WYŻSZA SZKOŁA BANKOWA W GDAŃSKU
WYDZIAŁ EKONOMII I ZARZĄDZANIA W GDYNI

Łukasz Danilewicz
nr albumu 42491

PROJEKT I IMPLEMENTACJA URZĄDZENIA IoT – ZDALNIE STEROWANEGO ZEGARA Z FUNKCJAMI STACJI POGODOWEJ, MONITORINGU ŚRODOWISKA ORAZ EDUKACYJNO-ROZRYWKOWYMI

Praca inżynierska
na kierunku Informatyka

Praca napisana pod kierunkiem
dra inż. Rafała Grzeszczyka

Gdynia 2020

Spis treści

Wstęp....5
1. Metodyka zarządzania i specyfikacja wymagań projektu....7
1.1 Opis metodyki zarządzania projektem – Feature-Driven Development....7
1.2 Zbieranie, analiza i negocjacja wymagań projektu....10
1.3 Specyfikacja wymagań projektu....13
1.4 Schemat blokowy urządzenia i modelowanie systemu....17
2. Omówienie komponentów elektronicznych oraz technologii informatycznych....22
2.1 Wybór i charakterystyka układu mikrokontrolera....22
2.1.1 Wybór platformy sprzętowej oraz języka programowania....22
2.1.2 Charakterystyka platformy NodeMCU z mikrochipem ESP-12-E....24
2.1.3 Interfejsy komunikacyjne w mikrokontrolerach....27
2.1.3.1 Wejścia-wyjścia ogólnego przeznaczenia – GPIO....27
2.1.3.2 Protokół komunikacyjny i magistrala danych I2C....29
2.1.4 Odmierzanie czasu – oscylator kwarcowy....31
2.2 Komponenty i moduły elektroniczne wraz z bibliotekami....32
2.2.1 Wyświetlacz graficzny OLED 1.3"....34
2.2.2 Wyświetlacz LCD 4x20 znaków z modułem podświetlenia....34
2.2.3 Mikrofon elektretowy ze wzmacniaczem....35
2.2.4 7 kanałowy equalizer....35
2.2.5 Moduł czujnika natężenia światła....36
2.2.6 Pierścień 60 diod LED RGB....36
2.2.7 Moduł czujnika ruchu PIR....37
2.2.8 Moduł czujnika temperatury, wilgotności i ciśnienia....37
2.2.9 Moduł wykrywania gestów....38
2.2.10 Głośnik z interfejsem cyfrowym....38
2.2.11 Moduł przycisku....39
2.2.12 Moduł zegara czasu rzeczywistego RTC....39
2.2.13 Moduł czujnika czystości powietrza....40
2.3 Technologie informatyczne....40
2.3.1 Opis zintegrowanego środowiska programistycznego Arduino IDE....40
2.3.2 Wybrane protokoły komunikacyjne....43

2.3.2.1 Protokoły synchronizacji czasu NTP i SNTP....43
2.3.2.2 Protokół internetowy UDP....45
2.3.2.3 Protokół transmisji danych MQTT....47
2.4 Struktury danych – format wymiany danych JSON....49
3. Projekt i realizacja....52
3.1 Warstwa sprzętowa....52
3.1.1 Schemat elektryczny urządzenia....52
3.1.2 Fazy powstawania, projekt i wykonanie płytki PCB....56
3.1.3 Projekt i wykonanie obudowy....59
3.2 Warstwa oprogramowania – implementacja funkcjonalności....64
3.2.1 Inicjalizacja....64
3.2.2 Tryb konfiguracji i zarządzanie danymi konfiguracyjnymi....66
3.2.3 Urządzenia i strumienie wejścia/wyjścia oraz sterowanie zegarem....69
3.2.4 Tryb zegara....73
3.2.5 Tryb stacji pogodowej....77
3.2.6 Monitoring pomieszczenia i transmisja danych do brokera MQTT....82
3.2.7 Tryb efektów świetlnych....83
3.2.8 Tryb edukacyjny....86
3.2.9 Tryb budzika dźwiękowego i świetlnego....88
3.2.10 Aplikacja mobilna „Zegar Led”....90
3.2.10.1 Budowa i działanie aplikacji....90
3.2.10.2 Komunikacja aplikacji z zegarem....93
3.2.10.3 Tryb palety barw, preferencje użytkownika....95
3.2.10.4 Ustawienia – edycja parametrów konfiguracyjnych....96
3.3 Przeprowadzone testy oprogramowania....99
Zakończenie....102
Spis wykorzystanej literatury....105
Spis tabel....109
Spis rysunków....110
Spis oprogramowania wykorzystanego w projekcie i niniejszej pracy....113
Spis bibliotek programistycznych użytych w projekcie....114
Załączniki....115

Wstęp

Czas oraz jego upływ są względne. Wymiar czasowy jest nierozerwalnie związany z wymiarami przestrzennymi tworząc z nimi czasoprzestrzeń. Pomimo, iż w układzie odniesienia związanym z Ziemią, takie postrzeganie świata może wydawać nienaturalne, to - jak naukowo dowiedziono[1] - w tej i w szerszej perspektywie jest właściwie.

Jak odpowiedzieć dziecku na pytanie typu „Daleko jeszcze?", aby mogło zrozumieć odpowiedź? Czyż nie pojawia się pokusa, aby podać odległość **wymiarze czasowym**? Potrzeba odpowiedzi na to i szereg podobnych pytań, zadanych przez 5-letnie dziecko autora pracy, stało się inspiracją oraz motywacją do jej napisania i zrealizowania. Wstępne sposoby wytłumaczenia w postaci użycia miary i stopera w terenie, a nawet zbudowania prostego układu z przygasającą co sekundę diodą nie były satysfakcjonujące i musiały zostać znacznie rozszerzone.

W dobie ciągłego i intensywnego rozwoju technologii elektronicznej oraz masowej produkcji urządzeń, wybór elektroniki użytkowej na rynku lokalnym jaki i światowym jest bardzo bogaty. Kolejnym, prostym rozwiązaniem problemu mógł być zakup jednego z dostępnych zegarów elektronicznych. Okazało się jednak, że młody klient, postawił tak wyśrubowane wymagania, że żaden pojedynczy produkt nie był w stanie ich spełnić.

Głównym **celem pracy** było zebranie i zaimplementowanie wymagań prowadzących do realizacji krystalizującego się projektu - bezpiecznego, estetycznego i funkcjonalnego wyrobu – zdalnie sterowanego modułu **internetu rzeczy** IoT[2]. Produkt ten to **wielofunkcyjny zegar elektroniczny** przedstawiający animację upływu czasu, wyświetlający bieżący dzień oraz godzinę, aktualne i prognozowane warunki pogodowe, informujący o czasie zachodów słońca i fazie księżyca, a także generujący efekty świetlne w rytmie dźwięków z otoczenia. Inne wyróżniające funkcjonalności urządzenia to wyświetlanie i monitorowanie parametrów pomieszczenia i przesyłanie ich do zewnętrznego brokera MQTT[3], a także tryb edukacyjny z zagadkami

1 Halliday D. [i in.], Podstawy Fizyki, Wydawnictwo Naukowe PWN, Warszawa 2003, Tom 4, s.146-150

2 IoT – skrót od ang. Internet of Things, internet rzeczy

3 MQTT – skrót z ang. Message Queue Telemetry Transport, protokół transmisji danych

z rozpoznawaniem znaków, słów oraz podstawowymi działaniami matematycznymi. Maksymalny czas na realizację przedsięwzięcia określono na 9 miesięcy.

Praca została podzielona na trzy rozdziały, które w dużym stopniu odzwierciedlają fazy powstawania produktu końcowego. **W rozdziale 1** została zawarta krótka charakterystyka wybranej metodyki zarządzania projektami informatycznymi – *Feature-Driven Development* wraz z uzasadnieniem tego wyboru. Następnie przedstawiono specyfikację wymagań projektu, która posłużyła do utworzenia modelu urządzenia zarówno w wersji sprzętowej jaki i programowej.

Rozdział 2 opisuje fazę projektowania i przygotowania do realizacji projektu. Zawarto w nim opis podjęcia wielokryterialnej decyzji o wyborze platformy sprzętowej wraz z językiem programowania. Dokonano krótkiej charakterystyki układu NodeMCU ze szczególnym podkreśleniem wykorzystanych magistrali danych. W następnym podrozdziale scharakteryzowano moduły sprzętowe niezbędne do realizacji założeń, zgodnie z ustaloną wcześniej specyfikacją wymagań. Rozdział kończy się teoretycznym przybliżeniem zintegrowanego środowiska programistycznego Arduino IDE, opisem zaimplementowanych protokołów sieciowych UDP, NTP i MQTT oraz formatu wymiany danych JSON.

W ostatnim, **3 rozdziale**, opisano tworzenie rozszerzenia platformy sprzętowej o wybrane wcześniej elektroniczne moduły zewnętrzne, a także przedstawiono projekt płytki PCB oraz wizualizację obudowy. Następnie wyszczególniono i przedstawiono wszystkie tryby pracy zbudowanego urządzenia: tryb konfiguracji, budzika świetlnego, zegara i stacji pogodowej, efektów świetlnych w rytm muzyki, tryb edukacyjny (zagadki dla dzieci przez wyświetlenie losowej litery/znaku/obrazka/słowa) oraz paletę barw. W rozwinięciu funkcjonalności stacji pogodowej nakreślono sposób integracji urządzenia z API dostarczającymi dane pogodowe i dane o zanieczyszczeniach środowiska. W kolejnym etapie omówiono budowę i działanie aplikacji mobilnej służącej do zdalnego sterowania zegarem. Na zakończenie scharakteryzowano przeprowadzone testy oprogramowania na przestrzeni całego projektu.

W tym miejscu autor pracy chciałby podziękować starszej córce, inspiratorce projektu i osobie specyfikującej wymagania oraz żonie za doradztwo projektowe oraz okazaną cierpliwość.

1. Metodyka zarządzania i specyfikacja wymagań projektu

1.1 Opis metodyki zarządzania projektem – Feature-Driven Development

Aby osiągnąć postawiony w tytule pracy cel, mimo stosunkowo i pozornie niewielkiego zagadania, zastosowano systematyczne i zdyscyplinowane podejście do wytwarzania oprogramowania. Rolę menedżera projektu, głównego architekta oraz programisty pełnił autor pracy, w roli klienta została postawiona pozostała część jego najbliżej rodziny. Już wstępna koncepcja wskazywała na wielofunkcyjny charakter projektowanego zegara, co implikowało znaczny stopień złożoności i konieczność skutecznego i spójnego zarządzania całym projektem. Dodatkowo, proces powstawania urządzenia wraz z jego oprogramowaniem musiał dawać możliwość wymagającym klientom częsty wgląd w postęp prac, co w szczególności oznaczało możliwość regularnego zobrazowania jej rezultatów. Ten specyficzny i niełatwy układ spowodował, iż do zarządzania całym przedsięwzięciem wybrano zwinną metodykę programowania ***Feature-Driven Development (FDD)***, stworzoną w 1997r. przez Jeffa De Luca i Petera Coad'a[4].

FDD[5] to wysoce adaptacyjny proces tworzenia oprogramowania, charakteryzujący się licznymi, lecz krótkimi iteracjami (nie przekraczającymi dwóch tygodni), z naciskiem na jakość wytwarzanego produktu we wszystkich etapach jego kreacji. Metodyka pozwala zapewnić częste i namacalne wyniki prac, dając informacje o postępie przy niewielkich kosztach ogólnych i zakłóceniach dla programistów. Myślą przewodnią tej metodyki jest więc kierowanie rozwojem oprogramowania poprzez cechy (właściwości) opisujące funkcjonalności powstającego informatycznego produktu.

Wskazana metodyka w znaczącym stopniu pozwala złagodzić i rozwiązać najczęściej występujące problemy podczas wytwarzania oprogramowania:

a) komunikację (w szeroko rozumianym zespole),
b) złożoność zagadnienia,
c) jakość produktu (nie zawsze rozumianą wieloaspektowo).

4 http://www.nebulon.com/fdd/

5 Goyal Sadhna, Major Seminar On Feature Driven Development, Technical University Munich, WS 2007/08, s. 3-5

Niezbędne znaczne zaangażowanie użytkowników, w tym wypadku zamawiających, powoduje zbliżone rozumienie problemu przez nich oraz programistów. Minimalizowane są szanse na niejawne założenia czynione przez którąkolwiek ze stron, co ma zasadnicze znaczenie dla sukcesu przedsięwzięcia.

Dekompozycja domeny problemu na mniejsze zagadnienia, w szczególności, gdy są od siebie niezależne, pozwala na ich rozwiązanie w relatywnie krótkim czasie i zmniejsza ilość zbędnej komunikacji między uczestnikami projektu. Dodatkowo, wybrana metodyka dzieli projekt na takie iteracje, że czas pomiędzy analizą i testowaniem jest znacząco skrócony. Pozwala to na szybkie wykrycie błędów i drastycznie zmniejsza ewentualny koszt ich naprawy.

Percepcja jakości oprogramowania zależy od osób dokonujących oceny. Użytkownik, deweloper, menedżer czy zamawiający pod tym pojęciem będą rozumieli i wyliczali czasem znacząco różne cechy. Pojęcie jakości powinno być więc postrzegane spektralnie, łącząc w sobie jakość z perspektywy wewnętrznej oraz jakość z perspektywy zewnętrznej. Dokładnie takie podejście prezentuje metodyka *Feature-Driven Development* rozszerzając pojęcie jakości ponad tą rozumianą przez wytwórców oprogramowania jako łatwość zarządzania, ulepszania i zgodność ze standardami.

Głównym elementem wybranej metodyki jest termin cecha (ang. *feature*), stanowiąca niewielki fragment funkcjonalności produktu, jednakże przedstawiający dostrzegalną wartość dla klienta. Cechy są więc sposobem na wyrażenie klienckich wymagań funkcjonalnych[6]. Mogą być one grupowane w zbiory cech (ang. *features set*) i zostać zrealizowane w postaci odrębnych komponentów.

W metodyce FDD wyróżnić można 5 procesów składowych, które wraz z elementami wynikowymi zostały przedstawione na rysunku nr 1. Rozpoczynając od stworzenia wizji projektu, poprzez opisanie jego właściwości i ich ewaluację, prowadząc do planu wytwarzania oprogramowania opartego o funkcjonalności. Na jego podstawie tworząc projekt, który jest realizowany sukcesywną implementacją.

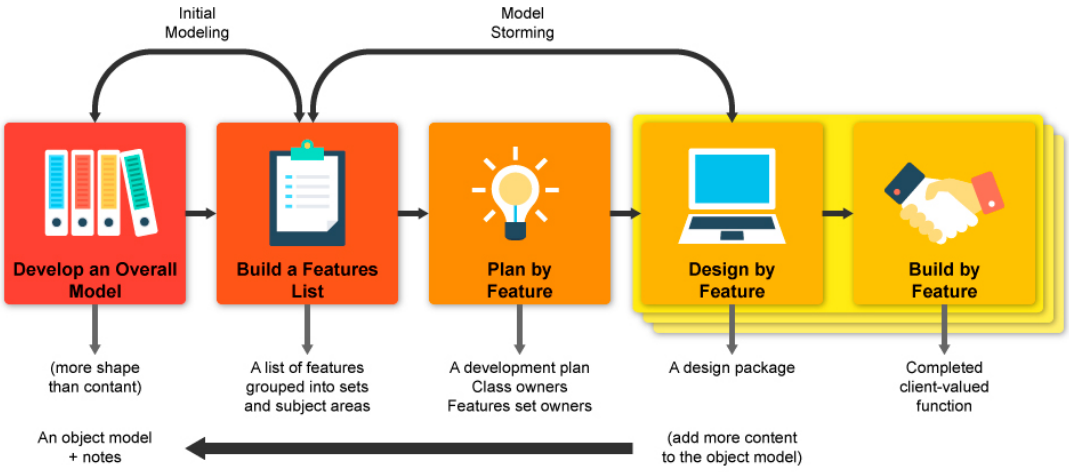
Podobnie do innych zwinnych metodyk, *Feature-Driven Development* poza ustalonym, procesem tworzenia oprogramowania, powinna opierać się o tak zwane najlepsze praktyki (ang. *best practices*)[7], które zostały wyszczególnione poniżej.

6 Rybus R., Feature Driven Development – lekka metodyka tworzenia oprogramowania, s.1

7 Goyal Sadhna, Major Seminar On Feature Driven Development, Technical University Munich, WS 2007/08, s. 6-10

Lista najlepszych praktyk programowania:

1. Modelowanie obiektowe, domenowe,
2. Rozwój przez funkcjonalności,
3. Przypisanie własności komponentu / klasy,
4. Zespoły przypisane do pożądanej funkcjonalności,
5. Inspekcje,
6. Zarządzanie konfiguracją (wersjonowanie kodu źródłowego, śledzenie artefaktów),
7. Regularne budowanie oprogramowania (zawsze dostępna działająca wersja),
8. Widoczność postępach i wynikach.



Rysunek 1: 5 procesów FDD wraz z ich produktami

Źródło: https://newline.tech/blog/feature-driven-development-methodology/

Poza zaletami wynikającymi z wybranej metodyki, dołożono starań, aby budowane urządzenie wraz z jego oprogramowaniem, posiadało cechy dobrego oprogramowania[8] tj. łatwość utrzymania (pielęgnacji), rzetelność (niezawodność w działaniu), efektywność oraz użyteczność. Cechy te stanowią podstawę inżynierii oprogramowania i znacząco zwiększają prawdopodobieństwo sukcesu na przestrzeni całego cyklu życia oprogramowania.

8 Sommerville I., Software Engineering, Ninth Edition, Addison Wesley, Boston 2011, s.12-13

1.2 Zbieranie, analiza i negocjacja wymagań projektu

Głównym celem przedsięwzięcia jest zbudowanie wielofunkcyjnego zegara. Na wstępnym etapie prac przeprowadzono szereg dyskusji w wyniku których wyłoniła się ogólna lista pożądanych funkcjonalności i cech produktu.

Przy każdym wymaganiu podano jego priorytet oraz symbol kryjący za sobą pomysłodawcę, w celu późniejszych konsultacji. Spisane funkcjonalności poddano analizie, po której zostały one przedstawione na osobnych kartkach formatu A4. Każda z nich stanowiła tak zwany przypadek użycia[9] spisany wraz ze swoim scenariuszem. Nieomal wszystkie scenariusze polegały na możliwości uruchomienia przez użytkownika wybranej funkcji i prezentacji wyników pomiarów lub animacji. Wyróżniającym się scenariuszem jest tryb konfiguracji, w którym przewidziano możliwość wprowadzenia danych konfiguracyjnych takich jak m. in. parametry do połączenia, godzina budzenia czy współrzędne geograficzne.

Odmiennie wykorzystano listę cech produktu. Część z wyszczególnionych cech przypisano do wymagań funkcjonalnych jako atrybuty przypadków użycia. Pozostałe dotyczące wyglądu lub budowy urządzenia stały się wytycznymi przy projektowaniu jego obudowy.

Kolejnym krokiem stało się doprecyzowanie każdego wymagania. Do opisu użyto prostego, spójnego języka, starając się przy tym uzyskać potrzebną szczegółowość tzn. precyzję w sformułowaniu wymagania. Następnie przeprowadzono negocjacje, po których część wymagań została zmodyfikowana i połączona, a niektóre zostały odrzucone jako nieuzasadnione lub niewykonalne.

Każde wymaganie zostało sprawdzone pod kątem możliwości realizacji na płaszczyźnie sprzętowej, co obejmowało sprawdzenie dostępności gotowych komponentów elektronicznych realizujących zadania pomiarowe lub wyświetlania, ich wymagania zasilania oraz protokoły komunikacyjne. Poza tym, w ramach sprawdzenia wykonalności całego przedsięwzięcia, rozpoczęto typowanie układu mikrokontrolera wyposażonego w moduł komunikacji bezprzewodowej WiFi, zdolnego do obsłużenia wszystkich wcześniej znalezionych i niezbędnych komponentów elektronicznych.

9 Sommerville I., Software Engineering, Ninth Edition, Addison Wesley, Boston 2011, s.106-108

W celu zobrazowania działań podjętych w ramach opracowywania wymagań projektu, przygotowano dwa zestawienia w postaci tabel, stanowiące niejako wyciąg ze stworzonych przypadków użycia – w przeważającym stopniu tożsamych w tym projekcie z funkcjonalnościami produktu.

Ostateczna lista funkcjonalności została przedstawiona w tabeli nr 1 stanowiąc wymagania funkcjonalne projektowanego zegara.

Tabela 1 - Lista funkcjonalności urządzenia po negocjacjach

Symb.	Wyszczególnienie funkcjonalności	Autor	Prio-rytet
F1	Zegar cyfrowy (HH:MM)	Ż	1
F2	Zegar analogowy (na okręgu z 60 diod LED) z animacją upływu czasu	A	2
F3	Monitoring środowiska w pobliżu urządzenia: temperatura, wilgotność względna, ciśnienie atmosferyczne, natężenie oświetlenia, stężenie związków organicznych; okresowe przesyłanie danych do zewnętrznego serwera MQTT	Ż, A	1
F4	Monitoring środowiska zewnętrznego i dane o ciałach niebieskich: temperatura, wilgotność względna, ciśnienie, zachmurzenie, wielkość opadów oraz zanieczyszczenia powietrza, godzina zachodu słońca, ilość dni do najbliższej pełni księżyca	Ż	2
F5	Efekty świetlne w rytm dźwięków z otoczenia, equalizer dla co najmniej 7 częstotliwości	C, A	1
F6	Budzik dźwiękowy	C	3
F7	Budzik świetlny	A	3
F8	Losowy generator znaków, słów oraz obrazków zwierząt - do nauki i rozpoznawania dla dzieci	C	1
F9	Paleta barw – możliwość wyświetlenia wybranego koloru na okręgu z diod LED	C	1
FK	Tryb konfiguracji do wprowadzenia danych umożliwiających nawiązanie połączenia z siecią komputerową	A	1
FP	Możliwość edycji parametrów konfiguracyjnych urządzenia przy użyciu dedykowanej aplikacji na telefon komórkowy	A	1
FZ	Zdalne sterowanie urządzeniem przy użyciu dedykowanej aplikacji na telefon komórkowy	A	2

Legenda: Ż-Żona, C-Córka, A-Autor

Ostateczna lista właściwości, stanowiąca niefunkcjonalne wymagania tworzonego produktu w warstwie sprzętowej i programowej, została przedstawiona w tabeli nr 2.

Tabela 2 - Lista cech urządzenia po negocjacjach

Symb.	Wyszczególnienie właściwości	Rodzaj	Atrybut funk.	Autor
C1	Front obudowa w kształcie okręgu, średnica 18±2cm, w kolorze czarnym	Konstrukcja	n. d.	Ż
C2	Nowoczesne wzornictwo, lekka i przezroczysta płyta czołowa, oszlifowane krawędzie	Konstrukcja	n. d.	Ż
C3	Niewielkie zużycie energii - do 10W	Budowa	n. d.	A
C4	Dopuszczalny jeden przewód doprowadzony do urządzenia, w kolorze czarnym (estetyka)	Konstrukcja	n. d.	Ż
C5	Przystosowanie do zawieszenia, stabilne zamocowanie do ściany	Konstrukcja	n. d.	A
C6	Bezdotykowa obsługa w zakresie wyboru trybu pracy urządzenia	Obsługa	F1-F8	A
C7	Konfiguracja przy pomocy smartfona lub komputera podłączonego do tej samej lokalnej sieci komputerowej co urządzenie	Obsługa	F6, F7, F8, FK	A
C8	Jasność diod zegara analogowego uzależniona od natężenia oświetlenia w pomieszczeniu	Budowa	F2	Ż
C9	Automatyczne ustawianie i ewentualna korekta czasu, brak potrzeby ręcznego ustawiania przez użytkownika np. po zaniku zasilania	Obsługa	F1, F2	Ż
C10	Dwa ekrany: pierwszy do ciągłego wyświetlania godziny i daty; drugi do danych z monitoringu środowiska oraz equalizera	Budowa	n. d.	A
C11	Zdalne sterowanie; możliwość wywoływania każdej funkcjonalności urządzenia z poziomu smartfona podłączonego do tej samej lokalnej sieci komputerowej	Obsługa	F1-F8	A
C12	Zapis parametrów konfiguracyjnych do pamięci nieulotnej	Obsługa	FK	A
C13	Podświetlenie ekranu LCD po wykryciu ruchu	Obsługa	F3,F4	Ż

Legenda: Ż-Żona, C-Córka, A-Autor; F1-F9, FK, FP, FZ – funkcjonalności z Tabela 1

Zastosowane i opisane w tym podrozdziale podejście podczas zbierania, analizy i negocjacji wymagań, prawidłowo wpisuje się w dobre praktyki inżynierii wymagań[10]. Mimo czasochłonności i pracochłonności interesariuszy etap ten okazał niezbędnym i kluczowym do skutecznej realizacji całego projektu.

1.3 Specyfikacja wymagań projektu

Celem projektu jest zbudowanie zegara cyfrowego z możliwością odczytu aktualnej daty i czasu z wyświetlacza w formacie „DD {nazwa miesiąca} YYYY" i HH:MM odpowiednio. Cyfry reprezentujące czas powinny być widoczne z co najmniej 3 metrów. Czas ma być wyświetlany w trybie ciągłym. Format czasu przyjmuje się na 24 godzinny. Urządzenie powinno wyświetlać aktualne informacje dnia rozumiane jako: data, godzina zachodu słońca, ilość dni do najbliższej pełni księżyca, temperaturę (w pomieszczeniu, zewnętrzną oraz zewnętrzną odczuwalną), wilgotność, ciśnienie oraz intensywność opadu. [F1]

Dodatkowo odczyt aktualnej godziny, minuty i sekundy powinien być umożliwiony przy wykorzystaniu 60 diod LED RGB rozmieszczonych na okręgu (symulacja końców wskazówek zegara analogowego). Upływ czasu powinien być animowany, a kolory świecących diod (wskazówek) ze zbioru: czerwony, zielony i niebieski. Czas powinien być ustawiany oraz korygowany automatycznie. [F2]

Niezmiernie ważne jest, aby opisane wyżej funkcjonalności działały równocześnie i włączały się samoczynnie po uruchomieniu zegara, a także po upływie 15 minutowego okresu bezczynności. Brak komend sterujących ma skutkować samoczynnym przełączeniem po pracy w dowolnym innym trybie.

