny parser CLI (dispatcher), który umożliwia zrozumienie odpowiedzi urządzenia i podejmowanie akcji na podstawie ich typu.

3.3.2 Komunikacja z urządzeniem

- Komendy są wysyłane tekstowo, np.:

```
1 send -id "test123" -m "Czesc!"
```

- Odpowiedzi są przetwarzane w czasie rzeczywistym i klasyfikowane według ich typu, do przykładowych typów należą:
 - msg.send.success — potwierdzenie wysłania wiadomości,
 - msg.conn.created — potwierdzenie utworzenia połączenia,
 - type.flush.mess — wiadomości odebrane z sieci,
 - type.list.nodes — lista aktualnie znanych węzłów mesh.

3.3.3 Warstwa BLE

Za zarządzanie połączeniem Bluetooth odpowiada klasa BleDeviceManager. Obsługuje ona:

- skanowanie i filtrowanie urządzeń LoRaptor,
- parowanie, łączenie i rozłączanie urządzeń,
- inicjalizację kanału komunikacji NUS,
- konfigurację początkową urządzenia (RTC, połączenia itp.),
- przesyłanie i odbieranie komend.

3.3.4 Automatyzacja konfiguracji

Po każdym połączeniu z urządzeniem aplikacja wykonuje tzw. komendy startowe:

1. Ustawienie czasu RTC na podstawie zegara telefonu:

```
1 set rtc -t 1711000000
```

2. Odtworzenie wcześniej zapisanych połączeń i ich odbiorców:

```
1 create connection -id "test" -k "key1234" -r [53052, 16888]
```

3.3.5 Warstwa wiadomości

Logiką wiadomości zarządza `MessagesManager`, który:

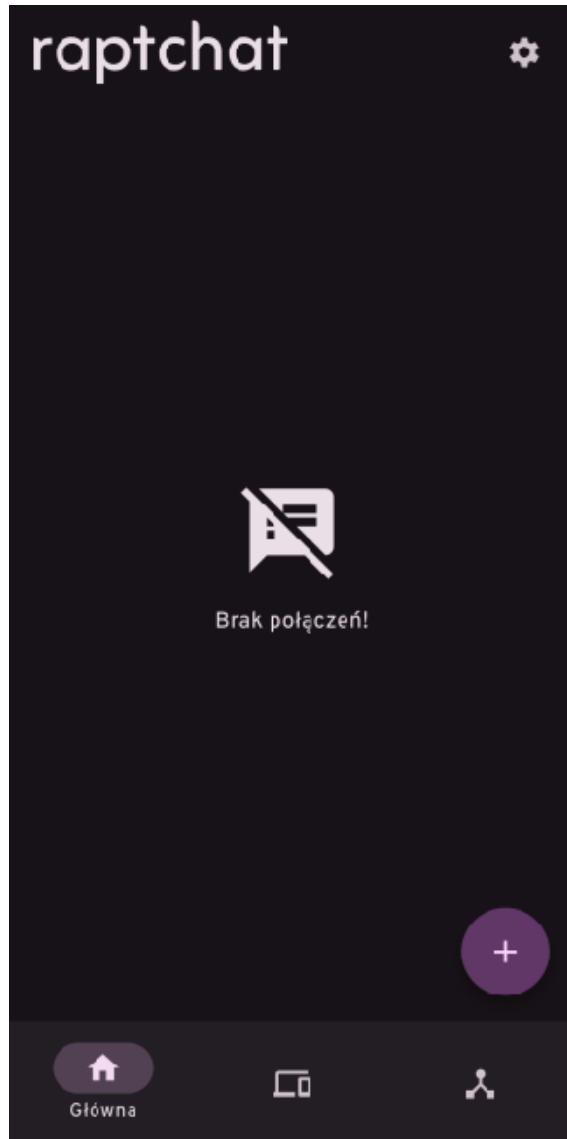
- zarządza wysyłaniem wiadomości i kolejką poleceń,
- cyklicznie odpytuje urządzenie o nowe wiadomości (komenda `flush`),
- aktualizuje interfejs użytkownika po nadaniu nowych danych.