Urządzenie powinno wyświetlać wartości parametrów środowiskowych: temperaturę (zakres pomiarowy co najmniej -10°C do 50°C, dokładność ±1°C), wilgotność względna (zakres pomiarowy co najmniej 10% do 80%, dokładność ±3%), ciśnienie (zakres pomiarowy co najmniej 900hPa do 1100hPa, dokładność ±1hPa), natężenie oświetlenia oraz stężenie lotnych związków organicznych TVOC[11] z pomieszczenia, w którym się znajduje. Wszystkie mierzone lokalnie wartości mają być regularnie raportowane do zdefiniowanego w ustawieniach, zewnętrznego brokera

10 Young R.,The Requirements Engineering Handbook, Artech House, Boston, 2004, s. 8-10

11 TVOC – skrót z ang. Total Volatile Organic Compound

MQTT do wykorzystania np. w celach informacyjnych lub do sterowania w systemie domu inteligentnego. [F3]

Tworzony układ powinien również przekazywać informację o warunkach środowiskowych na zewnątrz pomieszczenia (współrzędne geograficzne miejsca z ustawień użytkownika). Dotyczy to danych pogodowych: temperatury, temperatury odczuwalnej, ciśnienia, wilgotności względnej, prędkości wiatru, zachmurzenia, wielkości opadów, a także informacji o zanieczyszczeniach środowiska: SO_2, NO_2, O_3, PM10 oraz indeksu UV. Dane te powinny pochodzić z serwisu pogodowego umożliwiającego darmowe zapytania co najmniej 1 raz na 15 minut (96 razy na dobę). Parametry wejściowe to współrzędne geograficzne. Odgórnym założeniem jest przyjęcie jednostek metrycznych. Ewentualna konwersja jednostek powinna zachodzić przez odpowiednio sparametryzowane zapytanie do API[12] serwera pogodowego.[F4]

Wyświetlanie informacji, podczas pracy w dwóch wyżej wymienionych trybach, powinno trwać 5 minut, po czym ekran główny urządzenia (LCD) musi zostać wygaszony. Ponowne podświetlenie ekranu powinno nastąpić po wykryciu ruchu w pobliżu zegara. [C13]

Kluczową cechą projektowanego urządzenia elektronicznego ma być możliwość generowania efektów świetlnych w takt dźwięków z otocznia. Dźwięk powinien być zbierany przez dedykowany układ urządzenia - wyklucza się konieczność podłączenia dodatkowych przewodów sygnałowych. Efekty generowane na diodach LED powinny opierać się na animacji - ruchu punktów świetlnych - jak również na zmianie ich jasności, proporcjonalnie do zmierzonej amplitudy fali dźwiękowej na wybranych zakresach częstotliwości. Po czasie 3 minut efekty świetlne powinny zmieniać się w pętli w sposób losowy. W ramach trybu efektów świetlnych, niezależnie i równolegle od powyżej wymienionych, wymaga się, aby na ekranie LCD można było uzyskać graficzną reprezentację do 7 zakresów częstotliwości dźwięków z otoczenia – tzw. equalizer graficzny. [F5]

Użytecznymi i ważnymi funkcjonalnością zegara mają być tradycyjny budzik dźwiękowy oraz budzik świetlny, symulujący wschód słońca przez stopniowe rozświetlanie pomieszczenia. Wymaga się, aby można było ustawić dwie niezależne godziny budzenia – jedną aplikowaną w dni powszednie, drugą zaś dotyczącą soboty

12 API – skrót z ang. application programming interface, interfejs programistyczny aplikacji

i niedzieli. Czas rozświetlania budzika świetlnego powinien być możliwy do zmiany przez użytkownika, z domyślną wartością 15 minut. Dopuszczalne wartości z przedziału <0;60> minut. Każdy z dwóch rodzajów budzika powinien być globalnie możliwy do włączenia/wyłączenia. [F6, F7]

Podstawowe sterowanie urządzeniem w zakresie zmiany trybu pracy i wyświetlania kolejnych ekranów informacyjnych powinno odbywać się bezdotykowo przez gesty wykonywane przed urządzeniem (do 20 cm). W przypadku problemów ze sterowaniem gestami powinno się zapewnić dodatkową możliwość sterowania przez przycisk na obudowie urządzenia. Intuicyjna i przyjazna użytkownikowi konfiguracja zegara, polegająca na edycji parametrów tekstowych, ma się odbywać się przez dedykowaną aplikacją w systemie Android w wersji co najmniej 4.4 (API *level* 19)[13]. Ze względu na oszczędność energii i ograniczenie zbędnej emisji, karta radiowa WiFi ma być włączana automatycznie po uruchomieniu zegara na okres 15minut, a także na żądanie poprzez wykonanie ustalonego gestu przed urządzeniem. W drugim przypadku wymaga się, aby połączenie z lokalną siecią komputerową było aktywne przez 30 minut od włączenia lub od czasu wydania ostatniej zdalnej komendy. Niezależnie od aktualnego trybu w jakim pracuje urządzenie, konieczne jest unaocznienie aktywności lub braku aktywności połączenia WiFi, co ma być jednoznaczne z możliwością zdalnego sterowania. [FP, C6, C11,C7]

Aplikacja mobilna powinna umożliwiać zmianę trybu pracy zegara. Poszczególne tryby powinny być oznaczone adekwatnymi do funkcjonalności ikonami w celu zwiększenia intuicyjności obsługi. W trybie efektów świetlnych oraz edukacyjnym powinna być wyświetlana czytelna, przewijana lista wszystkich możliwości wyboru wraz ich krótkim opisem. Parowanie aplikacji z zegarem powinno być trwałe i nieskomplikowane, tzn. przeprowadzone przez jasno sformułowane komunikaty. [FP]

Istotną funkcją urządzenia ma być generowanie losowego napisu na ekranie celem rozpoznania i odgadnięcia przez dziecko. Tryb losowania powinien być animowany celem zwiększenia atrakcyjności. Powinno się umożliwić wybór stopnia trudności zagadek poprzez zwiększanie ilości podzbiorów, z których losowany jest znak: wielkie litery, małe litery, cyfry, znaki specjalne oraz krótkie słowa (do 5 liter).

13 https://developer.android.com/studio/releases/platforms

Wyzwalanie losowania w trybie edukacyjnym ma być możliwe z poziomu aplikacji w urządzeniu zewnętrznym oraz przez gesty wykonywane przed zegarem. [F8]

Funkcjonalnością, która otrzymała najwyższy priorytet jest tzw. paleta barw. Wybór barwy ma być realizowany na kolorowym kole dostępnym w aplikacji mobilnej. Dla barwy wybranej z palety kolorów, powinno dać się ustawić jasność w przedziale 0 – 100%. Zmiana koloru lub jego jasności powinna być niezwłocznie odzwierciedlona na urządzeniu przez odpowiednie zapalenie wszystkich 60 diod. Ostatnio wybrany kolor powinien być zapamiętany w aplikacji mobilnej i wyświetlany przy kolejnym uruchomieniu trybu palety barw tak w aplikacji jak i na zegarze. [F9]

Należy przewidzieć możliwość przyszłej rozbudowy urządzenia na płaszczyźnie sprzętowej np. o dodatkowy czujnik lub diody LED do 10szt jak również na płaszczyźnie informacyjnej (np. nowe źródło danych, dodatkowe zmienne środowiskowe). W ramach zapewnienia łatwej możliwości ewolucji oprogramowania wymagana jest dostępność portu do programowania układu mikrokontrolera na zewnątrz obudowy. W razie niemożliwości spełnienia tego warunku, port ten powinien łatwo dostępny po otwarciu obudowy lub główny układ urządzenia ma mieć możliwość wymontowania bez konieczności rozlutowywania.

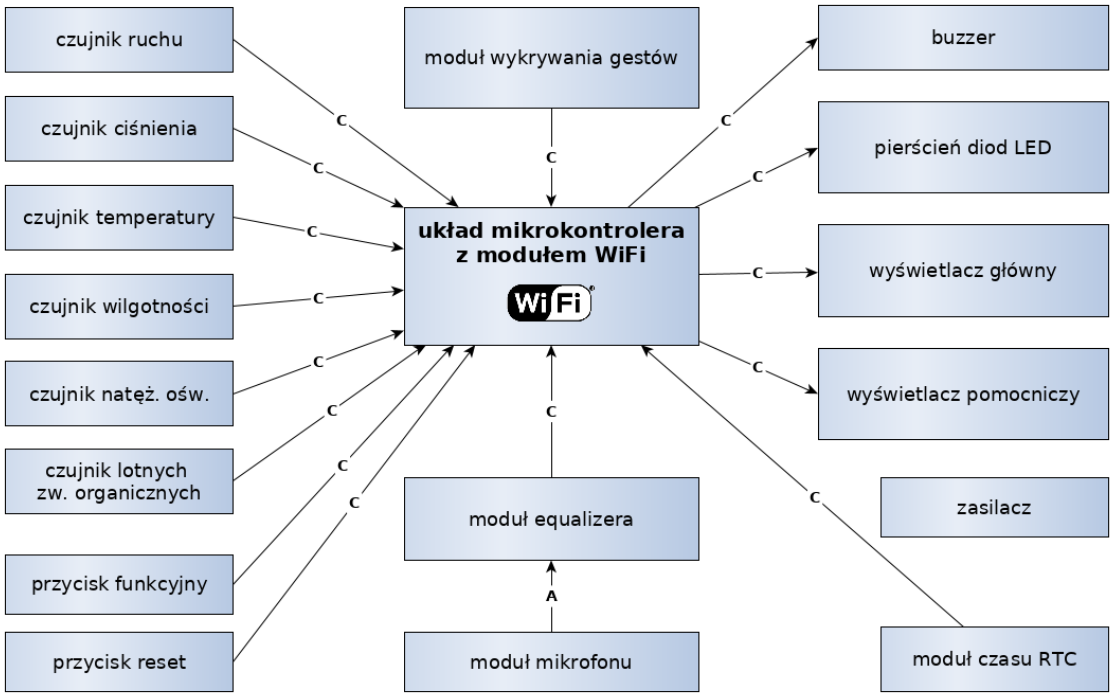
Urządzenie powinno mieć możliwość komunikacji bezprzewodowej z lokalną siecią komputerową i dostęp do internetu celem aktualizacji danych pogodowych i informacji o zanieczyszczeniach. Niezmiernie ważne jest, aby do urządzenia był doprowadzony tylko jeden przewód (zasilający) w kolorze czarnym. Zasilanie napięciem bezpiecznym z zewnętrznego zasilacza sieciowego. [C4]

Obudowa zegara ma mieć kształt koła o średnicy 18±2cm z możliwością zawieszenia na ścianie. Produkt powinien charakteryzować się nowoczesnym wzornictwem tj. przezroczysta płyta czołowa z oszlifowanymi krawędziami. Pozostała część obudowy ma być wykonana w kolorze czarnym. Maksymalnie dopuszczalna grubość urządzenia to 4cm. [C1,C2,C5]

1.4 Schemat blokowy urządzenia i modelowanie systemu

W tym stadium projektu, przeanalizowano specyfikację wymagań, co pozwoliło stworzyć modele urządzenia w przekroju sprzętowym jak również oprogramowania. W celu przedstawienia struktury urządzenia wykonano schemat blokowy, którego obiektami są moduły elektroniczne realizujące wyspecyfikowane wcześniej zadania. Dodatkowo, na schemacie przedstawionym na rysunku nr 2, uwidoczniono rodzaj i kierunek sygnału przenoszącego informacje dla każdego z komponentów w stosunku do centralnego układu mikrokontrolera z modułem WiFi.

Zwrócono szczególną uwagę na tor przetwarzania dźwięku, gdyż już na etapie oceny możliwości wykonania projektu, nie znaleziono pojedynczego komponentu do jego realizacji. Zaplanowano więc jego projekt i budowę za pomocą modułu mikrofonu oraz układu scalonego z funkcjonalnością korektora graficznego.



Rysunek 2: Schemat blokowy projektowanego urządzenia

Legenda: C – sygnał cyfrowy, A – sygnał analogowy

Źródło: opracowanie własne

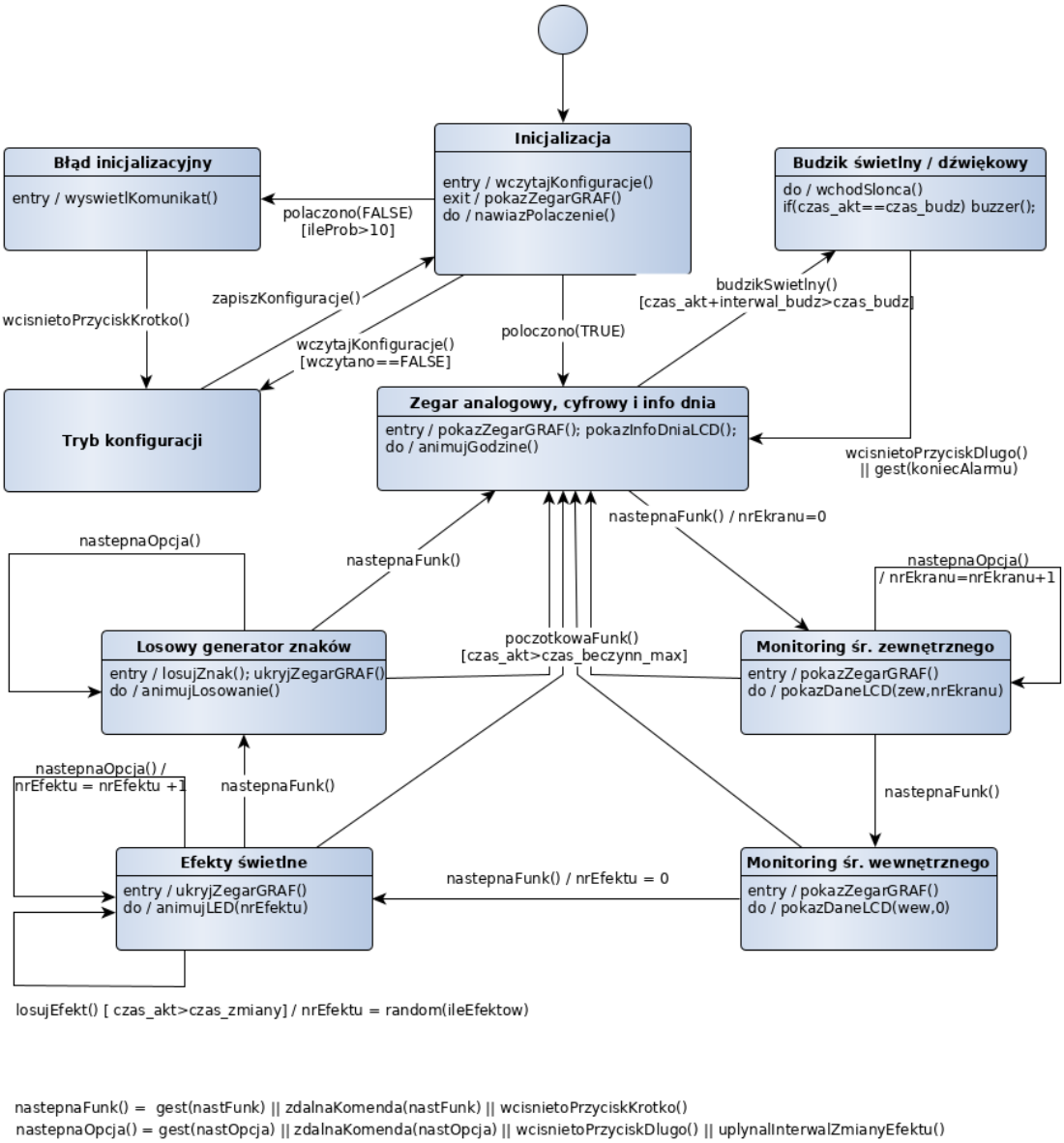
Kolejny etap, znacznie zbliżający ludzkie postrzeganie działania programu z jego realizacją w postaci kodu źródłowego, opierał się na zastosowaniu ujednoliconego języka modelowania systemów informatycznych UML[14]. Język ten ewoluuje już od 1994r. i dzięki swej semantyce pozwala opisywać szeroki zakres problemów. Jest elastyczny, pozwalając obok systemów informatycznych modelować również systemy biznesowe, a dodatkowo, wykazuje niezależność od konkretnego języka programowania. Z licznego zbioru dostępnych w UML rodzajów diagramów, adekwatnymi i wystarczającymi w bieżącym projekcie okazały się diagram maszyny stanowej oraz diagram klas.

Graficzna reprezentacja możliwych stanów tworzonego programu, pozwoliła na jego szerszą percepcję i stworzenie logicznie spójnego układu powiązań. Umożliwiła ona doprecyzowanie warunków przejść między stanami, a także odkryła wcześniej nieujawniony stan związany z niewłaściwą inicjalizacją urządzenia. Jest to bardzo istotny wyjątek do obsługi, gdyż jego wystąpienie nie pozwalałoby korzystać z podstawowych funkcji urządzenia dotyczących daty i czasu oraz wyświetlania danych dotyczących środowiska zewnętrznego.

Istotną zmianą, wynikającą z analizy powstającego modelu stanów, stało się dodanie do diagramu stanów obiektu reprezentującego tryb konfiguracji. W tym trybie powinna być możliwość bezpośredniego połączenia z urządzeniem i wprowadzenia przez użytkownika danych do połączenia z lokalną siecią bezprzewodową WiFi tzn. loginu (identyfikatora sieci SSID) oraz właściwego hasła. Tryb konfiguracji był nie wprost wymieniony pod symbolem [C7] i w rzeczywistości musiał stać się odrębną funkcjonalnością. Diagram stanów przedstawiono na rysunku nr 3.

W przeciwieństwie do diagramu stanów oddającego dynamikę tworzonego oprogramowania, diagram klas miał na celu oddanie jego statycznego charakteru. Ze względu na ograniczoną pamięć mikrokontrolera, zaplanowano zaprogramować go w stylu proceduralnym. Jednakże, ze względu na przetwarzanie licznych wielkości fizycznych (pogodowe, środowiskowe) wraz z ich wartościami aktualnymi i prognozowanymi, zdecydowano się na utworzenie klasy dedykowanej wielkościom fizycznym. Ułatwiło to przetwarzanie tych danych oraz ich przechowywanie w strukturze podczas działania programu. Strukturę tej klasy przedstawia rysunek nr 4.

14 Wrycza S., Marcinkowski B., Wyrzykowski K., Język UML 2.0 w modelowaniu systemów informatycznych, Wydawnictwo Helion, Gliwice, 2005, s.17-18

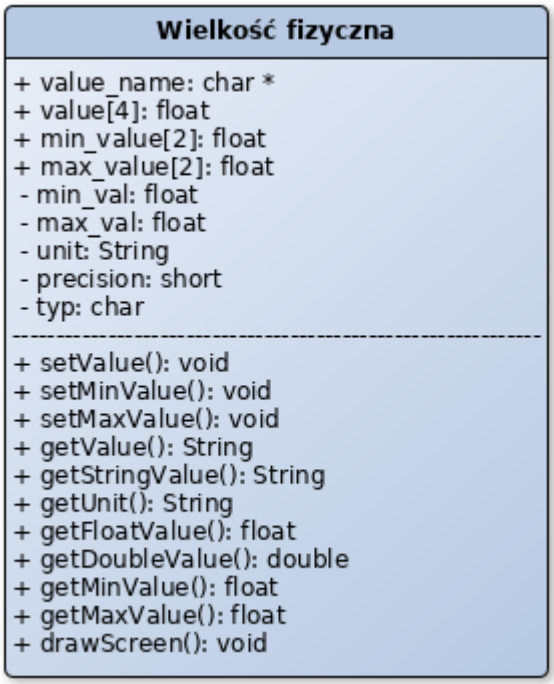


Rysunek 3: Diagram stanów - modelowanie historii życia programu

Źródło: opracowanie własne

Atrybuty klasy „Wielkość fizyczna” to kolejno: nazwa wielkości fizycznej (`value_name`), wartość poprzednia, wartość aktualna i dwie prognozowane (tablica `value`), minimalne wartości prognozowane (tablica `min_value`), maksymalne wartości prognozowane (tablica `max_value`), minimalnie dopuszczalna wartość do osiągnięcia (`min_val`), maksymalnie dopuszczalna wartość do osiągnięcia (`max_val`), jednostka wielkości fizycznej (`unit`), dokładność (rozumiana jako ilość miejsc po przecinku) (`precision`) oraz kategoria wielkości ze zbioru: {p - pogoda, z - zanieczyszczenie}.

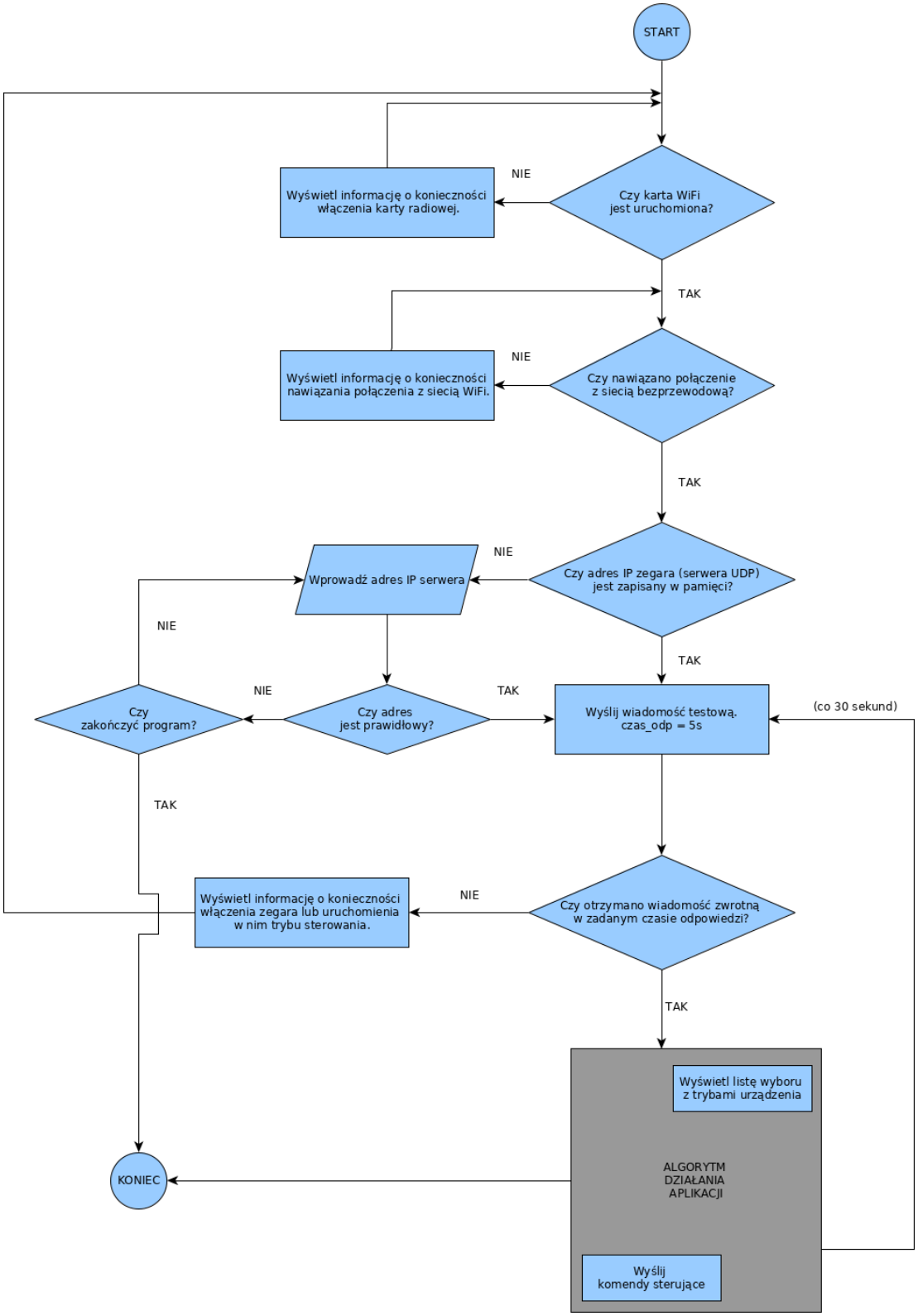
Metody klasy mają pozwalać na nadanie wartości aktualnej (`setValue`) i dopuszczalnych najmniejszej i największej (`setMinValue, setMaxValue`) wartości wielkości fizycznej, pobranie wartości z jednostką (`getStringValue`), pobranie tylko wartości (`getValue`) lub tylko jednostki (`getUnit`). Poza tym umożliwiają uzyskanie dostępu do wartości w postaci liczb zmiennoprzecinkowych (`getFloatValue`, `getDoubleValue`) oraz dostęp do prywatnych atrybutów z wartością minimalną (`getMinValue`) i maksymalną (`getMaxValue`). Ostatnia metoda (`drawScreen`) służy do wyświetlenia na ekranie głównym (LCD) zbiorczych informacji o danej wielkości fizycznej: nazwy, jednostki, wartości aktualnej i dwóch prognozowanych (+4h, +24h).



Rysunek 4: Struktura klasy *Wielkość fizyczna*

Źródło: opracowanie własne

Dla aplikacji mobilnej, w celu zapewnienia zdalnego sterowania zegarem, szczególnie istotna jest obsługa połączeń sieciowych. W tym celu stworzono algorytm uruchomienia aplikacji, którego fazą docelową jest wyświetlenie listy wyboru trybów pracy zegara. Poza głównym zadaniem, polegającym na nawiązaniu połączenia aplikacji z zegarem, algorytm uwzględnia regularne sprawdzanie jego dostępności. Algorytm został przedstawiony na rysunku nr 5.



Rysunek 5: Algorytm uruchamiania aplikacji mobilnej i kontroli połączenia

Źródło: opracowanie własne

2. Omówienie komponentów elektronicznych oraz technologii informatycznych

2.1 Wybór i charakterystyka układu mikrokontrolera

2.1.1 Wybór platformy sprzętowej oraz języka programowania

W celu integracji i obsługi wyspecyfikowanych w poprzednim rozdziale modułów elektronicznych (zgodnie ze schematem blokowym z rysunku nr 2) oraz zapewnienia wymaganej łączności z internetem, rozpoczęto analizę specyfikacji dostępnych na rynku układów elektronicznych obsługujących komunikację radiową WiFi. Mając na uwadze łatwość rozpoczęcia programowania tzn. dostępną literaturę, brak zewnętrznego programatora, dostępność bezpłatnego zintegrowanego środowiska programistycznego oraz cenę, jako główny komponent elektroniczny projektu postanowiono wybrać układ z popularnej[15] oraz bogatej rodziny platform sprzętowych opartych o moduł WiFi ESP8266-12E[16].

Ta wstępna decyzja została podjęta na podstawie warunków koniecznych przedstawionych w postaci poniżej wymienionych kryteriów i wynikających z analizy wykonalności projektu (2, 3, 5, 6, 7, 8) oraz jego specyfikacji wymagań (1, 4):

1. wbudowany moduł WiFi spełniający standardy IEEE 802.11 (opisujące warstwy fizyczną i MAC lokalnych sieci bezprzewodowych)[17],
2. zasilanie w zakresie napięć 3.3V-12V,
3. logika układu na poziomie 3.3V lub 5V,
4. dostępność co najmniej 7 portów wejścia/wyjścia ogólnego zastosowania (GPIO),
5. bezpośrednie programowanie przez port USB w układzie,
6. dostępność co najmniej jednego przetwornika analogowo-cyfrowego ADC,
7. dostępność magistrali I^2C,
8. raster wyprowadzeń: 2.54 mm.

15 https://circuitdigest.com/article/top-hardware-platforms-for-internet-of-things-iot
16 AI-Thinker team, ESP-12E WiFi Module, Version1.0, s. 3-6
17 http://www.ieee802.org/11/

Spośród wielu dostępnych na rynku układów ze zintegrowanym modułem WiFi ESP8266-12E należało dokonać optymalnego wyboru. Wytypowano 5 układów[18] spełniających warunki konieczne (nazwa: wielkość pamięci flash, cena, czas dostawy, ilość dostępnych GPIO, wymiary płytki):

A) WiFi SparkFun ESP8266: 512 KB, 69zł, 2dni, 3xGPIO, 50 x 26 mm
B) NodeMCU v3 (WiFi ESP8266): 4096KB, 35zł, 2dni, 10xGPIO, 58 x 30 mm
C) LinkIt Smart 7688 Duo: 32MB, 125zł, 2dni, 27xGPIO, 61 x 26 mm
D) Adafruit Feather Huzzah ESP8266: 4096 KB, 2dni, 110zł, 9xGPIO, 51 x 23 mm
E) D1 R2 WiFi ESP8266: 4096KB, 50zł, 2dni, 11xGPIO, 69 x 54 mm

Powstałe zagadnienie wielokryterialne rozwiązano stosując wagi dla kryteriów i obliczając sumę ważoną. Zestawienie kryteriów z wagami przedstawiono poniższej tabeli nr 5. Uzyskany tą metodą wynik nie jest obiektywnie najlepszy, jednak wskazuje najlepsze rozwiązanie w kontekście przyjętych kryteriów i ich ważności, będących wyrazem preferencji ich autora[19].

Tabela 3 - Wagi dla kryteriów przy wyborze platformy sprzętowej

LP.	Kryterium	Wart. pożądana	Jednostka	Waga
1	Cena z dostawą	mniej	zł	4
2	Wielkość pamięci flash	więcej	kB	3
3	Ilość dostępnych portów GPIO	więcej	szt	2
4	Czas dostawy	mniej	dni	1
5	Powierzchnia płytki	mniej	cm^2	1

W podsumowaniu, zdecydowanie najlepszy wynik uzyskała platforma sprzętowa **NodeMCU v3,** co stanowiło racjonalną podstawę do jej użycia w projekcie. W największym stopniu przyczyniła się do tego najniższa cena z dostawą z Polski na poziomie około **35zł** przy czasie dostawy 2dni, a także wielkość dostępnej pamięci Flash 4096 kB. W tym miejscu warto nadmienić, że na znanych chińskich platformach aukcyjnych **NodeMCU v3** jest dostępny od **9zł** z wysyłką przy około 30 dniowym czasie dostawy.

18 Jayakumar M., The inernet of things with esp8266 Hands on approach, CreateSpace Independent Publishing Platform, 2017r, s. 40-43
19 http://www.cs.put.poznan.pl/mmika/Optymalizacja%20wielokryterialna.pdf s. 1-10

Wybrany układ elektroniczny może być zaprogramowany w:

a) języku Lua (zaimplementowanym jako biblioteka języka C)[20]
b) odmianie języka C zaimplementowanej w zintegrowanym środowisku programistycznym (Arduino IDE)[21],
c) w języku JavaScript (w stylu zorientowanym na zdarzenia – *event driven*) przy użyciu środowiska Espruino[22]

Ze względu na doświadczenie programistyczne autora pracy pierwszym i naturalnym wyborem była opcja c). Jednakże, po analizie możliwości poszczególnych języków i dostępności bibliotek programistycznych do wyspecyfikowanych modułów, do implementacji projektu został wybrany wariant b) – **Arduino C**[23] – niepełna, lecz wystarczająco funkcjonalna implementacja **języka ANSI C**.

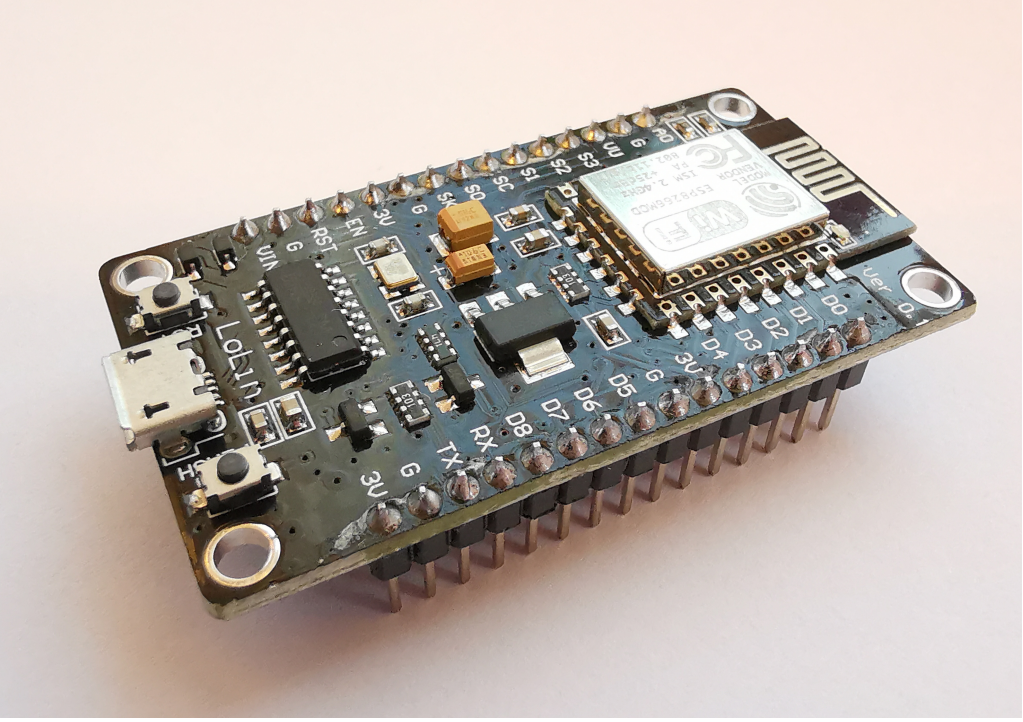
C jest proceduralnym językiem programowania wysokiego poziomu stworzonym przez Dennisa Ritchie w 1972r. Głównym jego przeznaczeniem było tworzenie systemów operacyjnych, jednakże z upływem czasu znalazł on szersze zastosowanie. Używa się go również do programowania mikrokontrolerów oraz systemów wbudowanych. Głównymi cechami tego języka jest niskopoziomowy dostęp do pamięci, prosty zestaw słów kluczowych oraz przejrzysty styl. Jego niepodważalne zalety to wydajność, przenośność, moc i elastyczność oraz cechy użytkowe [24].

2.1.2 Charakterystyka platformy NodeMCU z mikrochipem ESP-12-E

Platforma sprzętowa NodeMCU, przedstawiona na rysunku nr 6, może być traktowana jako zestaw rozwojowy dla mikrochipa ESP-12-E z rodziny ESP8266. Jego pierwsza wersja została wytworzona w 2014r. przez szanghajską firmę Espressif Systems[25]. Układ ten cechując się wysoką wydajnością i wysoko zintegrowaną budową (*system on chip*, tzw. SoC) został zaprojektowany jako rozwiązanie do komunikacji

20 https://www.lua.org/about.html
21 https://www.arduino.cc/en/main/FAQ
22 https://www.espruino.com/
23 Purdum J., Beginning C for Arduino, Apress Media, New York, 2012, s. 1-5
24 Prata S., Szkoła programowania. Język C. wyd 5, Helion, Gliwice, 2006, s. 2-4
25 UpSkill Learning, ESP8266: Programming NodeMCU Using Arduino IDE, CreateSpace Independent Publishing Platform, 2016r., s. 18-19

bezprzewodowej, gdzie pobór energii oraz zajmowana przestrzeń mają krytyczne znaczenie. Układy z serii ESP8266 mogą być używane samodzielnie i zaprogramowane jako mikrokontrolery z funkcjonalnością komunikacji bezprzewodowej. Drugą możliwością jest praca jako zewnętrzny moduł WiFi, sterowany odrębnym mikrokontrolerem za pomocą układu scalonego UART przy wykorzystaniu standardowego zestawu komend AT.



Rysunek 6: Platforma sprzętowa NodeMCU z ESP-12-E

Źródło: Opracowanie własne

Wybrane, kluczowe cechy sprzętowe charakteryzujące mikrochipy z rodziny ESP8266 przedstawiają się następująco:[26] [27]

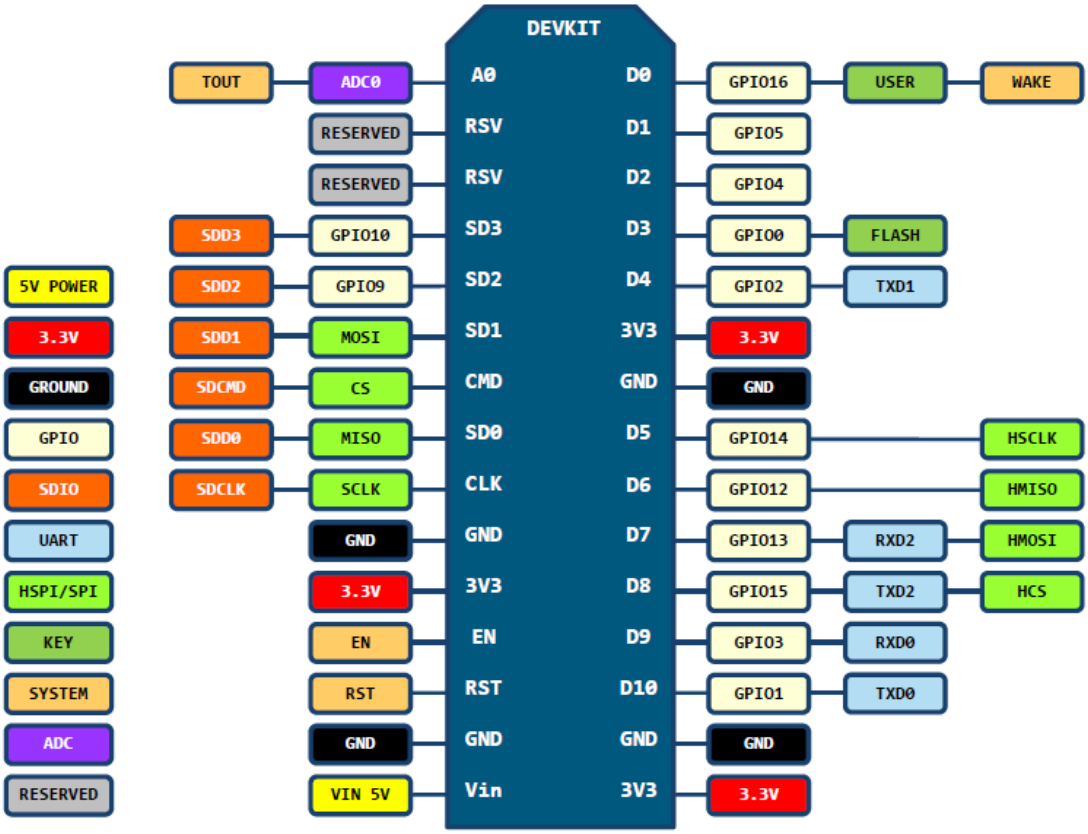
- 32-bitowy procesor RISC[28] taktowany zegarem 80MHz (Tensilica L106), z możliwością użycia jako procesor wgranego programu,
- komunikacja Wi-Fi 2.4GHz (sieci otwarte oraz zabezpieczone w standardzie szyfrowania WPA/WPA2),
- praca w standardach 802.11 b/g/n,

26 Espressif Systems, ESPRESSIF SMART CONNECTIVITY PLATFORM: ESP8266, https://espressif.com, 2013r., s. 5-6

27 Kolban N., Kolban's Book on the ESP32 & ESP8266, http://neilkolban.com/tech/esp8266/ s. 30

28 RISC – skrót z ang. Reduced Instruction Set Computing

- technologia bezpośredniego połączenia Wi-Fi Direct oraz tworzenia punktu dostępowego soft-AP,
- zaimplementowany stos TCP/IP,
- protokoły komunikacyjne SPI, I^2C i UART,
- pobór mocy w stanie spoczynku około 1mW,
- pobór prądu w zakresie 10uA (głębokie uśpienie) – 170mA (w trakcie transmisji),
- czas od wznowienia pracy z uśpienia do wysyłki pakietu 2ms,
- zintegrowany 10-bitowy przetwornik analogowo-cyfrowy,
- temperatura pracy od -40°C do 125°C,
- napięcie zasilania i logika 3.3V.



Rysunek 7: Porty ESP-12-E zmapowane na płytce NodeMCU

Źródło: https://github.com/nodemcu/nodemcu-devkit-v1.0

Omawiany mikrochip posiada 16 linii GPIO, z czego część jest używanych wewnętrznie w SoC do komunikacji m. in. z pamięcią flash. To powoduje, że do konfiguracji pozostaje 11 linii wejść/wyjść. Uwzględniając fakt, że dwa kolejne piny są zarezerwowane do linii transmisji TX oraz odbioru RX i używane do komunikacji np. z komputerem PC, ostatecznie do oprogramowania i wykorzystania zostaje 9 GPIO, oznaczonych w NodeMCU symbolami od D0 do D8. Z racji, że wyprowadzenia chipa i płytki rozwojowej nie są tożsame, na powyższej ilustracji nr 7 przedstawiono mapowanie pinów ESP-12-E z wyprowadzeniami płytki PCB.

Poza chipem ESP-12-E na płytce zestawu rozwojowego znajdują się 2 układy zintegrowany konwerter CP2102 oraz stabilizator LM1117. Odpowiedzialne są one odpowiednio za konwersję USB-UART (umożliwiając bezpośrednie programowanie przez magistralę USB) oraz za regulację napięcia z 5V do 3.3V. Na płytce umieszczono także antenę PCB, dwa przełączniki (RST, FLASH) oraz dwie diody LED.

Podsumowując, wybrana płytka rozwojowa posiada liczne wartości użytkowe, pozwala na bezpośrednie programowanie z komputera PC, a piny GPIO wyprowadzone są z rastrem 2.54mm. Cechy i możliwości zestawu są zaś takie same jak opisywanego chipa. Pozwala to na wygodny i stosunkowo szybki rozwój funkcjonalności produktu, dzięki stabilnej elektrycznie i funkcjonalnej platformie sprzętowej.

Wszystkie wymienione wyżej właściwości oraz wysoka wydajność mikrochipa, stosunkowo niski pobór prądu i niskie koszty produkcji spowodowały, że ESP-12-E znalazł szerokie zastosowanie i stał się bardzo popularnym komponentem w urządzeniach internetu rzeczy IoT.

2.1.3 Interfejsy komunikacyjne w mikrokontrolerach

2.1.3.1 Wejścia-wyjścia ogólnego przeznaczenia – GPIO

Nie tylko z punktu widzenia programisty, podstawowym interfejsem komunikacyjnym pomiędzy układami zintegrowanymi, a urządzeniami peryferyjnymi są wejścia-wyjścia ogólnego przeznaczenia, szeroko znane pod nazwą GPIO[29]. Mogą one być zazwyczaj skonfigurowane w trakcie wykonywania programu jako[30]:

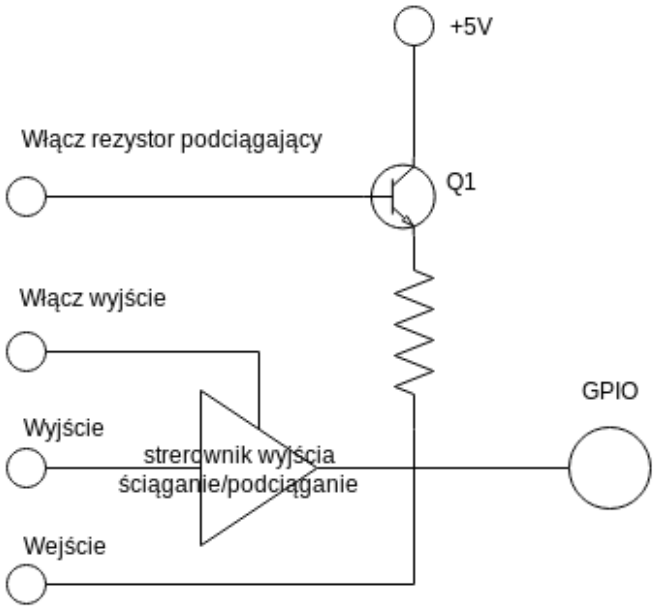
29 GPIO – skrót z ang. general-purpose input/output

30 Monk S., Electronics Cookbook. Practical Electronic Recipes with Arduino and Raspberry Pi, O'Reilly Media, Sebastopol, 2017, s. 145-146

a) cyfrowe wejście do odczytu potencjału elektrycznego i przyporządkowania mu stanu niskiego lub wysokiego

b) cyfrowe wyjście ze stanem niskim (0V) - *(ang. sink current)*

c) cyfrowe wyjście ze stanem wysokim (np. 3.3V lub 5V) - źródło napięciowe



Rysunek 8: Schemat budowy pinu GPIO

Źródło: Opracowanie własne na podstawie Monk S., Electronics Cookbook, s. 25

Linii GPIO w trybie wyjścia można użyć w podstawowy sposób w charakterze binarnym tzn. do włączania / wyłączania podłączonego do niego obciążenia np. diody LED, ustając stan wysoki lub niski determinujący przepływ prądu i w tym wypadku emisję światła. Innym, bardziej zaawansowanym sposobem, jest transmisja danych przez GPIO w postaci sygnału cyfrowego o ustalonym czasie trwania pojedynczego stanu i kodowaniu przesyłanych informacji na kolejnych bitach za pomocą stanu niskiego (logiczne „0") i wysokiego (logiczna „1"). Na rysunku nr 8 przedstawiono schemat budowy GPIO, z opcją użycia wewnętrznego rezystora podciągającego w trybie wyjścia. Niektóre piny mogą jednak mieć prostszą budowę. Brak wbudowanego rezystora, wymusza użycie zewnętrznego opornika w obwodzie

z obciążeniem danego portu. Ogranicza to natężenia prądu elektrycznego pobieranego z mikrokontrolera. Dla wybranego układu scalonego ESP-12-E maksymalne natężenie w trybie źródła to 12mA, natomiast w trybie *sink* to 20mA.

Ze schematu przedstawionym na rysunku 8 można wywnioskować, że deaktywacja sterownika wyjścia umożliwia pinowi GPIO funkcjonowanie jako wejście. Jeśli w tym stanie działanie rezystora podciągającego zostanie programowo włączone, tranzystor Q1 zaczyna przewodzić prąd i ustala wejście GPIO domyślnie w stanie wysokim. Jest to dobra praktyka przy podłączaniu m. in. przełączników do cyfrowego wejścia, gdyż zapobiega oscylacji wartości tego wejścia między stanem niskim a wysokim.

2.1.3.2 Protokół komunikacyjny i magistrala danych I2C

Kolejnym, bardziej wyrafinowanym sposobem na komunikację między mikrokontrolerami i urządzeniami peryferyjnymi jest wykorzystanie dwukierunkowej, szeregowej magistrali danych I^2C [31], opracowanej w 1982r. przez firmę Philips Semiconductor. Składa się ona z dwóch linii i pozwala na wymianę danych pomiędzy dwoma dowolnymi urządzeniami podłączonymi do magistrali. Są to odpowiednio[32]:

a) linia SDA (Serial Data) przenosząca sygnał danych,

b) linia SCL (Serial Clock) przenosząca sygnał zegarowy.

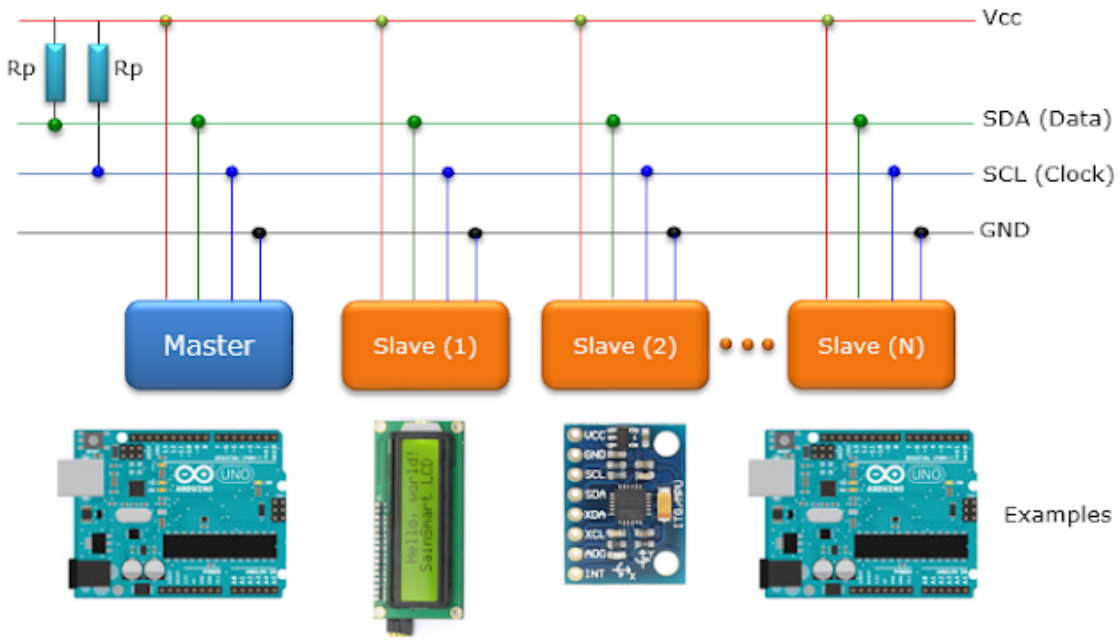
Obie linie są aktywne w stanie niskim, tak więc w prawidłowej aplikacji wymagają zastosowania rezystorów podciągających. Wartość napięcia V_{cc} nie jest odgórnie zdefiniowana, ale zwykle wynosi ona 3.3V lub 5V. Na poniższym rysunku nr 9 przedstawiono schemat budowy magistrali I^2C.

Urządzenia podłączane do magistrali, wyróżniane unikalnymi adresami, podzielić można ze względu na charakter transmisji na nadajniki i odbiorniki, przy czym niektóre urządzenia mogę zarówno nadawać jak i odbierać informacje jak np. dwa komunikujące się mikrokontrolery. Kontrprzykładem jest zaś wyświetlacz LCD podłączony do magistrali IIC, ściśle stanowiący urządzenie odbiorcze.

31 I^2C, I2C lub IIC – skróty z ang. Inter-Integrated Circuit

32 Mielczarek W., Szeregowe interfejsy cyfrowe, Wydawnictwo Helion, Gliwice, 1993r., s. 111-113

Urządzenia podpinane do magistrali podzielić można również ze względu na specjalny status *master* i *slave.* Inicjacja i prowadzenie transmisji charakteryzuje urządzenia o statusie pierwszym statusie. Drugi zaś, otrzymuje każde prawidło podłączone i zaadresowane urządzenie na magistrali.



Rysunek 9: Schemat budowy magistrali I2C z nadajnikami i odbiornikami

Źródło: http://www.mbeddedc.com/2017/05/i2c-bus-communication-protocol-tutorial.html

Początkowo po opracowaniu protokołu, podstawowa prędkość transmisji wynosiła 100kbit/s (tryb *standard*), a na magistrali można było zaadresować 128 urządzeń. W miarę rozwoju technologii prędkość została zwiększona czterokrotnie (w trybie *fast)*, a ilość 8 krotnie - do 1024 urządzeń. Aktualnie dostępne są dostępne jeszcze tryby *fast mode* oraz *high speed* z dostępną przepustowością 1Mbit/s oraz 3.2Mbit/s odpowiednio.