3.3.6 Interfejs użytkownika

Aplikacja składa się z kilku ekranów:

- **Ekran główny** — lista zapisanych połączeń i dostęp do funkcji czatu.
- **Czat** — dwukierunkowa komunikacja tekstowa z danym połączeniem.
- **Mapa sieci** — wizualizacja połączonych węzłów mesh.
- **Edycja połączenia** — konfiguracja ID, klucza prywatnego i listy odbiorców.

- **Ustawienia** — język aplikacji, motyw kolorystyczny itp.



Rysunek 28: Ekran główny (tryb ciemny)

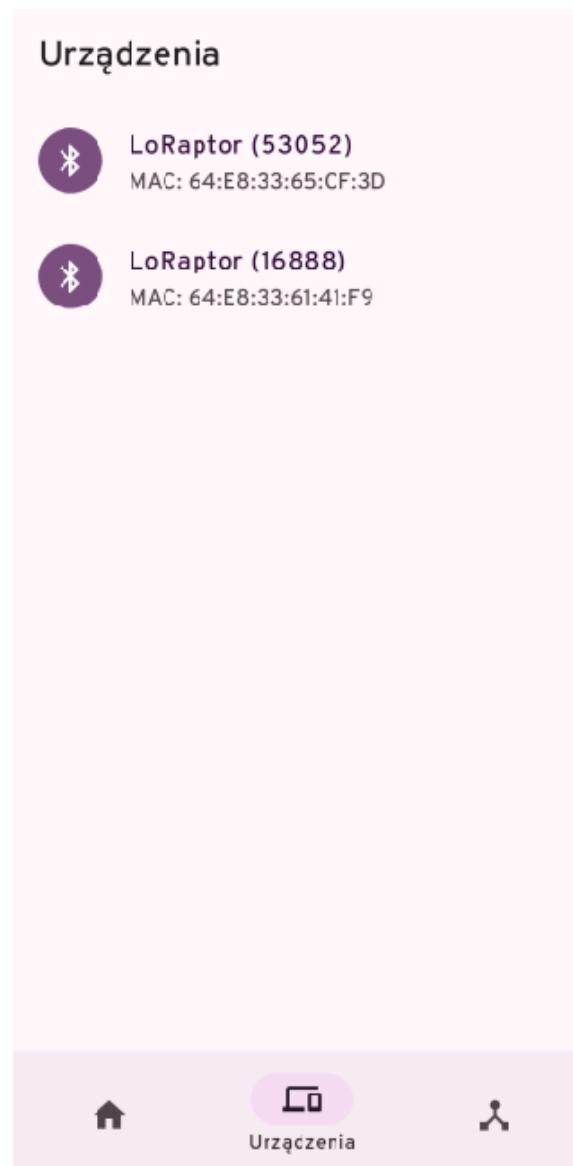


Rysunek 29: Ekran główny (tryb jasny)

Dokumentacja techniczna projektu "LoRaptor"