Wybrana w pracy płytka rozwojowa NodeMCU z układem scalonym ESP-12-E dysponuje interfejsem do obsługi magistrali I^2C z maksymalną prędkością 100kbit/s[33]. Potrafi pracować w trybie zarówno *master* jak i *slave*. Funkcjonalność omawianej magistrali jest realizowana przez oprogramowanie układowe, przez co w trakcie trwania programu użytkowego można zdefiniować piny z liniami dla SDA i SCL. Domyślnie są

33 Espressif IOT Team, ESP8266 Technical Reference version 1.2, www.espressif.com, 2016r. s. 1-3

to odpowiednio linie GPIO 4 oraz GPIO 5 mikrokontrolera wyprowadzone na piny D2 i D1 płytki.

2.1.4 Odmierzanie czasu – oscylator kwarcowy

NodeMCU posiada moduł *rtctime* wspomagający podtrzymanie czasu. Jednakże, zgodnie z dokumentacją oprogramowania układowego[34], jego dokładność jest silnie zależna od temperatury. Temperatura zaś zmienia się w zależności stanu aktywności w jakim znajduje się chip ESP-12-E. Powoduje to utrudnienie we właściwej kalibracji zegara. Dodatkowym utrudnieniem w kalibracji są różne częstotliwości pracy układu zintegrowanego: 52MHz podczas procedury startu, 80Mhz w trakcie zwykłej pracy oraz 160MHz w trybie *boosted*. Poza tym utrata zasilania lub niespodziewany reset układu powoduje utratę wartości czasu przechowywanej przez moduł *rtctime*. Układ dysponuje jednak możliwością kompensacji niedokładności poprzez automatyczną, regularną synchronizację za pomocą protokołu SNTP. Dokładność w tym trybie wynosie jednak średnio 5ppm, co w ujęciu miesięcznym przekłada się nawet na 15 sekund odchylenia.

Powyższe niedogodności i utrudnienia, spowodowały potrzebę użycia zewnętrznego zegara czasu rzeczywistego. Spośród dostępnych na rynku wybrano moduł oparty o układ DS3231[35] typu TXCO[36] z obsługą protokołu I^2C. Nieocenionymi jego zaletami są zintegrowany rezonator kwarcowy oraz wbudowany układ pomiaru temperatury. Zapewnienie kompensacji temperaturowej, niwelujące fluktuacje częstotliwości oscylacji, umożliwia utrzymanie precyzyjnego zegara. Jego dokładność w zakresie temperatur 0°C – 40°C jest rzędu 2ppm, co przekłada się na maksymalną odchyłkę w ciągu roku na poziomie około 1 minuty (około 5s na miesiąc).

$$2\,ppm \cdot 1\,rok = 2 \cdot 10^{-6} \cdot 365.25 \cdot 24 \cdot 3600\,s = 63.1152\,s \approx 60\,s$$

Wskazany moduł pozwala na odczyt czasu w postaci godziny, minuty i sekundy oraz daty: miesiąca, dnia i roku i zapewnia bateryjne podtrzymanie tych wartości, dzięki zasilaniu przez akumulator litowy CR2032. Właściwe wyliczenie lat przestępnych zapewnione jest aż do 2100 roku.

34 https://nodemcu.readthedocs.io/en/master/modules/rtctime/

35 Karta katalogowa układu DS3231, Źródło: https://datasheets.maximintegrated.com/en/ds/DS3231.pdf

36 TXCO – skrót z ang. Temperature Compensated Crystal Oscillator

Poza powyżej przytoczonymi możliwościami, moduł z zegarem wysokiej precyzji pozwala wygenerować na jednym ze swoich wyjść falę prostokątną o częstotliwościach 1Hz, 1024Hz, 4096Hz lub 8192Hz. Szczególnie użyteczna w projekcie jest pierwsza wymieniona wartość. Podanie tak ukształtowanego sygnału na wejście GPIO głównego mikrokontrolera i wykrywanie jego zbocza narastającego oraz opadającego, poprzez wykorzystanie przerwań, umożliwi animację upływu czasu.

Dodatkowym atutem jest możliwość przechowania w pamięci modułu DS3231 dwóch alarmów dziennych, dopasowywanych do wybranych podzbiorów wartości: sekunda, minuta, godzina, dzień tygodnia oraz dzień miesiąca. Kończąc podsumowanie, warto dodać, że możliwe jest uzyskanie informacji o przybliżonej temperaturze układu zintegrowanego, z dokładnością na poziomie ±3°C.

Wszystkie wymienione cechy uzasadniają słuszność zastosowania modułu zewnętrznego zegara RTC, gdyż wpływa to na stabilność i dokładność odmierzania upływu czasu – podstawowej funkcji budowanego urządzenia.

2.2 Komponenty i moduły elektroniczne wraz z bibliotekami

Analiza funkcjonalności i cech produktu pozwoliła wytypować niezbędne komponenty elektroniczne do zrealizowania postawionych wymagań. Informacje na ich temat tzn. nazwę, zastosowanie, wymiary, sposób montażu, parametry elektryczne, oznaczenia linii GPIO mikrokontrolera oraz wykorzystaną magistralę danych, zebrano w tabeli nr 4. To syntetyczne zestawienie okazało się niezwykle praktyczne i pomocne w fazie projektowania, a także realizacji, dając na żądanie szybki obraz całości przedsięwzięcia związanego z czujnikami, ekranami i pozostałymi elementami elektronicznymi.

Dodatkowo, uszczegółowione dane na temat każdego z modułów przedstawione zostały w tabelach o nr od 5 do 17. Zawierają one informacje o użytych bibliotekach programistycznych, interfejsach sprzętowych, napięciu zasilania, realizowanych zadaniach w odniesieniu do specyfikacji wymagań oraz poglądowe zdjęcie. Odnośniki do stron z bibliotekami zamieszone zostały na końcu pracy w tabeli nr 30 w spisie bibliotek programistycznych.

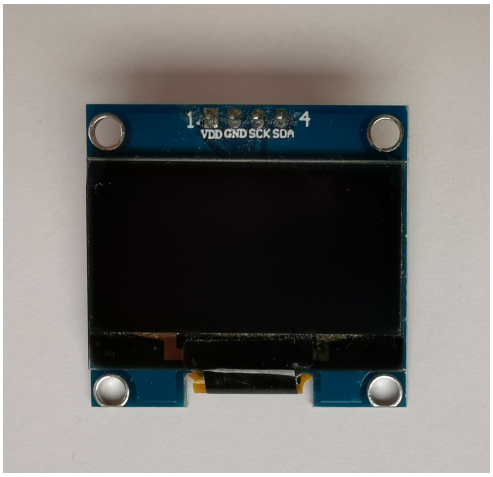
Tabela 4 - Zestawienie peryferyjnych modułów elektronicznych dla NodeMCU

LP.	Nazwa	Opis, przeznaczenie	Wymiary [d x s x w]	Montaż	Zasilanie [V]	Prąd max [mA]	Oznacz. PCB	Magistrala danych	PIN mikrokontrolera	Wtyczka
1	Wyświetlacz OLED niebieski graficzny 1,3" 128x64px I2C	Wyświetlanie informacji dodatkowych	33.5 x 33.5 x 4.3 mm	front	3.3-5V	21 mA	U1_OLED	I2C	D1,D2	JST PH 4pin
2	Wyświetlacz LCD 4x20 znaków	Wyświetlanie wyników pomiarów, equalizera	9.8 x 6.0 x 1.2 cm	front	5V	120 mA	U2_LCD	I2C	D1,D2	JST PH 4pin
3	Moduł z mikrofonem Adafruit MAX9814	Zbieranie dźwięków z otoczenia	26 x 14 mm	prawy bok	2.7-5.5V	6 mA	U3_MIC	analog	n. d.	JST PH 3pin
4	7 kanałowy equalizer MSGEQ7	Graficzny equalizer, pomiar amplitudy 7 częstotliwości spektrum dźwięków z otoczenia	10mm x 10mm	wnętrze	2.7-5.5V	1 mA	U4_MSGEQ7	Cyfr., 2 pin	D6,D7	n. d.
5	Czujnik natężenia światła BH1750	Pomiar natężenia światła celem regulacji jasności wyświetlaczy	13.9 x 18.5mm	góra	3-5V	0,19 mA	U5_BH1750	I2C	D1,D2	JST PH 4pin
6	Pierścień LED RGB 60	Wyświetlanie aktualnego czasu, efektów świetlnych	średnica 170mm	front	5V	1100 mA	U6_LED_RING	Cyfr.,1 pin	D5	JST ST 3pin
7	Czujnik ruchu PIR HC-SR501	Wykrywanie ruchu celem wzbudzenia wyświetlacza	32.5 x 24.5 x 30mm	front	4.5-20V	65 mA	U7_PIR	Cyfr.,1 pin	D4	JST PH 3pin
8	Czujnik temperatury, wilgotności i ciśnienia BMP280	Pomiar temperatury, wilgotności i ciśnienia	15mm x 10 mm	lewy bok	1.8-5V	0,3 mA	U8_BMP280	I2C	D1,D2	JST PH 4pin
9	Moduł wykrywania gestów PAJ7620U2	Zmiana aktualnej funkcji wyświetlacza	20mm x 16 mm	front	3.3-5V	10 mA	U9_PAJ7620	I2C	D1,D2	JST PH 4pin
10	Głośnik z interfejsem cyfrowym	Dźwięki alarmów, dźwiękowa komunikacja błędów	22.7 x 13 x 12mm	wnętrze	3-5V	32 mA	U10_BUZZ	Cyfr.,1 pin	D0	JST PH 3pin
11	Przycisk	Zmiana aktualnej funkcji zegara	12 x 12mm	prawy bok	3-5V	5mA	U11_BUTTON	Cyfr.,1 pin	D8	n. d.
12	Zegar precyzyjny DS3231	Zachowanie czasu po utracie zasilania, generowanie przebiegu prostokątnego f=1Hz	38mm x 22.5mm	wnętrze	3.3-5V	0,65 mA	U12_RTC	I2C	D3	JST PH 5pin
13	Czujnik czystości powietrza CCS811	Wykrywanie związków organicznych, dwutlenek węgla i tlenków metali w powietrzu.	18mm x 20mm	prawy bok	1.8-3.6V	30 mA	U13_CCS811	I2C	D1,D2	JST PH 4pin

2.2.1 Wyświetlacz graficzny OLED 1.3"

Tabela 5 - Podsumowanie informacji o wyświetlaczu OLED SSD1306

Nazwa	Wyświetlacz OLED SSD1306
Biblioteka	U8g2.h
Interfejsy	I2C
Zasilanie	3.3V – 5V
Symbol spec.	[F1], [F8], [C10]
Zadanie	Ciągłe wyświetlanie czasu w formacie HH:MM
Uwagi	Bezpośrednie podłączenie do magistrali +5V, GND, SCL, SDA. 128 x 64px. Kolor znaków niebieski.



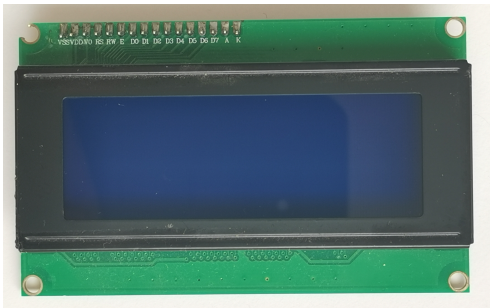
Rysunek 10: Wyświetlacz SSD1306

Źródło: opracowanie własne

2.2.2 Wyświetlacz LCD 4x20 znaków z modułem podświetlenia

Tabela 6 - Podsumowanie informacji o wyświetlaczu LCD 4x20

Nazwa	Wyświetlacz LCD 4x20
Biblioteka	LiquidCrystal_I2C.h
Interfejsy	I^2C
Zasilanie	5V
Symbol spec.	[F3], [F4], [F5], [C10]
Zadanie	Wyświetlanie danych z monitoringu środowiska oraz equalizera
Uwagi	Bezpośrednie podłączenie do magistrali: +5V, GND, SCL, SDA. Posiada dołączony konwerter do I^2C. Kolor podświetlenia niebieski. Przekątna: 2.90"



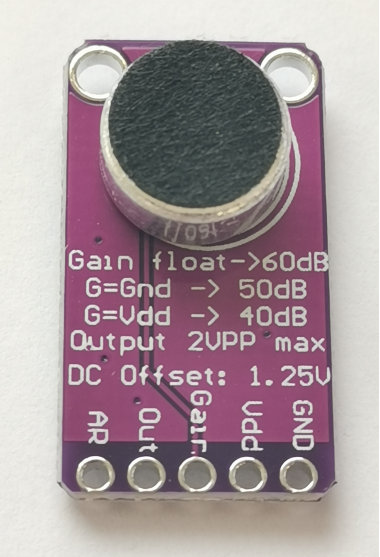
Rysunek 11: Wyświetlacz LCD 4x20

Źródło: opracowanie własne

2.2.3 Mikrofon elektretowy ze wzmacniaczem

Tabela 7 - Podsumowanie informacji o mikrofonie Adafruit MAX9814

Nazwa	Mikrofon Adafruit MAX9814
Biblioteka	nie dotyczy
Interfejsy	linia analogowa (Out)
Zasilanie	2.7 – 5.5V
Symbol spec.	[F5]
Zadanie	Zbieranie dźwięków z otoczenia i zamiana na sygnał elektryczny
Uwagi	Wyjście powinno być podłączone do wejścia equalizera MSGEQ7 i dopasowane w zakresie amplitudy napięcia sygnału. Układ MAX9814 posiada automatyczną regulację wzmocnienia.



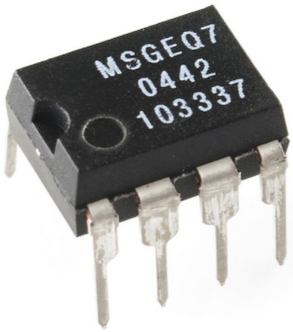
Rysunek 12: Adafruit MAX9814

Źródło: opracowanie własne

2.2.4 7 kanałowy equalizer

Tabela 8 - Podsumowanie informacji o korektorze graficznym MSGEQ7

Nazwa	MSGEQ7
Biblioteka	nie dotyczy
Interfejsy	GPIO, 2 pin [RESET, STROBE]
Zasilanie	2.7V – 5V
Symbol spec.	[F5]
Zadanie	Podział widma sygnału audio na 7 częstotl. 63Hz, 160Hz, 400Hz, 1kHz, 2.5kHz, 6.25kHz i 16 kHz
Uwagi	Wymaga dodatkowych elementów zgodnie ze schematem aplikacji z karty katalogowej. Obudowa DIP8.



Rysunek 13: Equalizer MSGEQ7

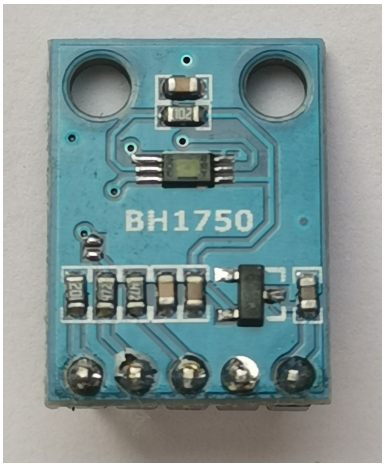
Źródło: www.robotshop.com[37]

37 https://www.robotshop.com/en/audio-frequency-analyzer-chip-msgeq7.html

2.2.5 Moduł czujnika natężenia światła

Tabela 9 - Podsumowanie informacji o czujniku BH1750

Nazwa	Czujnik BH1750
Biblioteka	BH1750.h
Interfejsy	I^2C
Zasilanie	3 – 5V
Symbol spec.	[F3], [C8]
Zadanie	Pomiar natężenia światła celem dostosowania jasności diod na pierścieniu LED RGB
Uwagi	Bezpośrednie podłączenie do magistrali: +5V, GND, SCL, SDA.



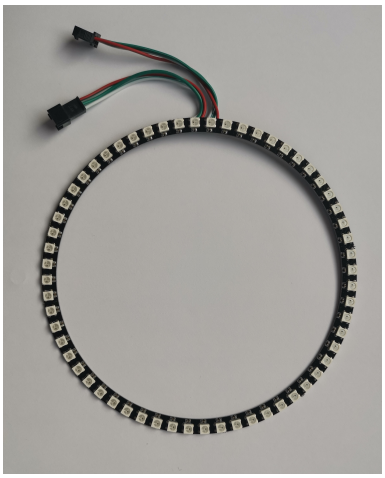
Rysunek 14: Czujnik BH1750

Źródło: opracowanie własne

2.2.6 Pierścień 60 diod LED RGB

Tabela 10 - Podsumowanie informacji o pierścieniu LED RGB (WS2812B)

Nazwa	WS2812B
Biblioteka	FastLED.h, WS2812FX.h
Interfejsy	GPIO, 1pin
Zasilanie	5V
Symbol spec.	[F2], [F5], [F7], [F8], [F9]
Zadanie	Wyświetlanie zegara analogowego oraz efektów świetlnych
Uwagi	Oparty na zintegrowanych źródłach światła WS2812B wzmocnienia.



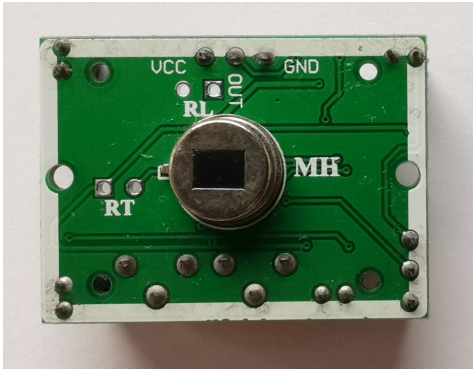
Rysunek 15: Pierścień LED RGB

Źródło: opracowanie własne

2.2.7 Moduł czujnika ruchu PIR

Tabela 11 - Podsumowanie informacji o czujniku PIR HC-SR501

Nazwa	Czujnik PIR HC-SR501
Biblioteka	nie dotyczy
Interfejsy	GPIO, 1pin
Zasilanie	4.5V – 20V
Symbol spec.	[C13]
Zadanie	Wykrywanie ruchu
Uwagi	Zakres pomiarowy: maks. 7 m, kąt widzenia: do 100°, regulacja czasu stanu wysokiego i czułości wykrywania ruchu za pomocą potencjometrów.



Rysunek 16: Czujnik ruchu PIR HC-SR501

Źródło: opracowanie własne

2.2.8 Moduł czujnika temperatury, wilgotności i ciśnienia

Tabela 12 - Podsumowanie informacji o czujniku BMP280

Nazwa	Czujnik BMP280
Biblioteka	BME280I2C.h
Interfejsy	I^2C
Zasilanie	1.8V – 5V
Symbol spec.	[F3]
Zadanie	Pomiar temperatury, wilgotności ciśnienia – monitorowanie środowiska wewnątrz pomieszczenia
Uwagi	Bezpośrednie podłączenie do magistrali: +5V, GND, SCL, SDA.



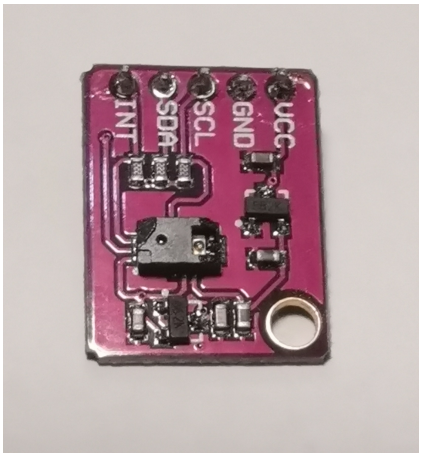
Rysunek 17: Czujnik BMP280

Źródło: opracowanie własne

2.2.9 Moduł wykrywania gestów

Tabela 13 - Podsumowanie informacji o module PAJ7620U2

Nazwa	Moduł PAJ7620U2
Biblioteka	paj7620.h
Interfejsy	I^2C
Zasilanie	3.3V – 5V
Symbol spec.	[C6]
Zadanie	Wykrywanie gestów w celu nawigacji po funkcjach urządzenia m. in. następna/poprzednia
Uwagi	Bezpośrednie podłączenie do magistrali: +5V, GND, SCL, SDA.



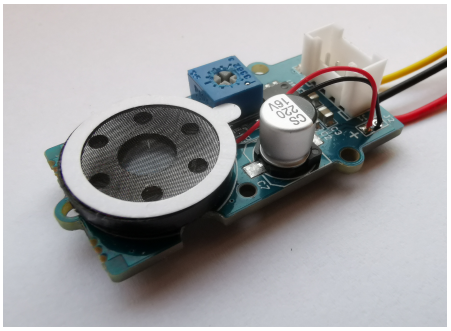
Rysunek 18: Moduł PAJ7620U2

Źródło: opracowanie własne

2.2.10 Głośnik z interfejsem cyfrowym

Tabela 14 - Podsumowanie informacji o głośniku Grove Speaker

Nazwa	głośnik Grove Speaker
Biblioteka	nie dotyczy
Interfejsy	GPIO, 1 pin
Zasilanie	3V – 5V
Symbol spec.	[F6]
Zadanie	Generowanie dźwięków melodii budzika
Uwagi	Tworzenie sygnałów dźwiękowych przy pomocy stałego sygnału napięciowego. Wyzwalany stanem wysokim.



Rysunek 19: Głośnik z interfejsem cyfrowym Grove Speaker

Źródło: opracowanie własne

2.2.11 Moduł przycisku

Tabela 15 - Podsumowanie informacji o module przycisku

Nazwa	Moduł przycisku
Biblioteka	nie dotyczy
Interfejsy	GPIO, 1 pin
Zasilanie	3.3V – 5V
Symbol spec.	[F1-F8], [FK]
Zadanie	Nawigacja, uruchamianie trybu konfiguracji
Uwagi	Nawigacja po funkcjach urządzenia szczególnie w sytuacji, gdy nie działa sterowanie gestami



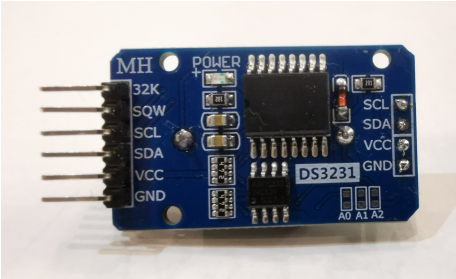
Rysunek 20: Moduł przycisku

Źródło: opracowanie własne

2.2.12 Moduł zegara czasu rzeczywistego RTC

Tabela 16 - Podsumowanie informacji o zegarze czasu rzeczywistego RTC DS3231

Nazwa	Moduł czasu RTC - DS3231
Biblioteka	RtcDS3231.h
Interfejsy	I^2C, 1 pin GPIO (SQW)
Zasilanie	3.3V – 5V
Symbol spec.	[F1], [F2], [F6], [F7]
Zadanie	Odmierzanie czasu, utrzymanie w pamięci czasu po zaniku zasilania
Uwagi	Tworzenie sygnału prostokątnego o częstotliwości f=1Hz



Rysunek 21: Moduł czasu RTC DS3231

Źródło: opracowanie własne

2.2.13 Moduł czujnika czystości powietrza

Tabela 17 - Podsumowanie informacji o czujniku CCS811

Nazwa	Czujnik czystości powietrza CCS811
Biblioteka	SparkFunCCS811.h
Interfejsy	I^2C
Zasilanie	3.3V
Symbol spec.	[F3]
Zadanie	Pomiar całkowitego stężenia lotnych związków organicznych
Uwagi	Czujnik wymaga 20 minutowego czasu ‘rozgrzania’ zanim zacznie dostarczać wiarygodne dane



Rysunek 22: Moduł czujnika czystości powietrza CCS811

Źródło: opracowanie własne

2.3 Technologie informatyczne

2.3.1 Opis zintegrowanego środowiska programistycznego Arduino IDE

Arduino IDE[38] to zintegrowane środowisko programistyczne umożliwiające programowanie mikrokontrolerów z platform sprzętowych typu *open hardware*[39], takich jak Arduino, ESPresso Lite czy NodeMCU, poprzez wykorzystanie portu USB w komputerze. Jest to wieloplatformowy zespół aplikacji napisany w języku Java i dostępny dla systemów operacyjnych Linux, MacOS X oraz Windows[40]. Środowisko poza dostępnym intuicyjnym interfejsem graficznym posiada też możliwość obsługi wyłącznie z linii poleceń. W tej ostatniej odmianie nie ma konieczności instalacji Javy.[41]

38 IDE – skrót z ang. Integrated Development Environment
39 https://www.arduino.cc/en/main/FAQ#toc3
40 https://www.arduino.cc/en/Main/Software
41 https://github.com/arduino/Arduino/blob/master/build/shared/manpage.adoc

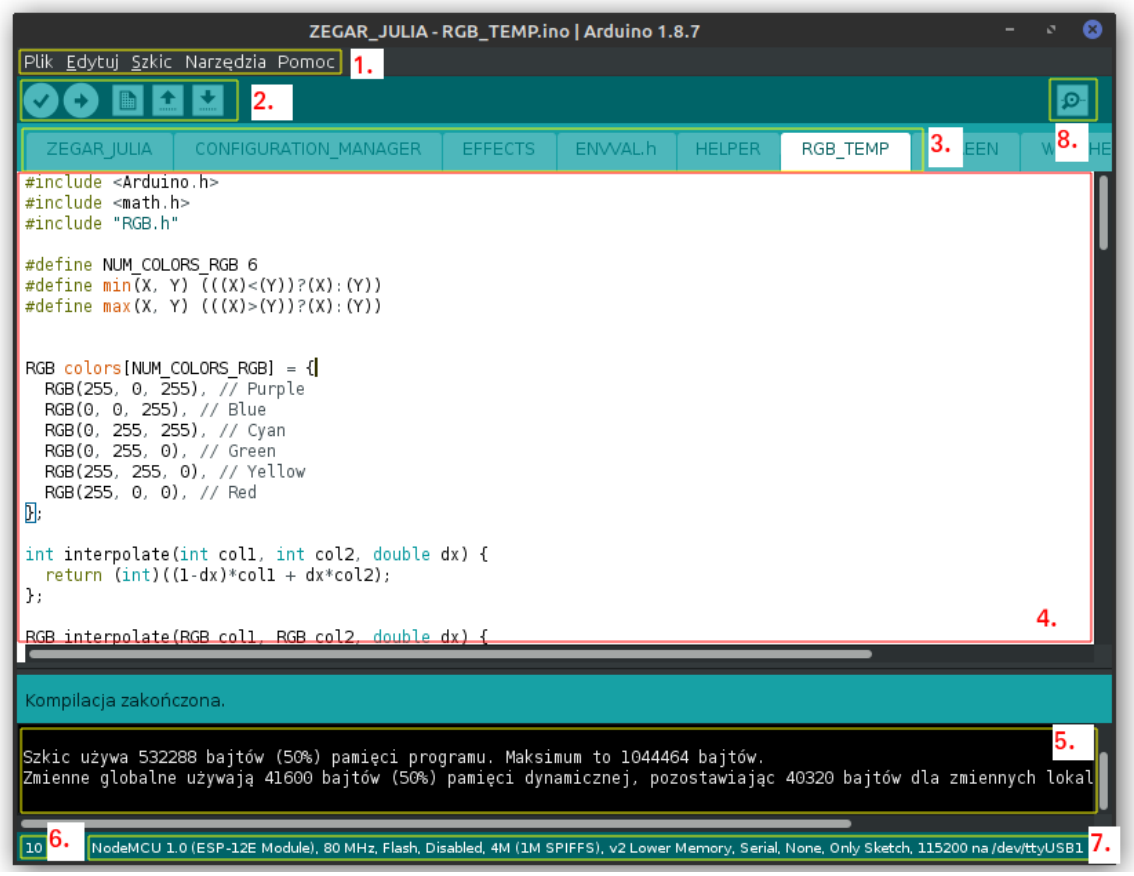
Główne funkcjonalności dostarczane przez Arduino IDE to[42] [43] :

1) edytor kodu źródłowego[44] – wraz z podstawowymi narzędziami ułatwiającymi tworzenie kodu takimi jak kopiowanie/wklejanie, podświetlanie składni, szybkie dodawanie wcięć, szybkie tworzenie komentarzy oraz wyszukiwanie,
2) kompilator – narzędzie do zautomatyzowanej zamiany tworzonego kodu źródłowego na język maszynowy. Wykorzystywany jest kompilator *avr-g++,* co umożliwia używanie w programach standardowych konstrukcji z języka C,
3) *uploader* – umożliwiający przesłanie „produktu" kompilatora do mikrokontrolera i jego właściwe zaprogramowanie, poprzez zapis rozkazów do sekcji aplikacji pamięci FLASH. Do wykonywania tego zadania w środowisku zostało zintegrowane narzędzie linii poleceń *avrdude.*

Opisywana platforma programistyczna wraz ze wspieranymi platformami sprzętowymi jest niezwykle przystępna zarówno dla rozpoczynających programowanie systemów wbudowanych jak i pracę z elektroniką cyfrową. Na rysunku nr 23 zostały oznaczone i ponumerowane (1-8) kluczowe elementy interfejsu Arduino IDE[45]:

1. Menu główne,
2. Szybki dostęp do użytecznych funkcji dotyczących operacji na aktualnym szkicu[46], kolejno od lewej: zweryfikuj, wgraj, nowy, otwórz, zapisz,
3. Karty tworzące aktualny szkic,
4. Edytor kodu programu,
5. Konsola tekstowa z komunikatami kompilatora i aplikacji do wgrywania programów do mikrokontrolerów,
6. Numer wiersza, w którym znajduje się kursor,
7. Informacje o podłączonym układzie z mikrokontrolerem, parametry transmisji oraz nazwa portu szeregowego do komunikacji,
8. Monitor portu szeregowego umożliwiający śledzenie komunikacji z programowanym mikrokontrolerem z podłączonej płytce.