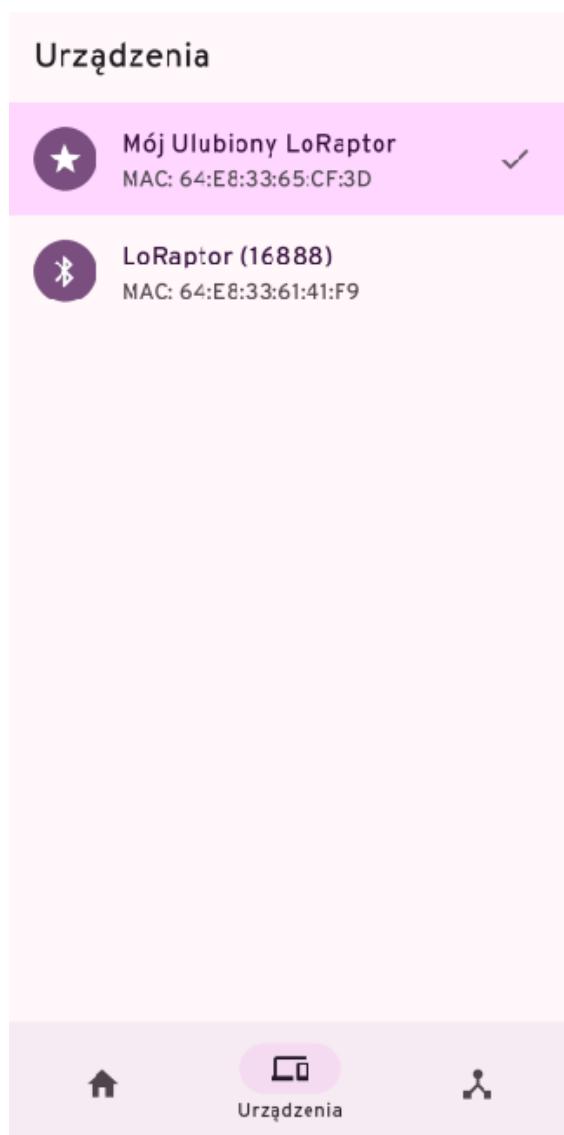


Rysunek 30: Panel ustawień



Rysunek 31: Widok urządzeń

Dokumentacja techniczna projektu "LoRaptor"

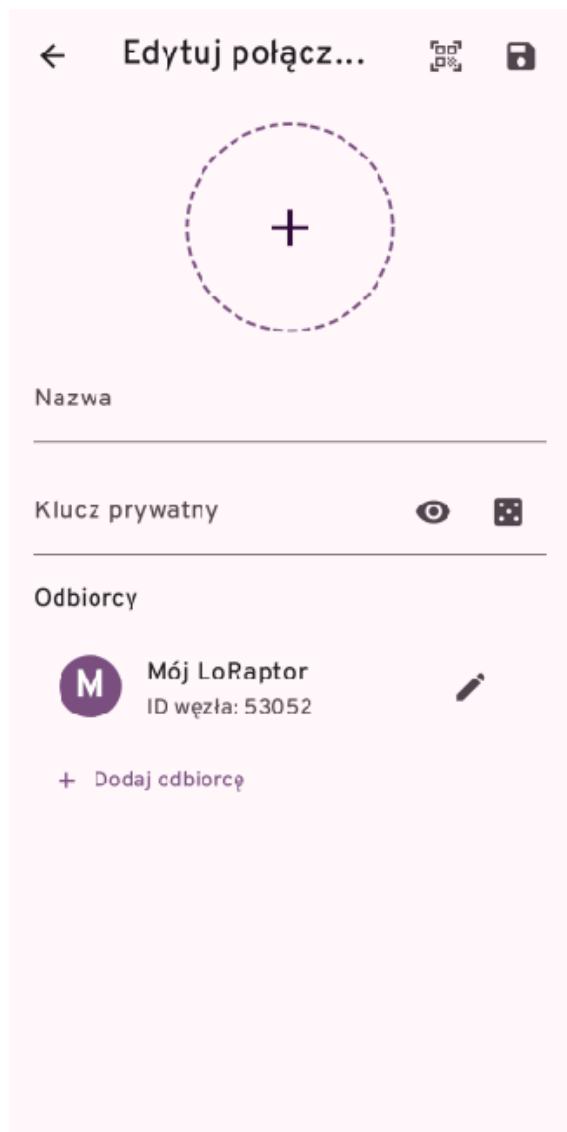


Rysunek 32: Widok urządzeń ze sparowanym LoRaptorem

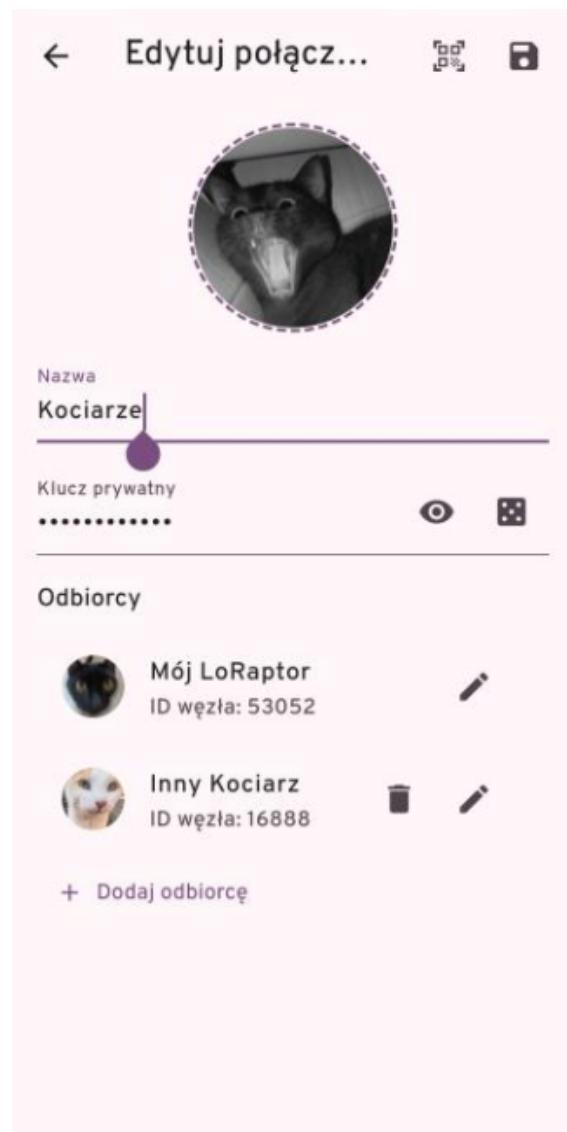


Rysunek 33: Szczegóły mojego urządzenia

Dokumentacja techniczna projektu "LoRaptor"



Rysunek 34: Tworzenie nowego połączenia



Rysunek 35: Uzupełnione dane połączenia

Dokumentacja techniczna projektu "LoRaptor"