42 https://www.arduino.cc/en/Main/FAQ#toc13
43 Wheat D., Arduino Internals, Apress Media, New York, 2011, s. 32-33
44 https://www.arduino.cc/en/guide/environment#toc3
45 Margolis M., Arduino Cookbook, O'Reilly Media, Sebastopol, 2017, s. 8-11
46 W nomenklaturze Arduino IDE pod pojęciem szkicu należy rozumieć aktualnie tworzony plik z kodem źródłowym programu



Rysunek 23: Interfejs graficzny Arduino IDE

Źródło: Opracowanie własne

Programowanie mikrochipa ESP-12-E nie jest możliwe od razu po instalacji Arduino IDE w wersji 1.8.7[47]. Wstępnie, należy zainstalować niezbędne pliki platformy ESP8266. W menu „Plik→Preferencje→Dodatkowe adresy URL do menedżera płytek”[48] należy podać adres www[49] do pliku z definicjami i danymi konfiguracyjnymi, a następnie zainstalować platformę za pomocą menedżera płytek (dostępnym przez menu „Narzędzia→ Płytki→ Menedżer płytek”).

W celu prawidłowego ustanowienia połączenia z mikrokontrolerem, należy dokonać minimalnej konfiguracji dostępnej w menu „Narzędzia”. Polega ona na wybraniu właściwej płytki rozwojowej, a następnie prędkości przesyłu, częstotliwości jednostki CPU, wielkości pamięci Flash, portu i poziomu debugowania,

47 https://www.arduino.cc/en/Main/Software
48 Schwartz M., ESP8266 Internet of Things Cookbook, Packt Publishing, Birmingham, 2017 s. 2-4
49 http://arduino.esp8266.com/stable/package_esp8266com_index.json

sposobu zapisu funkcji wirtualnych, sygnalizacji wyjątków oraz sposobu zapisu do pamięci Flash.

Podobnie jak większość środowisk deweloperskich, możliwości Arduino IDE mogą zostać rozszerzone za pomocą bibliotek programistycznych[50]. Przykładem może być obsługa układów peryferyjnych podłączanych do płytek rozwojowych jak czujniki, karty pamięci lub ekrany. Znaczna ich ilość jest instalowana wraz z IDE i dostępna w menu „Plik→Przykłady". Środowisko dysponuje jednak rozbudowanym menedżerem bibliotek (menu „Narzędzia→Zarządzaj bibliotekami"), który pozwala wyszukiwać, pobierać i instalować je z zewnętrznego repozytorium. Poza tym, istnieje możliwość importu bibliotek w formacie .zip lub ich bezpośrednia instalacja przez wypakowanie w ustalonym folderze (zdefiniowanym pod opcją menu „Plik→Preferencje→Lokalizacja szkicownika").

2.3.2 Wybrane protokoły komunikacyjne

2.3.2.1 Protokoły synchronizacji czasu NTP i SNTP

Aby zapewnić projektowanemu urządzeniu możliwość wyświetlania aktualnego czasu, w urządzeniu przewidziano użycie zegara czasu rzeczywistego RTC. Jednakże, aby spełnić wymaganie [F2] i zapewnić automatyczne wstępne ustawienie czasu, a następnie jego bezobsługową korektę, wykorzystano sieciowy protokołu synchronizacji czasu – SNTP[51]. Jest to uproszczona wersja pełnej implementacji *Network Time Protocol* (NTP) wykorzystywana do synchronizacji czasu przez urządzenia sieciowe o ograniczonej mocy obliczeniowej i pamięci – takie jak mikrokontrolery – a także w przypadkach, gdy wysoka precyzja czasu nie jest konieczna.

Protokół NTP oparty jest na architekturze klient-serwer[52], w której w odpowiedzi na żądanie czasu przesyłany jest zestaw danych (rys. 24) pozwalający na synchronizację zegara systemowego klienta. Jednakże, czas pomiędzy wygenerowaniem odpowiedzi przez serwer, a odebraniem jej i przetworzeniem przez klienta, jest niewiadomą, którą musi być uwzględniona, jeśli synchronizacja

50 https://www.arduino.cc/en/reference/libraries

51 SNTP – skrót z ang. *Simple Network Time Protocol*

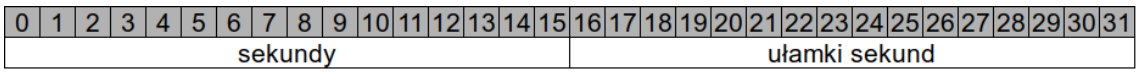
52 Pęczak A., NTP i synchronizacja czasu, IT w Administracji, Październik 2018r.

ma przebiegać z dokładnością do pojedynczych milisekund. Przy użyciu algorytmów statystycznych i wykorzystaniu kilku źródeł czasu oraz jego stopniowej korekcie problem ten rozwiązuje pełna wersja protokołu NTP.

Jego uproszczona wersja – SNTP – pozwala wykorzystać tylko jedno źródło czasu naraz i nie potrafi korygować przekłamań i opóźnień powstających w wyniku transmisji. Korekta czasu jest zazwyczaj aplikowana do zegara klienta skokowo.

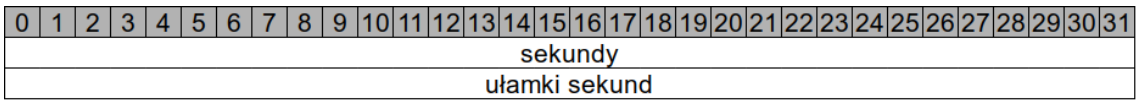
Cechą wspólną obu protokół są te same serwery czasu i pakiety sieciowe o identycznej budowie. Omawiane protokoły nie obsługują stref czasowych posługując się uniwersalnym czasem koordynowanym (UTC)[53]. Za zmianę strefy czasowej odpowiedzialny jest więc system operacyjny klienta.

Budowę dwóch formatów czasu w protokole NTP przedstawiono na rysunkach 24 i 25 [54], gdzie bity ponumerowane są od 0 do 31. Skrócone znaczniki czasu stosuje się do wyrażenia opóźnienia transmisji lub dyspersji zegarów, gdzie maksymalna wartość 2^{16}s (ok.18h) i minimalna 2^{-16}s (ok.15µs) są wystarczająco pojemne i precyzyjne. Natomiast 64-bitowy format, pozwalający wyrazić czas 2^{32}s (136 lat) i 2^{-32}s (232 pikosekundy) w części ułamkowej, jest podstawowym formatem czasu używanym w pakietach NTP. Oznacza on ilość sekund jaka upłynęła od 1 stycznia 1900r. 00:00 czasu UTC.



Rysunek 24: 32 bitowy format czasu NTP (skrócony znacznik czasu)

Źródło: opracowanie własne na podstawie[52]



Rysunek 25: 64 bitowy format czasu NTP (znacznik czasu)

Źródło: opracowanie własne na podstawie[53]

Pakiet NTP, którego strukturę z opisem przedstawiono na poniższym rysunku nr 26, jest datagramem UDP i przesyłany przez dedykowany port 123.

53 https://tools.ietf.org/html/rfc5905, s. 5
54 https://tools.ietf.org/html/rfc5905, s. 13

0 1	2 3 4	5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
LI	**VN**	**Mode**	**Stratrum**	**Poll**	**Precision**
Root Delay [t32] Całkowite opóźnienie wynikające z transmisji (wg zegara serwera – odniesienia)					
Root Dispersion [t32] Maksymalna różnica we wskazaniach zegarów klienta i serwera					
Reference ID (32 bity) 32 bitowy kod identyfikacyjny serwera lub zegara odniesienia					
Reference Timestamp [T64] Czas ostatniej korekty lub ustawienia zegara systemowego klienta					
Origin Timestamp [T64] Czas wysłania żądania do serwera (na zegarze klienta)					
Receive Timestamp [T64] Czas otrzymania żądania przez serwer(wg zegara serwera)					
Transmit Timestamp [T64] Czas wysłania odpowiedzi na żądanie przez serwer(wg zegara serwera)					
Extension Field 1 (zmienna długość)					
Extension Field 2 (zmienna długość)					
Key Identifier (32 bity)					
Dgst [128bitowy MD5 hash]					

Rysunek 26: Format nagłówka pakietu NTP

Legenda: t32 - format skróconego znacznika czasu, T64 – format standardowego znacznika czasu, LI – sekunda przestępna, VN- wersja protokołu, Mode – tryb pracy, Stratum – warstwa wzorca czasu, Pool – max. czas pomiędzy pakietami przesłanymi z sukcesem, Precison – dokładność zegara klienta. Oznaczenia t32 oraz T64 mają znaczenie umowne na potrzebę tego opracowania.

Źródło: opracowanie własne na podstawie[53]

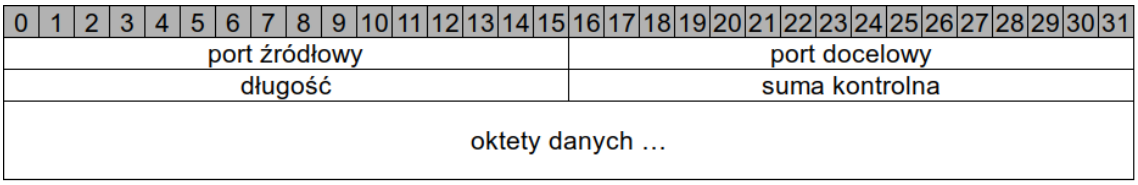
W implementacji projektu za domyślny serwer czasu przyjęto, ze względu na nieodległą lokalizację, maszynę o adresie ntp.task.gda.pl utrzymywaną przez Centrum Informatyczne TASK[55] z Gdańska.

2.3.2.2 Protokół internetowy UDP

W celu transmisji wiadomości przez lokalną sieć komputerową pomiędzy aplikacją mobilną a zegarem LED zdecydowano się wykorzystać protokół UDP (ang. *User Datagram Protocol*). Jest to protokół bezpołączeniowy i zorientowany

55 https://task.gda.pl/uslugi/czas/

transakcyjnie, a samo dostarczenie i wykrywanie zduplikowanych wiadomości nie jest gwarantowane. U podstaw protokołu UDP operującego w warstwie transportowej, leży wykorzystanie protokołu IP[56] z głębszej warstwy – tzw. warstwy sieciowej. Największymi jego zaletami jest szybkość transmisji i brak dodatkowych zadań dla adresata. Efektywność UDP spowodowana jest głównie przez krótki nagłówek oraz brak kontroli przepływu. Na poniższym rysunku przedstawiono budowę nagłówka datagramu UDP.



Rysunek 27: Budowa nagłówka datagramu UDP

Źródło: opracowanie własne na podstawie[55]

Port źródłowy (ang. *source port*) jest polem opcjonalnym i jeśli jest ono uzupełnione to zakłada się, że na niego powinna przyjść odpowiedź. Port docelowy (ang. *destination port*) należy rozumieć w kontekście docelowego adresu IP. Długość (ang. *length*) jest wyrażona w oktetach bitów (bajtach) i obejmuje zarówno nagłówek jak i przesyłane dane. Suma kontrolna (ang. *checksum*) obliczana jest z pseudo nagłówka tworzonego z nagłówka IP, wszystkich pól nagłówka UDP oraz jego danych.

56 https://tools.ietf.org/html/rfc768, s. 1

2.3.2.3 Protokół transmisji danych MQTT

Protokół MQTT (*Message Queue Telemetry Transport*) jest protokołem komunikacyjnym wynalezionym w 1999r. przez dra Andy Stanford-Clark (IBM)[57]. Jego aktualna wersja to 3.1.1 i od 2016r. stał się standardem ISO (ISO/IEC 20922). Charakteryzuje się on niskim obciążeniem przepustowości sieci komputerowej oraz stosunkowo niewielkim zapotrzebowaniem energetycznym, przez co często jest określany jako **lekki** protokół transmisji danych.

Cechy te powodują, że znalazł on szerokie zastosowanie poczynając od domowych projektów automatyzacji, przez dziedzinę ochrony zdrowia (np. do pozyskiwania wyników pacjentów), aż do monitorowania rozległych instalacji kolejowych bądź rurociągowych. Inne, globalne zastosowanie, protokół znalazł przez implementację w komunikatorze Facebook Messenger oraz usłudze Amazon AWS IOT[58]. Podsumowując zastosowania, ze względu na łatwość i ekonomiczność użycia MQTT jest kluczowym protokołem wykorzystywanym w komunikacji *Machine-to-Machine* (M2M) oraz Internecie Rzeczy (IoT)[59].

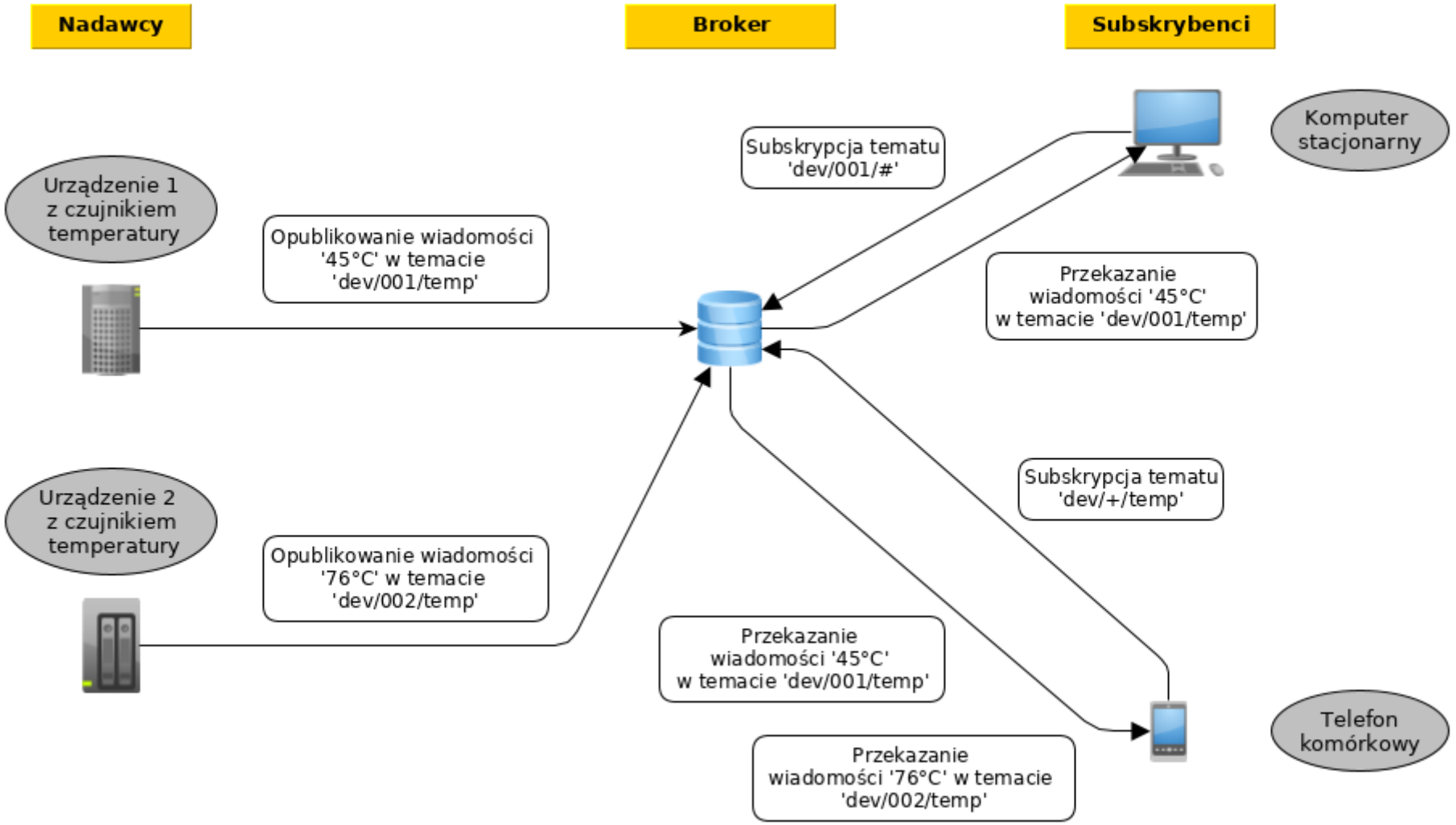
MQTT umożliwia komunikację pomiędzy różnymi systemami wykorzystując serwer pośredniczący, określany mianem **brokera**. Wykorzystując wzorzec *Publish-Subscribe* **odbiorcy** subskrybują „interesujące" ich tematy, na które to publikowane są dane pochodzące od **nadawców**.

W celu właściwego zobrazowania, stworzono czytelny przykład – sieć urządzeń do monitoringu temperatury, obejmujący wymienione 3 role. Został on przedstawiony na rysunku nr 28. Budowa tematów ma charakter hierarchiczny, przez co ułatwione jest nasłuchiwanie na określonych poziomach. Dla zobrazowania tej możliwości, w przykładzie użyto subskrypcji tematów z wykorzystaniem symboli **+** (dopasowanie do dowolnej nazwy na określonym poziomie) oraz **#** (dopasowanie do wszystkich poziomów wgłąb).

57 Pulver T., Hands-On Internet of Things with MQTT, Packt Publishing, Birmingam, 2019, rozdział „Introducing MQTT"

58 https://aws.amazon.com/iot/

59 Hillar G., MQTT Essentials - A Lightweight IoT Protocol, Packt Publishing, Birmingam, 2017, rozdział „Lightweight Messaging with MQTT 3.1.1 and Mosquitto"



Rysunek 28: Role w protokole MQTT oraz wyjaśnienie wzorca *Publish-Subscribe*

Źródło: opracowanie własne

Dane przesyłane przez protokół są z założenia tekstowe, tak więc obok wartości liczbowych lub prostych tekstów, po serializacji danych, do transmisji nadają się także formaty tekstowe takie jak JSON czy XML. Jednakże w tym przypadku deserializacja będzie zachodziła po stronie odbiorcy.

Protokół został wyposażony w trzy poziomy *Quality of Service* (QoS), co pozwala na kontrolę odbioru przesyłanych wiadomości przez subskrybenta:

a) Poziom 0 – wiadomości jest wysyłana i nie jest monitorowana,
b) Poziom 1 – wiadomość musi być dostarczona co najmniej raz (obowiązkowe jest potwierdzenie, może być wielu odbiorców),
c) Poziom 2 – wiadomość musi być dostarczona dokładnie do jednego odbiorcy (obowiązkowe jest potwierdzenie).

W przypadku wysyłki wiadomości z QoS na poziomie 1 lub 2, urządzenia-subskrybenci otrzymają ją przy najbliższym podłączeniu się do brokera, gdyż jednym z jego zadań jest buforowanie wiadomości.

Rolę centralnego huba w zastosowanym modelu *Publish-Subscribe* pełni broker MQTT. Poza już przytoczonymi zadaniami, jest on odpowiedzialny za autentykację i autoryzację klientów (login, hasło), którzy w następstwie mogą się stać nadawcami i/lub odbiorcami[60]. Dostępne są implementację serwerów na najpopularniejsze systemy operacyjne włączając Windows, MacOS oraz Linux. Domyślny port na których operuje protokół to 1883. W razie potrzeby, istnieje możliwość wykorzystania protokołu TLS poniżej MQTT, co określane jest mianem *Secure MQTT*. W tej sytuacji domyślnym portem serwera pośredniczącego jest port 8883.

2.4 Struktury danych – format wymiany danych JSON

JSON (z ang. *JavaScript Object Notation*) to tekstowy format wymiany danych pomiędzy różnymi systemami[61]. Mimo posiadania w nazwie skryptowego języka programowania, format ten nie jest zależny od żadnego z języków i jego notacja opiera się na reprezentacji uniwersalnych struktur danych. Są to zbiory par nazwa/wartość

60 Ibidem
61 Bassett L., Introduction to JavaScript Object Notation, O'Reilly Media, Sebastopol, 2015, s. 1-4

oraz uporządkowana lista wartości[62] – obecne w większości języków programowania (z tą różnicą, że są w nich inaczej zaimplementowane). W najnowszej specyfikacji JSON, zawartej w RFC 8259[63], do dwóch podstawowych wyżej wymienionych wartości JSON dołącza się jeszcze numer (ang. *number*), łańcuch tekstowy (ang. *string*) oraz literały *true*, *false* i *null.*

Zbiory par nazwa/wartość przechowywane są w sposób nieuporządkowany w obiektach, których opis rozpoczyna się od lewej klamry **{** a kończy prawą klamrą **}**. Pomiędzy nazwą a wartością występuje znak rozdzielający w postaci dwukropka **:**, poszczególne pary nazwa/wartość są oddzielane zaś za pomocą przecinka. Poniżej przedstawiono przykładowy obiekt JSON.

```
{

 "nazwa1": "wartość1",
 "nazwa2": false,
 "nazwa3": 256
}
```

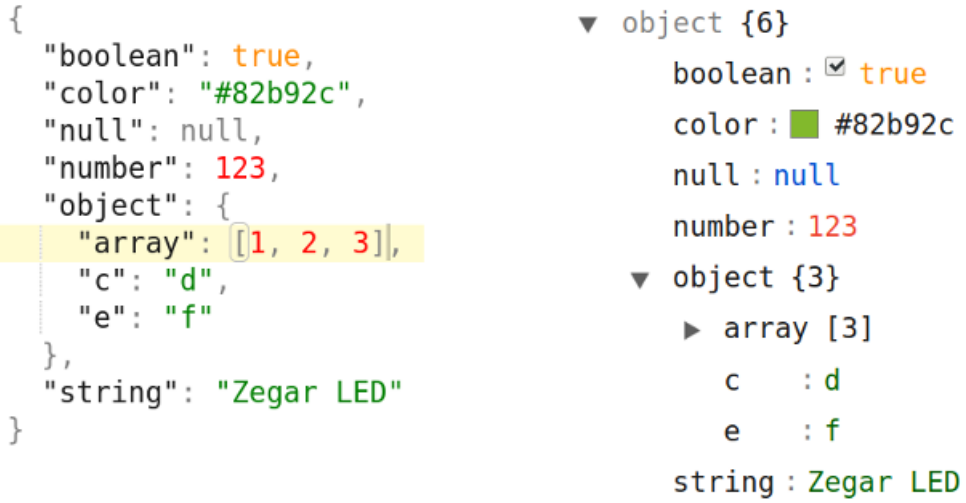
Tabele, służą w notacji używanej w JSON do przedstawienia uporządkowanego zbioru wartości. Ich opis zaczyna się od lewego nawiasy kwadratowego **[** a kończy prawym nawiasem kwadratowym **]**. Do rozdzielania wartości używa się separatora w postaci przecinka. Przykładową tabelę stosowaną w opisywanym formacie wymiany danych przedstawiono poniżej.

```
[ "napis1", true, 1024, "napis2", false, null ]
```

Mimo prostej gramatyki, za pomocą tego formatu mogą być zapisywane złożone struktury danych, gdyż jest możliwe stosowani zagnieżdżeń wartości. W tej sytuacji bardzo pomocne są walidatory jak np. *JSON Editor Online*[64]. Przykład z rysunku nr 29 pokazuje tekst w formacie JSON oraz jego obiektową reprezentację.