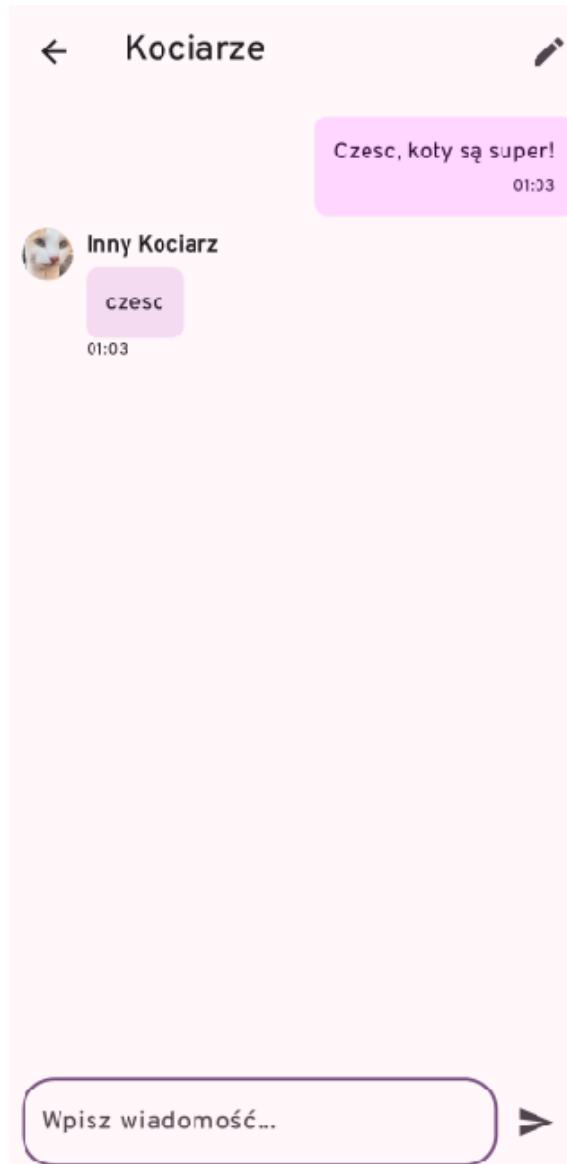


Rysunek 36: Lista aktywnych czatów



Rysunek 37: Pusty czat (nowa konwersacja)

Dokumentacja techniczna projektu "LoRaptor"



Rysunek 38: Konwersacja tekstowa (przykład 1)



Rysunek 39: Konwersacja tekstowa (przykład 2)



Rysunek 40: Kod QR do szybkiego dodania połączenia

3.3.7 Podsumowanie

RaptChat stanowi wygodny interfejs do zarządzania LoRaptorem, bez potrzeby używania terminala lub pisania skryptów. Dzięki rozbudowanej warstwie CLI wbudowanej w urządzenie, aplikacja zyskuje elastyczność, a użytkownik ma pełną kontrolę nad siecią i komunikacją.

4 Zalety i korzyści naszego rozwiązania

LoRaptor to nowoczesny system komunikacji bezprzewodowej, działający "off the grid" — poza zasięgiem tradycyjnych sieci, a jednocześnie "on the mesh", tworząc własną, odporną strukturę komunikacyjną. Główne korzyści to:

- **Zwiększone bezpieczeństwo:** Funkcjonowanie poza standardowymi sieciami Wi-Fi i GSM redukuje ryzyko awarii i cyberataków. System implementuje zaawansowane szyfrowanie AES-128 oraz efektywną kompresję danych.
- **Autonomiczność:** Skuteczne działanie w ekstremalnych warunkach środowiskowych — rozwiązanie dedykowane służbom ratunkowym, eksploatorom, jednostkom wojskowym i praktykom sztuki przetrwania.
- **Przyjazność użytkownikowi:** Bezproblemowa konfiguracja i zarządzanie poprzez dedykowaną aplikację RaptChat oraz zaawansowaną bibliotekę RaptorCLI.
- **Modułowość:** Zarówno komponenty sprzętowe, jak i oprogramowanie zaprojektowano z myślą o łatwej integracji z rozwiązaniami zewnętrznymi.
- **Przenośność:** Kompaktowa, odporna konstrukcja z wbudowanym lito-fanem oraz uniwersalnym portem USB-C zapewnia wygodę transportu i instalacji w różnorodnych lokalizacjach.

5 Konkurencyjność

LoRaptor wyróżnia się na tle istniejących rozwiązań dzięki unikalnemu połączeniu funkcji komunikacyjnych, energooszczędności i dostępności open-source. Podczas gdy inne systemy mesh są często ograniczone do zamkniętych ekosystemów (np. komercyjne sieci LoRaWAN), LoRaptor oferuje:

- **Brak opłat licencyjnych** — urządzenie działa bez potrzeby rejestracji w sieciach operatorów lub wykupywania planów taryfowych;
- **Pełna kontrola nad urządzeniem** — użytkownik ma dostęp do firmware'u, może go modyfikować, rozszerzać, tworzyć własne komendy i funkcje;
- **Wieloplatformowość** — dzięki BLE, USB i wsparciu dla różnych systemów (Android, Windows, Linux), LoRaptor integruje się z szeroką gamą sprzętów;
- **Niższy koszt produkcji jednostkowej** — szacowany koszt jednej jednostki (ok. 10-11 USD) pozwala wdrożyć system nawet przy ograniczonym budżecie, co daje przewagę nad droższymi, zamkniętymi urządzeniami;
- **Zaawansowana technologia w prostym opakowaniu** — LoRaptor łączy mikrokontroler ESP32-S3-MINI, mesh LoRa, szyfrowanie i CLI w jednym, przemyślanym projekcie.

6 Innowacyjność

Innowacyjność LoRaptora przejawia się nie tylko w jego funkcjach, ale także w podejściu do projektowania i rozwoju technologii:

Autorska biblioteka RaptorCLI Umożliwia tworzenie złożonych komend i interakcji tekstowych z urządzeniem w sposób przypominający klasyczne terminale systemów operacyjnych. To pozwala na elastyczne sterowanie urządzeniem bez konieczności pisania kodu.

Zastosowanie algorytmu SMAZ Do kompresji danych oraz AES-128 do szyfrowania — co w połączeniu daje wydajny i bezpieczny kanał komunikacji nawet przy ograniczonej przepustowości LoRa.

Zautomatyzowana konfiguracja Przez aplikację mobilną — użytkownik nie musi znać poleceń CLI ani procesów konfiguracyjnych. RaptChat prowadza je automatycznie, ustawiając czas, przywracając połączenia i wczytując konfiguracje.

Warstwa litofanowa w obudowie Zintegrowana z diodą RGB — to przykład połączenia estetyki z funkcjonalnością, gdzie design obudowy pełni również funkcję sygnalizacyjną.

Modularność i otwarta architektura Zarówno sprzęt, jak i kod źródłowy pozwalają na łatwe modyfikacje, co sprzyja innowacjom i wykorzystaniu w innych dziedzinach.

7 Plany na przyszłość

Projekt LoRaptor, choć obecnie w fazie w pełni funkcjonalnego prototypu, ma ambitne cele rozwojowe. Wśród planowanych kierunków rozwoju znajdują się:

Wersja terenowa z ekologicznym zasilaniem Ulepszona wersja z rozszerzoną pamięcią, modułem GPS oraz hybrydowym systemem zasilania (bateria + panel słoneczny), wykorzystująca rozwiązania energooszczędne z odnawialnymi źródłami energii, dostosowana do pracy w warunkach ekstremalnych i zapewniająca długotrwałą autonomię w terenie.

Rozbudowa aplikacji RaptChat Zaawansowane funkcje: mapa z geolokalizacją węzłów, wizualizacja aktywności sieci w czasie rzeczywistym, eksport i analiza danych komunikacyjnych.

Integracja z sieciami satelitarnymi Rozszerzenie zasięgu systemu poprzez integrację z konstelacjami satelitów LEO, umożliwiające komunikację globalną w obszarach pozabawionych jakiejkolwiek infrastruktury naziemnej.

Zastosowania humanitarne Adaptacja systemu do użytku w strefach konfliktów i katastrof naturalnych, ze szczególnym uwzględnieniem potrzeb komunikacyjnych w regionach dotkniętych działaniami wojennymi (np. Ukraina), zapewniająca niezależną i bezpieczną łączność dla służb ratunkowych i ludności cywilnej.

Certyfikacja sprzętowa Uzyskanie profesjonalnych certyfikatów bezpieczeństwa i niezawodności (IP67, MIL-STD-810G), umożliwiających wdrożenia w sektorach profesjonalnych: służbach ratunkowych, jednostkach wojskowych, logistyce terenowej oraz infrastrukturze krytycznej.

8 Gdzie nas można znaleźć?

Repozytorium kodu: <https://github.com/smegg99/LoRaptor>



*LoRaptor nie kończy się na prototypie — to fundament pod dynamiczny
i otwarty ekosystem przyszłościowej komunikacji.*