W implementacji projektu „Zegar LED” format JSON został użyty do przechowywania danych konfiguracyjnych w pliku zapisanym w pamięci flash układu NodeMCU. Drugim, bardziej zasobochłonnym dla aplikacji obszarem, gdzie wykorzystano ten format, było przetwarzanie danych pozyskiwanych z API serwisów pogodowych, co zostanie szerzej omówione w podrozdziale 3.2.5.

62 Smith B., Beginning JSON, Apress Media, New York, 2015, s. 56-57
63 https://tools.ietf.org/html/rfc8259
64 http://jsoneditoronline.org/



Rysunek 29: Zagnieżdżony tekst w formacie JSON wraz z obiektową reprezentacją

Źródło: opracowanie własne przy wykorzystaniu[63]

Rozważano również użycie notacji JSON do wykorzystania w komunikatach wymienianych pomiędzy aplikacją mobilną a zegarem. Ostatecznie, ze względu na nieskomplikowaną budowę komunikatów oraz potrzebę oszczędzania zasobów pamięciowych mikrokontrolera, komunikaty nie zostały w ten sposób zaimplementowane.

Niepodważalną zaletą opisywanego formatu jest jego prostota i lekkość, rozumiana jak brak dużego narzutu danych. Właśnie dzięki tej prostocie, nie są spodziewane zmiany w gramatyce JSON, co stanowi o jego ogromnej stabilności[65] Przytoczone powyżej cechy czynią go podstawową notacją wymiany danych pomiędzy systemami informatycznymi.

65 ECMA Technical Committee 39, The JSON Data Interchange Syntax, Standatd ECMA-404, Ecma International 2017, s. iii

3. Projekt i realizacja

3.1 Warstwa sprzętowa

3.1.1 Schemat elektryczny urządzenia

W kolejnym etapie pracy, korzystając z modułów i komponentów elektronicznych wyspecyfikowanych w rozdziale 2, rozpoczęto tworzenie schematu elektrycznego urządzenia, a następnie na jego podstawie projektowanie płytki PCB.

Analiza danych zebranych w tabeli nr 4, wykazała, że poza układem CCS811, wszystkie moduły elektroniczne tolerowały zasilanie na poziomie 5V, tzn. dokładnie takie jak główny moduł NodeMCU. Uwzględniono to prowadząc linie zasilające bezpośrednio z gniazda zasilającego (+5V, GND). Wymieniony wyjątek, wymagający napięcia zasilającego 3.3V, obsłużono poprzez zasilenie go z konwertera napięć LM1117 (wbudowanego w platformę sprzętową mikrochipa). Żądane napięcie jest dostępne na pinach płytki oznaczonych napisem „3V", zdolnych do wymaganego dodatkowego średniego wydatku prądowego w postaci 12mA.

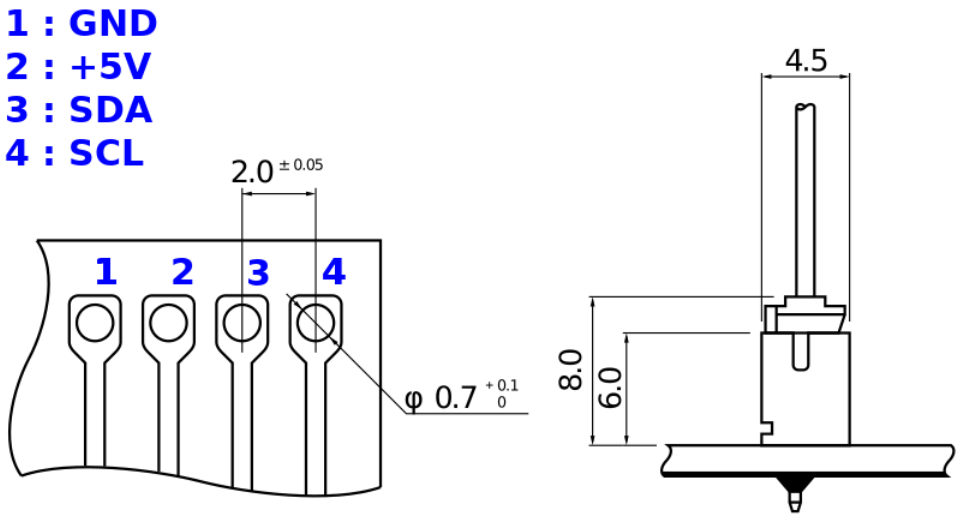
Wykluczając pierścień LED, wszystkie połączenia modułów elektronicznych zaprojektowano używając złącz JST PH (3-5pin), zdolnych do przenoszenia prądów o natężeniach aż do 2A i napięć do 100V[66]. Takie rozłączalne rozwiązanie autor motywuje przede wszystkim ułatwieniem diagnostyki i napraw w przyszłości w trakcie eksploatacji urządzenia. Zasilanie pierścienia zostało przewidziane poprzez połączenie trwałe przez pady lutownicze w płytce, wyprowadzone w odległości poniżej 1cm od gniazda zasilającego, celem ograniczenia strat energii i ciepła wydzielanego w części obwodu, przez którą mają płynąć prądy stałe o natężeniu rzędu 1 ampera.

4-pinowe złącza JST okazały się szczególnie użyteczne przy fizycznej realizacji magistrali I2C. Przyjętą logikę oznaczeń poszczególnych sygnałów w złączu ukazano na rysunku nr 30, którego podkład zawiera przekroje tego złącza.

Analogicznie do powyższego, w złączach 3-pinowych do pinu nr 1 i 2 przyporządkowano odpowiednio linie GND oraz 5V. Ostatni trzeci pin jest zaś częścią linii z właściwym sygnałem. Jedyne złącze 5-pinowe zastosowane na płytce służy do komunikacji z modułem zegara RTC. Ma ono przypisanie dokładnie takie jak wspomniane JST 4-pin, natomiast ostatni piąty pin przeznaczony jest do przenoszenia

66 Katalog JST Catalog vol. 120e, str. 21

sygnału prostokątnego o częstotliwości 1Hz, generowanego na wyjściu SQW modułu zegara czasu rzeczywistego.



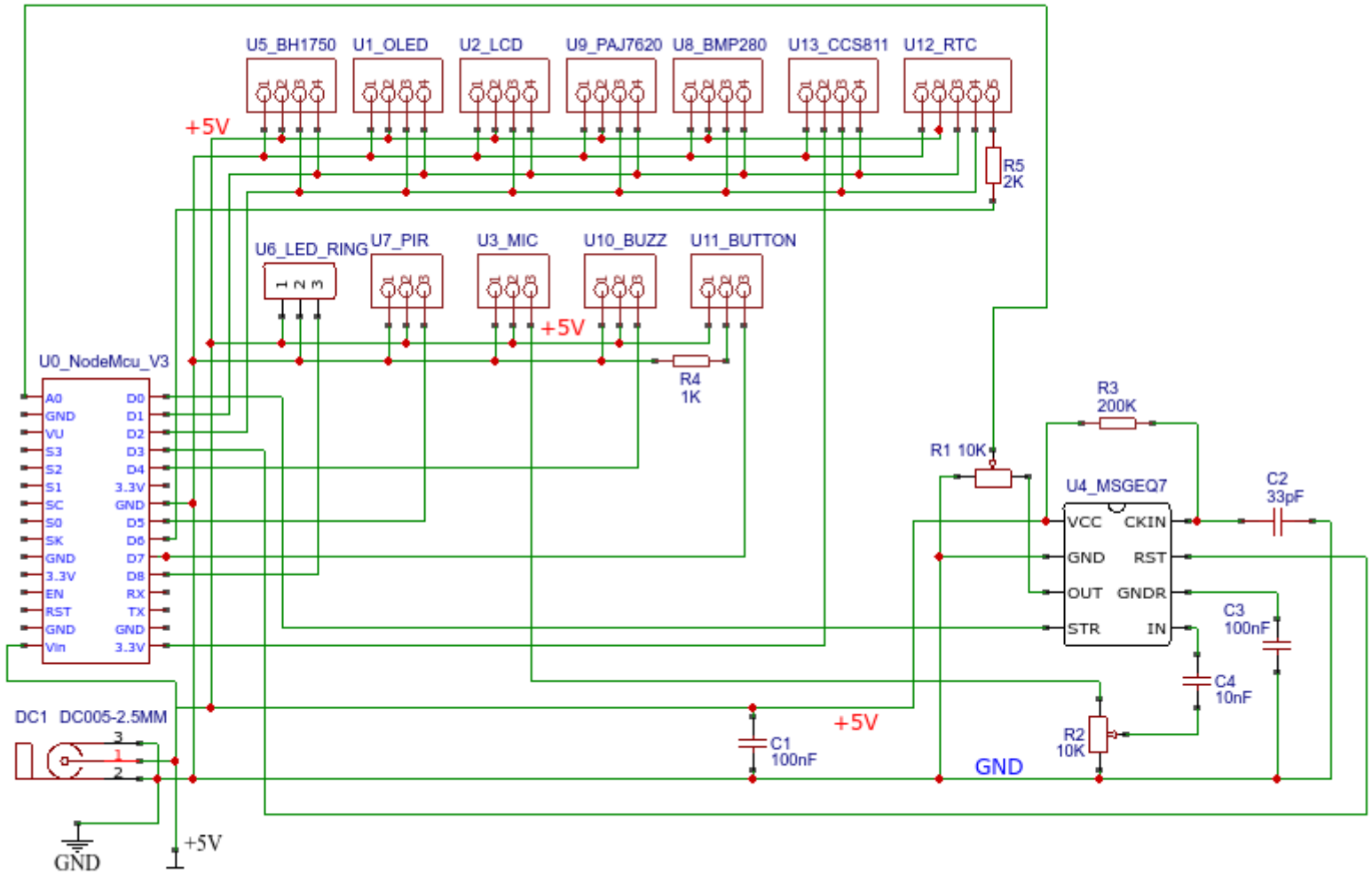
Rysunek 30: Sygnały w złączu JST

Źródło: Opracowanie własne na podstawie katalogu JST Catalog vol. 120e str. 21

Szczególną uwagę zwrócono na tor przetwarzania dźwięku, który został wykonany zgodnie z przykładową aplikacją z karty katalogowej modułu equalizera MSGEQ7[67]. Na wejście tego układu poprzez potencjometr *R2* podawany jest sygnał zmienny, przenoszący charakterystyki dźwięków zbieranych z otoczenia przez moduł mikrofonu i wzmacniacza MAX9814. Na wyjściu equalizera okresowo pojawia się napięcie proporcjonalne do amplitudy sygnału wejściowego dla poszczególnych analizowanych częstotliwości. Poprzez potencjometr *R1* sygnał ten podawany jest na jedyne wejście układu NodeMCU z konwerterem analogowo-cyfrowym. Oba potencjometry mają za zadanie regulację napięcia, przez co umożliwiają wstępne ustawienie parametrów pracy i czułości, rozumianej jako wstępna konfiguracja w danym pomieszczeniu.

Ostatecznie, schemat elektryczny został przedstawiony na rysunku nr 31. Jego istotne uzupełnienie stanowi tabela nr 18, bardziej szczegółowo specyfikująca poszczególne elementy.

67 Seven Band Graphic Equalizer Data Sheet, 9/2004, http://mix-sig.com



Rysunek 31: Schemat elektryczny płytki głównej zegara

Źródło: opracowanie własne

Tabela 18 - Zestawienie elementów do montażu na płytce głównej zegara

LP.	Oznaczenie	Opis
1	R1	Potencjometr montażowy leżący 10kΩ (0.1W)
2	R2	Potencjometr montażowy leżący 10kΩ (0.1W)
3	R3	Rezystor przewlekany 200kΩ (0.25W)
4	R4	Rezystor przewlekany 1kΩ (0.25W)
5	R5	Rezystor przewlekany 2 kΩ (0.25W)
6	C1	Kondensator ceramiczny 100nF
7	C2	Kondensator ceramiczny 33pF
8	C3	Kondensator ceramiczny 100nF
9	C4	Kondensator ceramiczny 10nF
10	DC1	Gniazdo DC 5.5 x 2.5mm
11	U0_NodeMcu_V3	30 x pad lutowniczy
12	U1_OLED	Złącze JST PH, 4 pin, proste
13	U2_LCD	Złącze JST PH, 4 pin, proste
14	U3_MIC	Złącze JST PH, 3 pin, proste
15	U4_MSGEQ7	Podstawka do układów DIP 8 pin
16	U5_BH1750	Złącze JST PH, 4 pin, proste
17	U6_LED_RING	3 x pad lutowniczy
18	U7_PIR	Złącze JST PH, 3 pin, proste
19	U8_BMP280	Złącze JST PH, 4 pin, proste
20	U9_PAJ7620	Złącze JST PH, 4 pin, proste
21	U10_BUZZ	Złącze JST PH, 3 pin, proste
22	U11_BUTTON	Złącze JST PH, 3 pin, proste
23	U12_RTC	Złącze JST PH, 5 pin, proste
24	U13_CCS811	Złącze JST PH, 4 pin, proste

Wykorzystując ponownie dane zgromadzone w tabeli nr 4, obliczono, iż maksymalny prąd pobierany przez projektowanie urządzenie może osiągnąć wartość około 1400mA (z czego w 1100mA, ≈ 79% udziału, generuje pierścień diod RGB w pełni rozświetlony kolorem białym).

Uwzględniając przeznaczenie zasilacza do pracy ciągłej i przyjmując najwyższy dopuszczalny pobór za 80% prądu maksymalnego, ustalono wydajność prądową zasilacza na 2A. Źródło napięciowe, ze względu na platformę sprzętową NodeMCU oraz podłączone do niej moduły, powinno dostarczać napięcie o wartości nominalnej 5V. Ostatecznie, wszystkie parametry zasilacza przedstawiono w poniższej tabeli nr 19.

Tabela 19 - Parametry zasilacza zegara

Rodzaj zasilacza:	impulsowy
Napięcie zasilania:	100 - 240 V AC
Zakres częstotliwości:	50 / 60 Hz
Napięcie wyjściowe:	5 V DC
Wydajność prądowa zasilacza:	2 A
Moc zasilacza:	10 W
Wyjście:	Wtyk 2.1 / 5.5 mm na przewodzie

3.1.2 Fazy powstawania, projekt i wykonanie płytki PCB

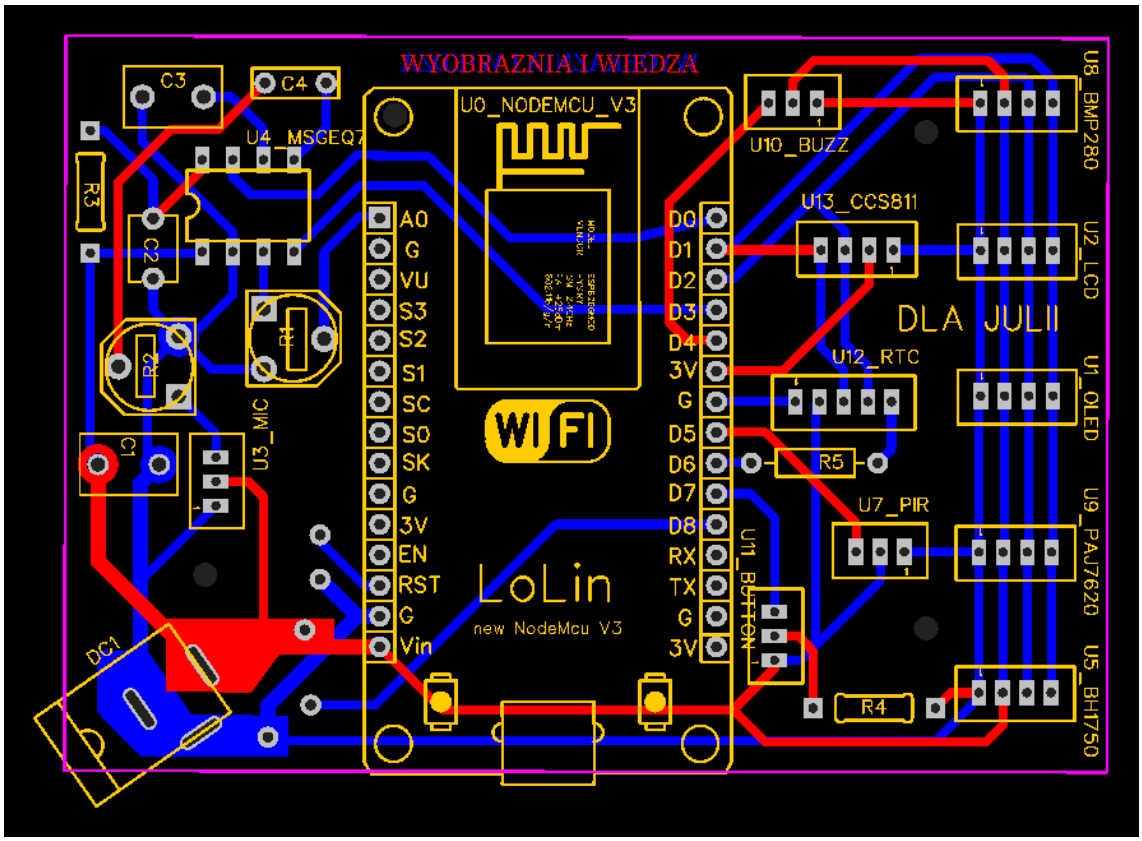
Pierwszą fizyczną realizację schematu z rysunku nr 31 wykonano na dwóch płytkach stykowych o rastrze 2.54mm (0.1cal). Wszystkie wymienione w tabeli nr 4 moduły (poza equalizerem MSGEQ7 w obudowie DIP8) posiadały dokładnie taki sam rozstaw wyprowadzeń (pinów). Wytworzony w ten sposób pierwszy prototyp urządzenia, przez zastosowanie przewodów połączeniowych, mógł być szybko modyfikowany, a także poddawany diagnostyce. Na tym etapie, przy wykorzystaniu bibliotek programistycznych do obsługi wyspecyfikowanych układów peryferyjnych, stworzono testowe oprogramowanie, dzięki któremu sprawdzono działanie modułów i potwierdzono prawidłowość założeń projektowych. Wyniki działania wyświetlane były w konsoli Arduino IDE na ekranie komputera. Problemy, jakie pojawiły się w tej fazie spowodowane były głównie nietrwałością połączeń lub ich zwiększoną rezystancją. Dla przykładu, podczas testów diod LED przy wyświetlaniu sekwencji kolorów, nie udawało się uzyskać koloru białego na całym okręgu, co spowodowane było stratami energii na linii zasilającej. Dlatego też, po ustaleniu i usunięciu przyczyn problemów, a następnie potwierdzeniu działania każdego z modułów, przystąpiono do zaprojektowania pierwszej wersji płytki PCB.

Do tego zadania wykorzystano narzędzie *Online Editor* dostępne w serwisie https://easyeda.com. Płytka miała kształt prostokąta o wymiarach 116mm x 78mm. Zlecenie wykonana płytki powierzono firmie Margol Electronics Sp.z o.o. z Blachowni. Za podłoże wybrano laminat epoksydowy FR-4 o grubości 1.2mm. Z racji, iż nie była to finalna wersja obwodu drukowanego, celem oszczędności, nie zamówiono dodatkowo płatnych opcji takich jak soldermaska, metalizacja otworów, cynowanie czy napisy. Płytka po odebraniu została sprawdzona pod kątem ciągłości ścieżek i zgodności z projektem. Następnie dokonano montażu elelementów elektronicznych przy użyciu 75W stacji lutowniczej, po czym dokonano ponownej weryfikacji połączeń. Pierwsze uruchomieniu tej wersji sprzętowej urządzenia wraz z podłączonymi wszystkimi modułami pod kontrolą oprogramowania testowego zakończyło się skucesem. Powstały układ pracował stabilnie i posłużył do rozwoju właściwego oprogramowania zegara, co z perspektywy czasu autor ocenia za prawidłowe postępowanie. Zagadnienia i pojawiające się problemy dotyczyły tworzonego kodu i były głównie natury implementacyjnej, a nie sprzętowej, co drastycznie skracało czas ich zdiagnozowania i rozwiązania. W dobrze przygotowanym środowisku prototypowym programowanie i realizacja wymagań projektu przebiegała w harmonii.

Wart krótkiego omówienia, jest fakt, iż zaprojektowano dwa pady lutownicze przedłużające wyjścia RST i GND platformy NodeMCU (rysunek 32). Autor, poza specyfikacją, postanowił na tym etapie zrobić te wyprowadzenia w celu podłączenia niewielkiego monostabilnego przycisku, dostępnego na obudowie. Krótkotrwałe zwarcie linii RST i GND powoduje reset sprzętowy, co może być praktyczną funkcjonalnością w trakcie użytkowania zegara.

Ostateczna wersja płytki, oparta na schemacie z rysunku nr 31, powstała przez nieznaczą modyfikację poprzedniej wersji. Dokonano optymalizacji rozplanowania elementów, przez co płytka PCB uzyskała rozmiar 87mm x 61mm, a sumaryczna długość ścieżek zmniejszyła się o około 29%. PCB posiadała dwie warstwy sygnałowe, umieszczono na niej 107 padów lutowniczych, 0 przelotek *via* (4 przed opytmalizacją) oraz 4 otwory montażowe. Widok płytki przedstawiono na rysunku nr 32. Fioletowa linia wyznacza prostokątny obrys płytki. Czerwonym kolorem zaznaczono ścieżki warstwy górnej, niebieskim zaś warstwy dolnej. Żółtym kolorem nakreślono obrysy komponentów elektronicznych oraz ich **oznaczeń**, które są tożsame z nazwami użytymi

w tabelach nr 4 i 18. Konsekwentnie podczas projektowania, implementacji oraz tworzenia niniejszej pracy odnoszono się do modułów za pomocą symbolicznych nazw o budowie: **U<liczba>_nazwa** co oszczędzało czas poprzez przyspieszona identyfikację. Za podłoże ponownie wybrano laminat epoksydowy FR-4 o grubości 1.2mm. Tym razem zdecydowano się na pokrycie obwodu drukowanego warstwą ochronną w postaci czarnej soldermaski. Otwory zostały poddane galwanicznej metalizacji, a do padów zamówiono opcję chromianowania w celu poprawy lutowności.



Rysunek 32: Projekt płytki głównej zegara – wersja finalna

Źródło: opracowanie własne

Napis „Wyobraźnia i wiedza”, na środku w górnej części powyższego rysunku (wykonany na obu stronach płytki), nie był wstępnie planowany. Jednakże, firma wytwarzająca obwód drukowany ze względów technologicznych, wymagała dowolnego ciągu znaków w warstwie ścieżek - w taki sposób powstał i zrealizował się pomysł na odautorski przekaz.

3.1.3 Projekt i wykonanie obudowy

Wykorzystując wolne oprogramowanie do komputerowego wspomagania projektowania **FreeCAD** oraz podpierając się jego oficjalną instrukcją użytkownika[68], a także książką z praktycznymi przykładami[69], rozpoczęto proces projektowania obudowy. Składała się ona z trzech części:

1. przedniej – pełnego, przezroczystego, walca o grubości 2mm z fazą 2mm, materiał: bezbarwny pleksiglas (rysunek nr 34),
2. głównej – wydrążonego walca o grubości 36mm, do osłony układów elektronicznych, materiał: czarny ABS (rysunki nr 35, 36, 37 i 38),
3. tylnej – pełnego walca o grubości 2mm z otworami wentylacyjnymi, przykręcanego do głównej obudowy i stanowiącego jej zamknięcie, materiał: czarny ABS (rys. nr 39).

We frontowej części głównej części obudowy zaplanowano dwa prostokątne otwory na ekrany LCD i graficzny [C10]. Poza nimi, w celu prawidłowego działania czujników ruchu i gestów, przewidziano dwa okrągłe otwory, każdy o średnicy 9mm. Spowodowane było to faktem, iż nawet przezroczysty materiał o grubości 2mm skutecznie tłumił promieniowanie termiczne człowieka.

W przekroju poprzecznym, zewnętrza część obudowy ma kształt okręgu o średnicy 190mm, wewnętrzna zaś 8-kąta foremnego o najdłuższej przekątnej 180mm. W ścianach bryły opartej na tej figurze zapewniono 8 otworów – 4 wentylacyjne, pozostałe 4 zaś zostały wykorzystane do zapewnienia dostępu do otoczenia dla czujników: czystości powietrza, temperatury/ciśnienia/wilgotności, natężenia oświetlenia oraz mikrofonu.

Płytce PCB z elektroniką zegara, wyniesionej 7mm ponad powierzchnię wewnętrzną obudowy, zapewniono 3 punktowe podparcie i mocowanie śrubami M2 o długości 8mm. Każdy z czujników został rozmieszczony w dedykowanej kieszeni z dopasowanymi otworami montażowymi. Mocowanie dla każdego z nich zaplanowano za pomocą śrub M2 i M3 o różniej długości, w zależności od dostępnej przestrzeni w głąb bryły.

68 Yorik van Havre i in., *FreeCAD Manual*

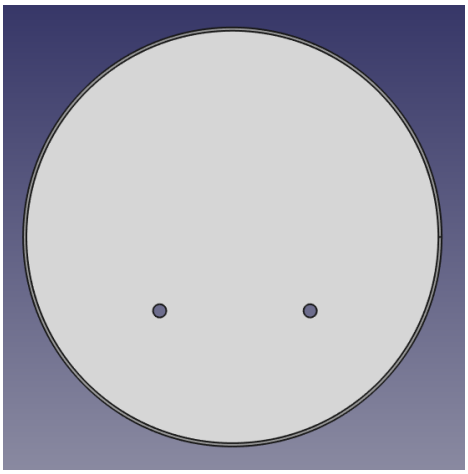
69 Falck D., Collette B., *FreeCAD [How-to]*, Packt Publishing, Birmingam, 2012

Nazwa:	**Zegar LED**
Numer seryjny:	0001
Rok produkcji:	2020
Wersja sprzętu:	1.0
Wersja oprogr.:	1.0.0
Zasilanie:	**5V DC, 2A**

Rysunek 33: Projekt tabliczki znamionowej urządzenia

Aby spełnić postawione wymagania dotyczące trwałości i estetyki, przednia część została wykonana z bezbarwnego plexiglasu, natomiast główna i tylna z czarnego tworzywa ABS, stosowanego często jako materiał na obudowy urządzeń elektronicznych, galanterii ozdobnej, zabawek oraz wielu innych[70]. Realizację projektu zlecono firmie T3D z Białegostoku świadczącej usługi druku 3D między innymi za pomocą czarnego filamentu ABS. Zamówiona wysokość warstw pionowych to 190µm, dokładność wymiarów liniowych to ±0.1mm, natomiast stopień wypełnienia drukowanej bryły został ustalony na 50%. Wydruk został wykonany w technologii *Fused Deposition Modeling* (FDM) polegającej na warstwowym osadzaniu termoplastycznego tworzywa sztucznego[71].

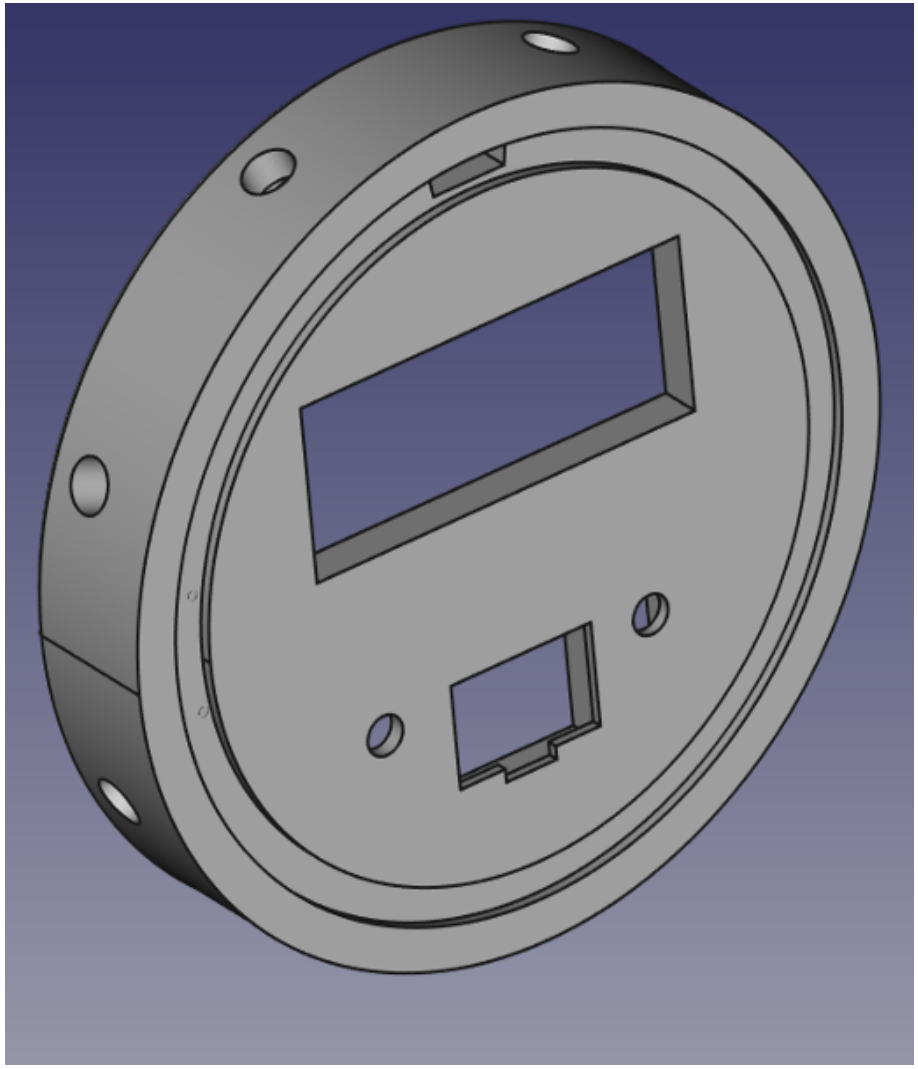
Wygląd modelu obudowy z kilku różnych perspektyw przedstawiono na poniższych rysunkach od nr 34 do 40. Poza tym zaprojektowano tabliczkę znamionową, z zamiarem wydrukowania jej na naklejce przytwierdzonej na tylnej części obudowy (rys. nr 33).



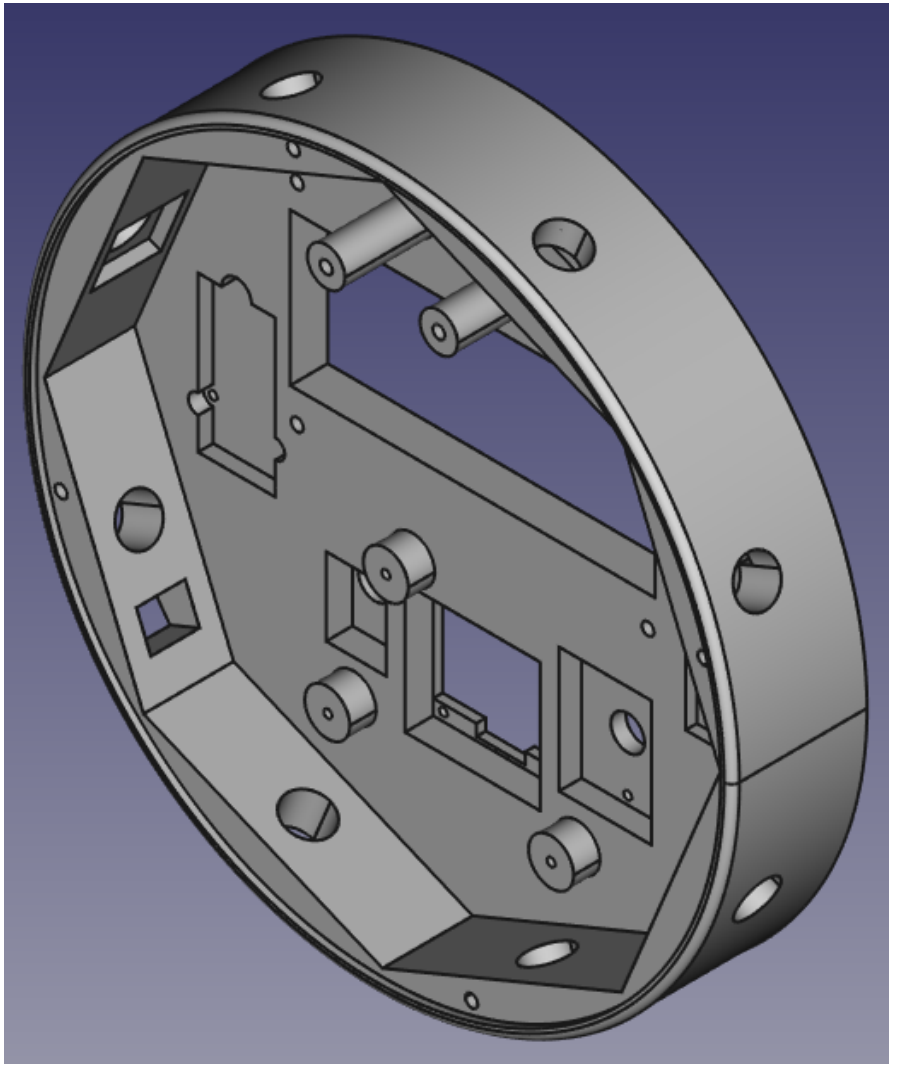
Rysunek 34: Model frontowej części obudowy

70 https://pl.wikipedia.org/wiki/ABS_(tworzywo)

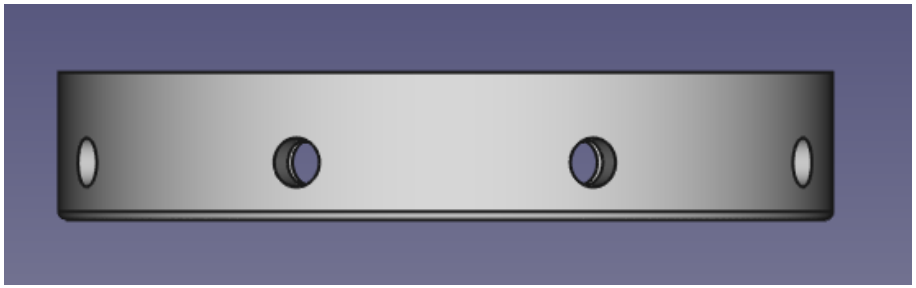
71 Mazurkiewicz A., Analiza jakości wydruku elementu z termoplastu ABS wykonanego w technologii FDM, str. 956-960



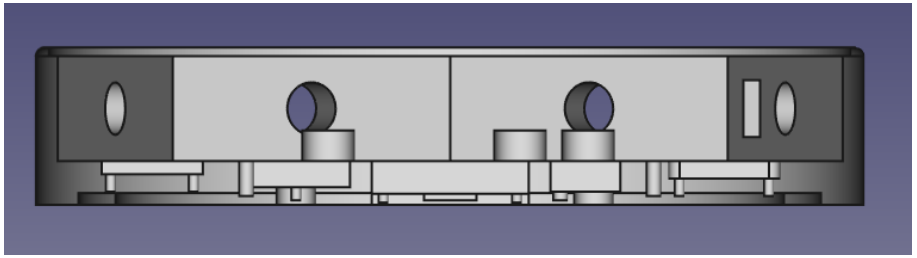
Rysunek 35: Widok izometryczny modelu zegara - front



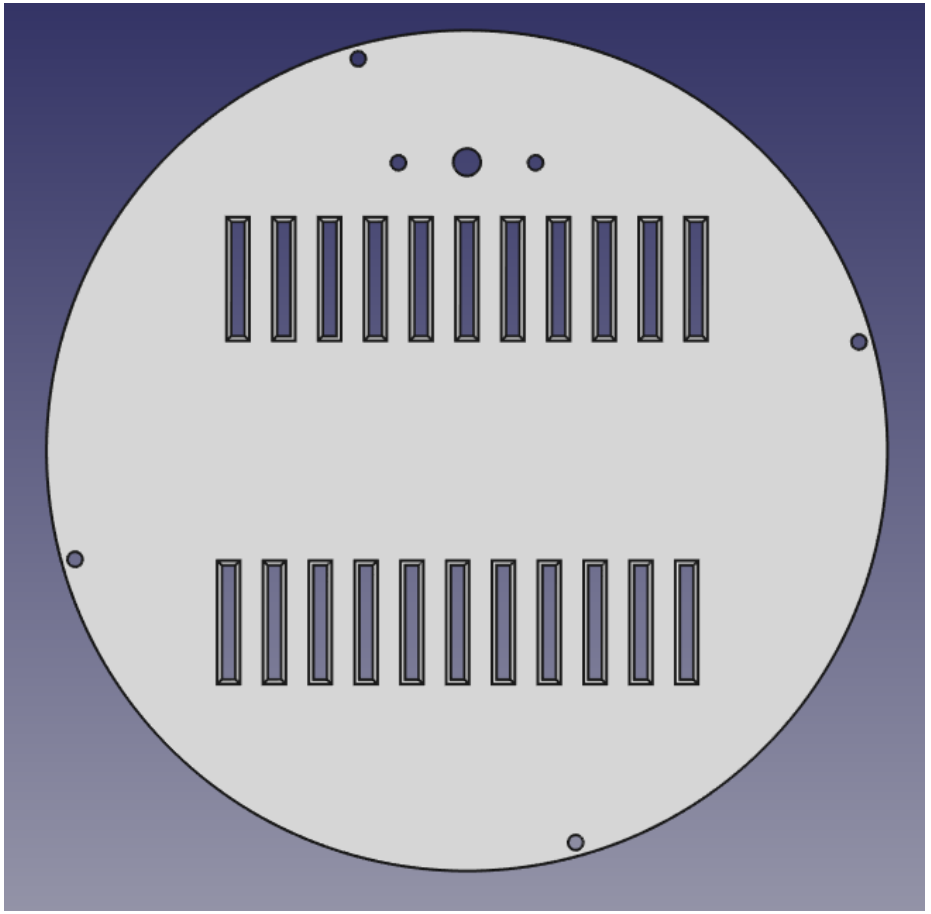
Rysunek 36: Widok izometryczny modelu zegara - tył



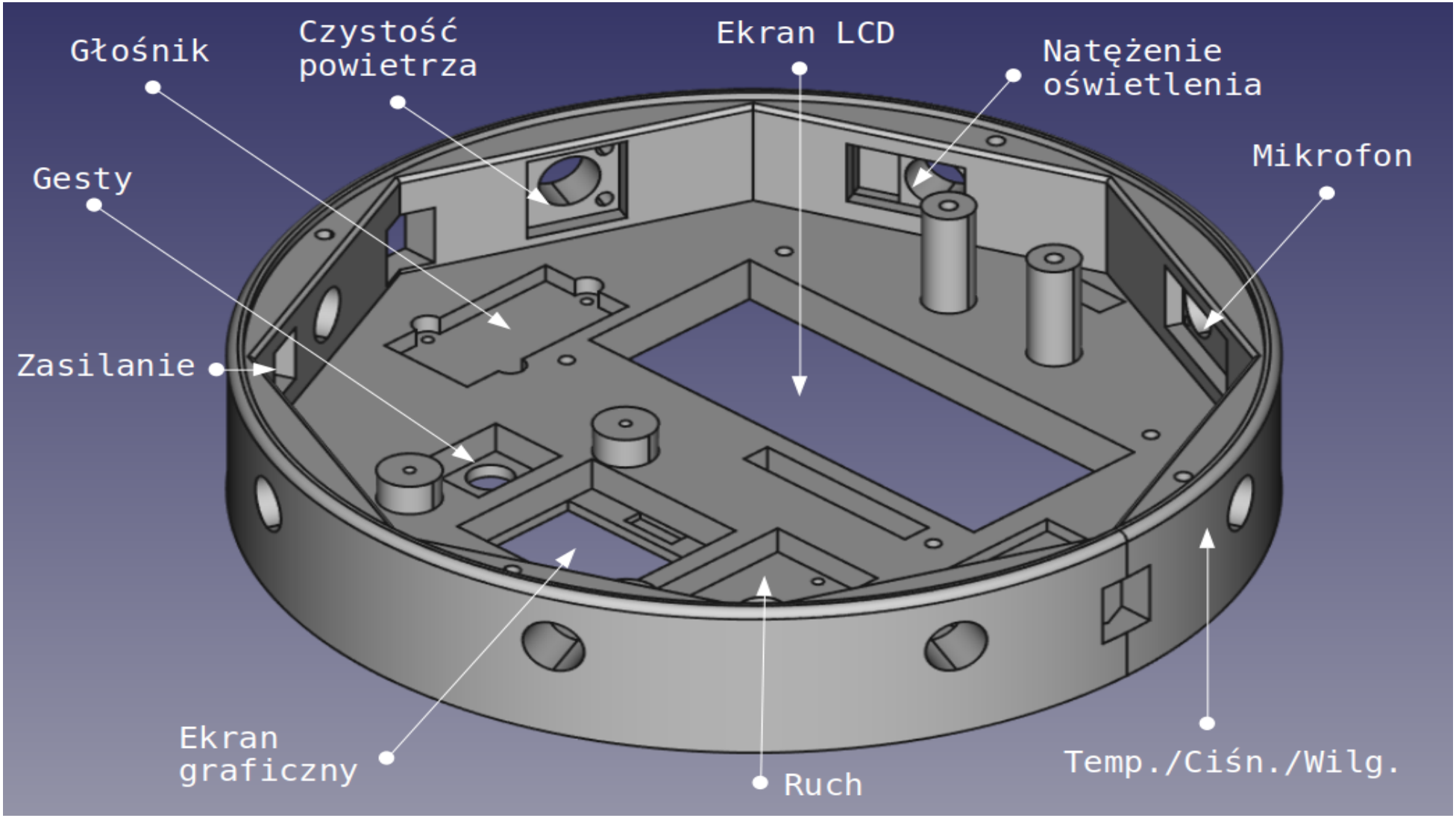
Rysunek 37: Widok modelu zegara z góry



Rysunek 38: Przekrój przez środkową płaszczyznę poziomą modelu zegara



Rysunek 39: Widok tylnej pokrywki modelu zegara



Rysunek 40: Rozmieszczenie czujników i innych komponentów wewnątrz obudowy

3.2 Warstwa oprogramowania – implementacja funkcjonalności

3.2.1 Inicjalizacja

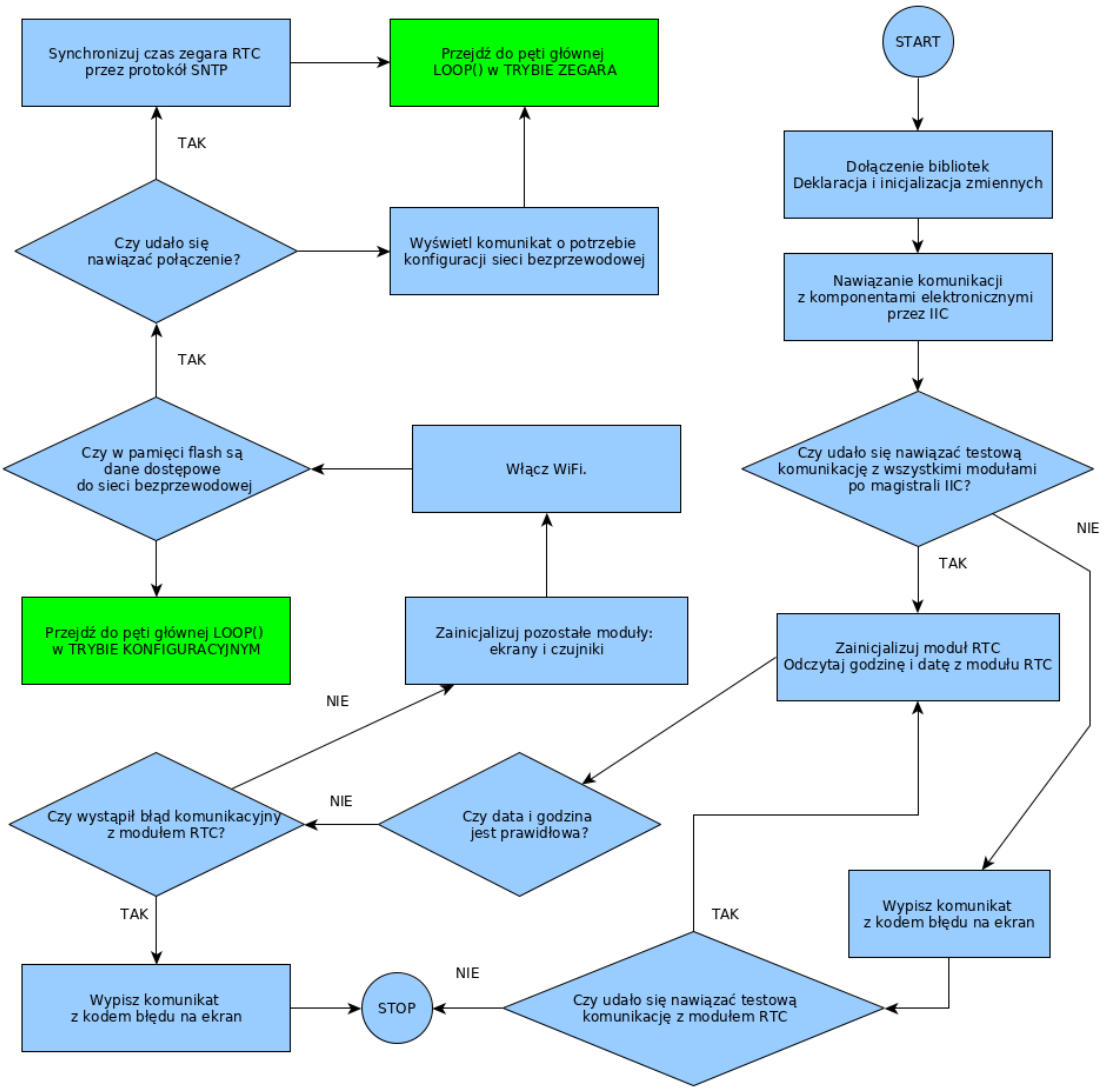
Po uruchomieniu urządzenia następuje wykonanie algorytmu uruchamiana zegara (rysunek nr 41) zaimplementowanego w ciele funkcji `setup()`. Zgodnie z nim, pierwszą informacją jaką otrzymuje użytkownik jest dostępność lub brak gotowości któregoś z 8 modułów sprzętowych. Brak kompletnej gotowości, poza komunikatem na ekranie, sygnalizowany jest przy pomocy dźwięku. Funkcja ta została zrealizowana poprzez skanowanie adresów z przedziału <8;127> (notacja dziesiętna) na magistrali IIC i zliczaniu prawidłowo zrealizowanych transmisji.

Kolejnym krokiem jest próba odczytania daty i godziny z modułu zegara czasu rzeczywistego w postaci uniksowego znacznika czasu jako 32 bitowej liczby całkowitej bez znaku. Uzyskanie powyższej liczby pozwala na dalsze wykonywanie algorytmu, w przeciwnym wypadku dochodzi do sprawdzenia wartości rejestru modułu RTC przechowującego kod ostatniego błędu. W przypadku braku błędu podnoszona jest flaga, dzięki której następuje próba nawiązania połączenia z siecią bezprzewodową i synchronizacji czasu. Odpowiada to sytuacji pierwszego uruchomienia zegara, kiedy to nie był on jeszcze synchronizowany, bądź wyczerpaniu się źródła energii, służącego do podtrzymania czasu. Jeżeli nie odczytano prawidłowo znacznika czasu i istnieją błędy, działanie programu jest przerywane. Odpowiada to sytuacji w której istnieją przekłamania w trakcie transmisji. Ten radykalny krok, wynika z założenia, że urządzenie nie będzie prawidłowo działało bez modułu zegara czasu rzeczywistego.

Następnie zadeklarowane wcześniej obiekty, odpowiadające za obsługę odpowiednich modułów sprzętowych, są inicjalizowane. Dokonano tego wzorując się na przykładach użycia dołączonych do każdej z bibliotek. Deklaracja tych obiektów ma charakter globalny i dokonywana jest przed blokiem `setup()`, co pozwala na wykorzystywanie ich w dowolnym potrzebnym kontekście w programie.

Na kolejnym etapie uruchamiana jest karta radiowa WiFi w trybie klienta zewnętrznego punktu dostępowego. Następuje próba nawiązania połączenia z siecią bezprzewodową za pomocą identyfikatora sieci SSID i hasła przechowywanego w pamięci flash zegara, po czym urządzenie uruchamia się w trybie zegara. Jeśli powyższe dane dostępowe nie są zdefiniowane urządzenie uruchamia się w trybie

konfiguracji celem ich ustanowienia przez użytkownika. Nieprawidłowe dane logowania, bądź brak możliwości nawiązania połączenia również skutkuje uruchomieniem trybu zegara, jednak na wyświetlaczu graficznym sygnalizowane to jest adekwatną ikoną. Oczywiste jest, iż nawiązane połączenia WiFi nie jest jednoznaczne z uzyskaniem dostępu do internetu, jednakże, ten wyjątek jest obsługiwany w czasie podejmowania takiej próby – przy synchronizacji czasu, aktualizacji danych pogodowych lub transmisji danych do brokera MQTT.



Rysunek 41: Algorytm inicjalizacji zegara LED

Źródło: opracowanie własne

3.2.2 Tryb konfiguracji i zarządzanie danymi konfiguracyjnymi

W celu realizacji funkcjonalności o symbolu FK tzn. trybu konfiguracji parametrów dostępowych do sieci WiFi postanowiono wykorzystać jedną z dostępnych w repozytorium Arduino bibliotek. Przetestowano 3 biblioteki i najbardziej stabilnie działającą i funkcjonalną okazała się *ConfigManager*[72]. Zgodnie z obowiązującą licencją MIT w udostępnionym kodzie można było wprowadzać zmiany. Autor pracy wykonał *fork* przytoczonej biblioteki i wprowadził drobne modyfikacje, dzięki czemu przejście w tryb konfiguracji stało się możliwe na żądanie, nawet w sytuacji, kiedy w pamięci nieulotnej znajdowały się prawidłowe dane dostępu do sieci bezprzewodowej. Ostatecznie tryb konfiguracji uruchamia się:

a) automatycznie w sytuacji opisanej w poprzednim podrozdziale, kiedy to dane logowania nie są dostępne,

b) ręcznie, dzięki zaimplementowanej możliwości wyzwolenia tego trybu przez wciśnięcie przycisku na obudowie zegara w trakcie jego uruchamiania.

Wprowadzone zmiany dotyczyły funkcji `setup()` w pliku `ConfigManager.cpp` i polegały na dodaniu nieopcjonalnego argumentu typu `bool`.

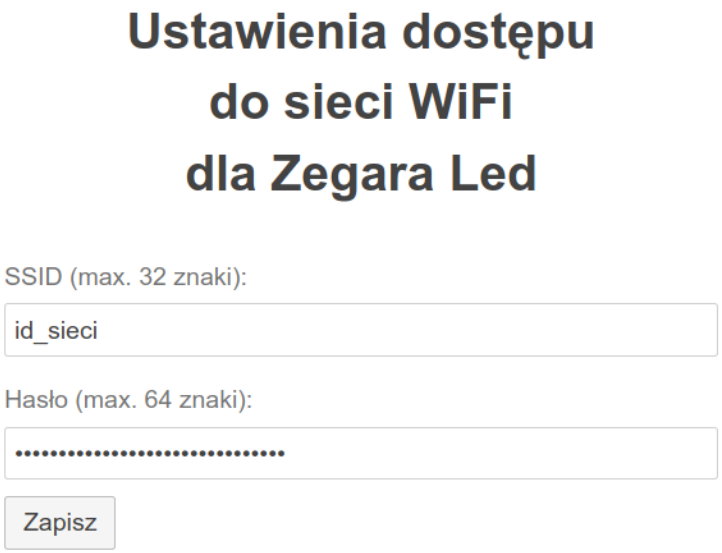
```
void ConfigManager::setup(bool force) {...}
```

Po wprowadzeniu zmian, jego wartość `true` powodowała wymuszenie uruchomienia zegara w trybie punktu dostępu, w którym przy pomocy zewnętrznego urządzenia np. smartfona można połączyć się z siecią „Zegar Led”. Po ustanowieniu połączenia, dzięki wykorzystaniu funkcji serwera DNS, każde zapytanie użytkownika przekierowuje do strony konfiguracji (rys. 42) dostępnej pod adresem IP 192.168.4.1. Tworzona sieć ma adres i maskę w postaci 192.168.4.0/24. Wartość `false` wprowadzanego argumentu powodowała te same działania, jedynie w sytuacji, gdy połączenie z siecią bezprzewodową, niezależnie od przyczyny, kończyło się niepowodzeniem. Wygląd ekranu LCD w tym trybie przedstawiono na rysunku nr 43.

Kod HTML strony konfiguracyjnej mieści się w jednym pliku, z zawartą w sobie sekcją kaskadowych arkuszy styków CSS, razem o wielkości 3465 bajtów. Zmiany wprowadzone w tym pliku, w stosunku do przykładu dołączonego w bibliotece, poza dodaniem deklaracji *charset* zestawu znaków i napisami w języku polskim, polegały na dodaniu ograniczeń w formularzu dotyczących pól SSID i hasła do 32 i 64

72 https://github.com/nrwiersma/ConfigManager

znaków odpowiednio[73]. Po wprowadzeniu danych i zatwierdzeniu przyciskiem „Zapisz” za pomocą metody POST protokołu HTTP generowanie jest zapytanie, które obsługiwane jest przez nasłuchujący serwer www tworzony przez *ConfigManager*. Przesłane dane dostępowe zapisywane są w pamięci EEPROM[74], a następnie zegar jest resetowany. Przy ponownym uruchomieniu, następuje próba połączenia zegara z zewnętrzną siecią WiFi za pomocą danych odczytanych ze wspomnianej nieulotnej pamięci.



Rysunek 42: Strona edycji danych dostępowych w trybie konfiguracyjnym

Źródło: opracowanie własne



Rysunek 43: Ekran LCD w trybie konfiguracyjnym

Źródło: opracowanie własne

73 IEEE Std 802.11-2016, https://standards.ieee.org/standard/802_11-2016.html, str. 135 i 790

74 EEPROM – skrót z ang. Electrically Erasable Programmable Read-Only Memory

Warto nadmienić, że wspomniany wyżej plik `index.html` został zapisany w pamięci flash urządzenia przy wykorzystaniu narzędzia *ESP8266 Data Sketch Upload*[75] dostępnym w Arduino IDE dopiero po jego dodatkowej instalacji.

Biblioteka *ConfigManager* poza wyżej opisaną obsługą zapisu i odczytu danych wprowadzonych przez użytkownika dotyczących sieci bezprzewodowych oferuje taką funkcjonalność w stosunku do dowolnych innych parametrów konfiguracyjnych. Jednakże, w specyfikacji wymagań pod symbolem FP, zawarto wymaganie, aby pozostałe parametry były wprowadzone w wygodniejszy sposób. Oznaczało to wykorzystanie aplikacji mobilnej pracującej w tej samej sieci WiFi co zegar, eliminując potrzebę przełączania użytkownika pomiędzy sieciami bezprzewodowymi.

Pierwszym krokiem do realizacji wymagania [FP] było stworzenie pliku `config.json` ze wstępną konfiguracją. Jego zawartość zaprezentowano na rys. nr 44. Opis budowy kluczy tego pliku konfiguracyjnego zawarto w podrozdziale 3.2.8.4. Od strony oprogramowania zegara zmiana ustawień polega na odebraniu ciągu znaków z aplikacji mobilnej, przesłanych przez sieć bezprzewodową, a następnie zapisania jej w pamięci flash jako plik `config1.json`. Następnie przy wykorzystaniu metody walidacyjnej biblioteki *ArduinoJson* zawartość tego pliku wczytywana jest do pamięci. Tylko po pomyślnej walidacji zawartości pliku pod kątem zgodności ze składnią JSON, następuje zmiana nazwy pliku z `config1.json` na docelowy `config.json`. Natychmiast po tej operacji następuje ponowne uruchomienie urządzenia w celu wczytania nowych parametrów. Taka procedura postępowania zapobiega pozostawieniu urządzenia z nieprawidłowym plikiem z preferencjami użytkownika i powodowaniu nieoczekiwanych zachowań urządzenia.

Pary klucz-wartość z poniższego rysunku nr 44 mają znaczenie:

adres URL serwera NTP (`ntp_pool_url`), szerokość (`fi`) i długość (`lambda`) geograficzna miejsca użytkowania zegara, adres URL oraz token API serwisu pogodowego, adres URL oraz token API do serwisu z zanieczyszczeniami i strefę czasową (`time_zone`). Następnie: opcja włączenia budzika świetlnego (`sunrise_clock`), godzinę jego pełnego rozświetlenia (`sunrise_clock_end`), która za razem jest godziną emisji dźwięku budzika oraz długość symulowanego wschodu

75 https://github.com/esp8266/arduino-esp8266fs-plugin/releases

słońca `sunrise_clock_offset`. Ostatni parametr `equalizer_bars_horizontal` daje opcję wyboru pomiędzy poziomą a pionową graficzną reprezentacją dźwięku w podziale na częstotliwości, wykorzystywaną w trybie efektów świetlnych.


```
{
   "ntp_pool_url!025s":"pl.pool.ntp.org",
   "fi!010f":"54.5188900",
   "lambda!010f":"18.5318800",
   "weather_api_url!150s":"https://api.darksky.net/forecast/{token}/{lat},{lon}?units=si&
exclude=minutely,hourly&lang=pl",
   "weather_api_token!050s":"",
   "pollution_api_url!150s":"https://api.waqi.info/feed/geo:{lat};{lon}/?token={token}",
   "pollution_api_token!050s":"",
   "time_zone!003d":"1",
   "sunrise_clock!001y":"t",
   "sunrise_clock_end!005h":"06:00",
   "sunrise_clock_offset!002d":"15",
   "equalizer_bars_horizontal!001y":"n"
}
```


Rysunek 44: Domyślna zawartość pliku konfiguracyjnego config.json

Źródło: opracowanie własne

3.2.3 Urządzenia i strumienie wejścia/wyjścia oraz sterowanie zegarem

W tym miejscu warto dokonać podziału ze względu na charakter komponentów z którymi użytkownik wchodzi w interakcję. Urządzenia wejścia i wyjścia zostały przedstawione odpowiednio w tabelach nr 20 i 21.

Tabela 20 - Urządzenia i strumienie wejścia

Wyszczególnienie	Miejsce	Rozdział	Zaimplementowane właściwości
przycisk funkcyjny	prawy bok	2.2.11	1-klik, 2-klik, długie przyciśnięcie
przycisk resetu	lewy bok	2.2.11	1-klik
czujnik gestów	front	2.2.9	lewo, prawo, góra, dół, przód, tył
czujnik ruchu PIR	front	2.2.7	wykrycie ruchu
smartfon/tablet	n. d.	3.2.9	wybór trybu głównego, a następnie opcji mu odpowiadających, wprowadzanie konfiguracji

Tabela 21 - Urządzenia i strumienie wyjścia

Wyszczególnienie	Miejsce	Rozdział	Zaimplementowane właściwości
pierścień diod LED	front	2.2.6	zmiana koloru, zmiana jasności, animacja punktów świetlnych
ekran LCD	front	2.2.2	znaki ASCII, samodzielnie zdefiniowane znaki (do 8szt./ekran), equalizer graficzny
ekran graficzny	front	2.2.1	znaki ASCII, polskie znaki, ikony, grafika, pasek postępu
głośnik	wnętrze	2.2.10	odtwarzanie melodii budzika
smartfon/tablet	n. d.	3.2.9	wyświetlanie zestawu danych pogodowych i danych o zanieczyszczeniach wew. i zew.

Jako uzupełnienie powyższego, w tabeli nr 23 zebrano informacje o stanie urządzeń wyjściowych w poszczególnych trybach pracy urządzenia. Numery trybów są tożsame z tymi, jakie przyjęto w implementacji.

Sterowaniem urządzeniem odbywa się w sposób trzytorowy tj.:

a) za pomocą gestów wykonywanych przed urządzeniem,

b) za pomocą aplikacji mobilnej,

c) przy użyciu przycisku funkcyjnego na obudowie.

Podstawową komendą sterującą jest zmiana trybu pracy. W obrębie danego trybu wyróżnia się zaś komendę wyboru opcji. Trzecią z dostępnych, podstawowych komend jest żądanie aktualizacji danych pogodowych i środowiskowych. Sposób implementacji wymienionych rozkazów sterujących w poszczególnych kanałach sterowania prezentuje poniższa tabela nr 22.

Wykorzystany moduł czujnika gestów jest w stanie rozpoznać 9 ruchów ludzkiej ręki: poruszanie w lewo, w prawo, w górę, w dół, do przodu, do tyłu, falę oraz ruch okrężny przeciwnie lub zgodnie ze wskazówkami zegara. W implementacji wykorzystano wszystkie gesty poza falą, z zastrzeżeniem, że ruch do przodu i do tyłu aktywuję tę samą funkcjonalność, opisaną poniżej. Powodem takiej decyzji jest zaobserwowane wzajemne wyzwalanie obu gestów, co w przypadku zastosowania ich w sposób niezależny jest zachowaniem mocno niepożądanym.

Tabela 22 - Komendy sterujące w zależności od kanału sterowania

Kanał/Kom.	Komenda zmiany trybu		Komenda wyboru opcji		Aktualizacja danych
Czujnik gestów	prawo	lewo	góra	dół	zatoczenie palcem okręgu zgodnie z ruchem wskazówek zegara
Przycisk funkcyjny	1-klik	-	2-klik	-	długie przyciśnięcie
Aplikacja	wybór z listy trybów	wybór z listy trybów	wybór z listy opcji	wybór z listy opcji	Inne→Aktualizacja danych

Z powodu oszczędności energii, urządzenie nie jest ciągle podłączone do sieci bezprzewodowej. Automatycznie łączenie odbywa się w przypadku pobierania danych pogodowo-środowiskowych oraz w trakcie przesyłania wyników pomiarów lokalnych do serwera MQTT. Aby wyzwolić **połączenie z siecią**, np. w celu zdalnego sterowania przez aplikację mobilną, użytkownik powinien wykonać **gest** przybliżając lub oddalając otwartą dłoń od zegara. Nawiązanie połączenia zostanie zobrazowane stosowną ikoną na wyświetlaczu (rys. 46, prawy górny róg wyświetlacza graficznego), a w zegarze zostanie uruchomiony serwer nasłuchujący na porcie 4210 datagramów UDP.

Najskuteczniejszą możliwość miękkiego restartu urządzenia przewidziano za pomocą dedykowanego przycisku na obudowie urządzenia. Poza tym, identyczny rezultat uzyskuje się po wykryciu gestu w postaci okrężnego ruchu przeciwnie do wskazówek zegara lub wybranie opcji w aplikacji mobilnej (Inne→Restart).

Jednym z powodów powstania aplikacji mobilnej była potrzeba udostępnienia użytkownikowi precyzyjnego i łatwego w użyciu interfejsu do wydawania komend. O ile podstawowy ich zestaw, wymieniony w tabeli nr 22 w pełni wystarcza do codziennego użytku, o tyle dodatkowe komendy przekazywane z aplikacji dopełniają kontrolę nad urządzeniem pozwalając na jego zdalne sterowanie. Budowę i wyjaśnienie komunikatów wymienianych pomiędzy aplikacją i zegarem opisano w podrozdziale 3.2.10.2 pt. „Komunikacja aplikacji z zegarem”.

Zastosowanie lekkiego protokołu komunikacyjnego UDP dla pakietów z komendami sterującymi przesyłanych w sieci LAN, nawet przy ograniczonych zasobach obliczeniowych NodeMCU, dało bardzo dobre rezultaty. Objawiają się one szybkim wykonaniem zdalnych rozkazów, przez co sterowanie jest skuteczne i płynne, robiąc dobre wrażenie na użytkowniku (z ang. *user experience, UX* **)**.

Tabela 23 - Stan urządzeń wyjścia w zależności od aktualnego trybu pracy

Nr trybu	Tryb:	Ekran LCD	Ekran graficzny	Pierścień diod	Aplikacja	Głośnik
0	konfiguracji	informacja o potrzebie konfiguracji	wyłączony	wyłączone	-	cichy
1	zegara	wybrane dane środowiskowe	aktualna data i godzina, ikona statusu połączenia z WiFi	animacja upływu czasu	lista wyboru trybów	cichy
2	monitoring śr. zew.	aktualne i prognozowane wartości wielkości pogodowych oraz zanieczyszczeń	grafika symbolizująca aktualną pogodę	wyłączone	zestaw danych środowiska zew.	cichy
3	monitoring śr. wew.	aktualne wartości mierzone w pomieszczeniu	grafika symbolizująca aktualną pogodę	wyłączone	zestaw danych środowiska zew.	cichy
4	efektów świetlnych	graficzna reprezentacja amplitud dźwięku z otoczenia dla 7 lub 4 częstotliwości	numer efektu	dynamiczne efekty świetlne	lista wyboru efektów	cichy
5	edukacyjny	wyłączony	znak, działanie lub grafika do odgadnięcia	animacja losowania	lista wyboru zagadek	cichy
6	palety kolorów	wyłączony	wyłączony	wybrany kolor	paleta barw	cichy
7	budzika	wyłączony	aktualna data i godzina, ikona statusu połączenia z WiFi	symulacja wschodu słońca	-	melodia

3.2.4 Tryb zegara

Spełniając uszczegółowione wymagania ze specyfikacji, tryb zegara ustanowiono jako domyślny, co ze względu na jego charakter można określić również mianem reprezentacyjnym. Po uruchomieniu, a także po upływie 15 minutowego okresu bezczynności, urządzenie przechodzi do niego samoczynnie.

Zgodnie z projektem, za odmierzanie czasu odpowiada moduł zegara RTC, którego obsługę na płaszczyźnie oprogramowania zapewnia biblioteka **RtcDS3231**. Wykorzystano fizyczne wyjście modułu o oznaczeniu SQW[76] ustawiając jego charakter na sygnał prostokątny (`pinMode`) o częstotliwości 1Hz (`freq`) za pomocą metod odpowiednio:

```
SetSquareWavePin(DS3231SquareWaveOut pinMode),
SetSquareWavePinClockFrequency(DS3231SquareWaveClock freq).
```

Tak uformowany sygnał okresowy podawany jest na pin 12 (D6) platformy NodeMCU, pracujący w roli wejścia. W celu właściwego i skutecznego wykrywania zmian tego sygnału, dla tego wejścia zdeklarowano przerwanie zewnętrzne przez metodę `attachInterrupt`, wskazując w jej argumentach nazwę procedury do okresowego uruchamiania oraz sposób wywoływania - po wykryciu zbocza rosnącego sygnału. Prawidłowe wywołania procedury następują dzięki ustawieniu jej atrybutu `ICACHE_RAM_ATTR`, dzięki czemu *linker* w procesie kompilacji powoduje, iż kod maszynowy ją reprezentujący jest przechowywany w pamięci RAM. Istotny jest również czasowy i logiczny zakres obowiązywania, gdyż przerwanie jest odłączane metodą `detachInterrupt` po zmianie na inny tryb pracy i ponownie deklarowane po przejściu do trybu zegara. Jednym z zadań opisywanej procedury ISR[77] o nazwie `one_second`, wyzwalanej przez przerwania z częstotliwością 1Hz, jest tworzenie dwóch ciągów znaków reprezentujących aktualną godzinę w formacie HH:MM oraz datę i wyświetlanie ich na ekranie graficznym. Jednakże, jej fundamentalnym celem jest wyzwalanie animacji upływu czasu na pierścieniu diod LED i aktualizacja wartości trzech zmiennych, odpowiedzialnych za informację o aktualnej „pozycji" końców wskazówek na 60 diodowej tarczy zegara. Jedno-bajtowe zmienne o nazwach

76 SQW - skrót z ang. Square Wave

77 ISR – skrót z ang. Interrupt Service Routines, https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

`led_second`, `led_minute` oraz `led_hour` zostały zadeklarowane globalnie z użyciem atrybutu `volatile`[78], przez co są prawidłowo współdzielone pomiędzy programem głównym a procedurą ISR[79].

Wspomniana animacja, polega na płynnej zmianie położenia świecących diod wraz z upływem czasu. Świecące punkty reprezentują wskazówki sekundową, minutową i godzinową o kolorach odpowiednio niebieskim, zielonym i czerwonym. W oparciu o bibliotekę **FastLED** zaimplementowano obsługę pierścienia LED, co w omawianym trybie przekładało się wstępnie na zaświeceniu punktów świetlnych na wyliczonych (w procedurze `one_second`) pozycjach na okręgu, odpowiadających aktualnemu czasowi. Korzystając z jednej z wielu zalet biblioteki, w postaci zdefiniowanej ośmiobitowej (przez co szybkiej w wykonywaniu) matematycznej funkcji piło-kształtnej, obliczana jest aktualna jasność źródła światła (V), a następnie zadawana do wyświetlenia w postaci reprezentacji koloru HSV[80] . Maksymalna wartość jasności 255 przypada na górny „ząb" piły, co w czasie przekłada się na połowę okresu, czyli 500ms od wyzwolenia przejścia diody z jednej pozycji na kolejną. Warto podkreślić, iż w implementacji uwzględniono nakładanie się na siebie „wskazówek", zawierając w kodzie źródłowym odpowiednie warunki na przygasanie i rozświetlanie diod w momencie wejścia i wyjścia z jednakowej pozycji.

Niezmiernie ważne z użytkowego punktu widzenia, stało się dostosowywanie maksymalnej jasności wyświetlanej barwy, w oparciu o warunki oświetleniowe w pomieszczeniu. Temu celowi został przeznaczony dedykowany komponent elektroniczny (por. podrozdział 2.2.5) – czujnik natężenia światła BH1750. Co ustalony okres długości 3 sekund, posługując się metodami z biblioteki **BH1750**, odczytywana jest wartość natężenia światła i po odpowiednim przeskalowaniu aplikowana jako argument funkcji `FastLED.setBrightness(uint8_t brightness)`. Niestety, dostępna w Arduino implementacja funkcji mapującej `map` o deklaracji:

```
long map(long x, long in_min, long in_max, long out_min, long out_max);
```

nie spełniała tu właściwie swojej roli, więc autor pracy dokonał jej implementacji samodzielnie. Poprawy wymagała obsługa zakresu argumentu wejściowego `x` w przypadku, gdy nie należał on do przedziału `<in_min;in_max>`. Po zmianach,

78 Ibidem

79 Barr M., Massa A., Programming Embedded Systems, str. 157-159

80 HSV z ang. Hue Saturation Value

w sytuacji wystąpienia wspomnianego wyjątku, zgodnie z oczekiwaniami funkcja zwracała odpowiednią ze skrajnych wartości przedziału. Ostatecznie, w wyniku empirycznej oceny, za przedział do skalowania w ciele liczb ośmiobitowych przyjęto <50;255>, co w słabych warunkach oświetleniowych dawało dostatecznie widoczną, nie drażniącą oczu maksymalną jasnością, animację upływu czasu.

Podążając za zapisami specyfikacji, dopełnieniem danych o aktualnej dacie i godzinie, stały się wybrane parametry środowiskowe i astronomiczne, wyświetlane na ekranie LCD przez czas około 1 minuty po wykryciu ruchu przed zegarem. Ich odświeżanie dokonuje się co 15 sekund, ze względu na wyświetlaną tam temperaturę wewnątrz pomieszczenia. Gdyby nie ta wielkość, częstotliwość ta mogłaby być mniejsza ze względu na okres aktualizacji danych pogodowych, o czym szerzej napisano w podrozdziale 3.2.5. Do wykrywania ruchu fizycznie użyto czujnika PIR (por. podrozdział 2.2.7), a programowo wysoki stan jego wyjścia podawanego na pin nr 14 (D5) platformy NodeMCU. Czas stanu wysokiego, oznaczającego wykrycie ruchu, został zadany poprzez potencjometr konfiguracyjny na płytce czujnika. W implementacji do wykrycia zmian tego sygnału ponownie posłużono się przerwaniami, tym razem jednak wyzwalanymi zboczem rosnącymi i opadającym sygnału. W konsekwencji po wykryciu ruchu i osiągnięcia stanu wysokiego na wejściu, podświetlenie zegara LED zostaje programowo włączone, a po przejściu w stan przeciwny – wyłączone. Widok podświetlonego wyświetlacza w trybie zegara został przedstawiony na poniższym rysunku nr 45.



Rysunek 45: Zawartość ekranu LCD w trybie zegara

Źródło: opracowanie własne

Zaprezentowane symbole, usytuowane przed poszczególnymi wartościami liczbowymi, nie są dostępne w bibliotece zarządzającej wyświetlaczem LCD **LiquidCrystal_I2C** (ograniczenie obejmuje również widocznego napisu „°C” w postaci jednego znaku). Umożliwia ona jednak zdefiniowanie do 8 własnych symboli o wymiarach 8x5 pikseli przypadających na każdy ekran. Definicje tych dedykowanych znaków zostały wygenerowane przy użyciu arkusza kalkulacyjnego Calc z pakietu LibreOffice. Stworzono w nim matrycę 8x5 kwadratów, a następnie po zaprojektowaniu znaku przez wypełnianie kolorem kwadratów, zliczano wartości poszczególnych wierszy. Każdemu z wierszy o szerokości 5 kwadratowych „pikseli” odpowiadała więc liczba od 0 do 2^5-1 w systemie dziesiętnym, a więc od 00 (pusty wiersz) do 1F (pełny wiersz) w systemie heksadecymalnym. Dla przykładu definicja symbolu postaci ludzkiej wyświetlonej w lewym dolnym rogu ekranu rysunku nr 45, zapisana w postaci 8-elementowej tablicy 1 bajtowych liczb, przyjmuje postać:

```
uint8_t t_feels[8] ={0x04,0x0A,0x04,0x1F,0x04,0x04,0x0A,0x11};
```

Wyjaśniając, symbol ten oznacza temperaturą odczuwalną na dworze. W analogiczny sposób zaprojektowano kolejne znaki, które przez wywołanie metody:

```
lcd.createChar(<numer znaku 0-7>, <nazwa_tablicy_znaków>);
```

tworzyły w pamięci ekranu kolejne z 8-elementowego zestawu znaków, określanego dynamicznie po inicjalizacji trybu zegara. Dopiero po takim zabiegu zaprojektowane znaki mogły zostać prawidłowo wyświetlone na zadanej pozycji. Opisane postępowanie wymuszone było koniecznością zdefiniowana odrębnych zestawów do trybu efektów świetlnych. Generowane tam widmo sygnału akustycznego i użyte w nim znaki zostały przedstawione na rysunku nr 53.

W podsumowaniu trybu zegara (który w adekwatnej zmiennej kodu źródłowego otrzymał wartość 1), na poniższym rysunku 46 został przedstawiony widok pracującego pod jego kontrolą urządzenia. Tym sam całkowicie wykonane zostały funkcjonalności oznaczone symbolami [F1] i [F2], częściowo [F4] oraz zrealizowane wymagania niefunkcjonalne [C8] i [C13].



Rysunek 46: Widok urządzenia w trybie zegara

Źródło: opracowanie własne

3.2.5 Tryb stacji pogodowej

Zadaniem trybu stacji pogodowej jest dostarczenie użytkownikowi zestawu aktualnych danych pogodowych, dla lokalizacji określonej w ustawieniach, wraz z prognozami 4h i 24h. Uzupełnieniem informacji tekstowych ma być ikona obrazująca pogodę, wyświetlona na ekranie graficznym o wymiarach 128x64 pikseli.

Spośród kilku dostępnych serwisów oferujących usługi dostarczania danych pogodowych i danych o zanieczyszczeniach środowiska (por. tabela nr 24), wybrano dwa spełniające wymaganie o bezpłatnym dostępie przy minimalnej częstotliwości odświeżania 90/24h. Serwisy *Dark Sky* i *WAQI* wyróżniały się na tle konkurencyjnych

ilością dostępnych parametrów pogodowych, środowiskowych i astronomicznych wraz z prognozami godzinnymi, co przesądziło o ich wyborze. Poza tym, autor pracy wyrobił sobie o nich pozytywne zdanie podczas realizacji agregatora danych pogodowych[81] na wcześniejszym etapie studiów. Obie usługi dostarczają wyniki zapytania ustrukturyzowane zgodnie z formatem wymiany danych JSON.

Tabela 24 - Porównanie usług serwisów pogodowych

	APIXU	DarkSky	OpenWeatherMap	Weatherbit	WAQI	Airly	Airvisual
aktualna pogoda dla miasta	✔		✔	✔	✔		
aktualna pogoda dla wsp. geogr.	✔	✔	✔	✔	✔	✔	✔
prognoza pogody dla miasta	✔		✔	✔			
prognoza pogody dla wsp. geogr.	✔	✔	✔	✔		✔	
dane o zanieczyszczeniach					✔	✔	✔
ograniczenie wykorzystania API	330/24h	1000/24h	86400/24h	1000/24h	1440/24h	1000/24h	330/24h

Źródło: opracowanie własne [81]

Proces pozyskania danych z serwisów zewnętrznych został zaimplementowany w sposób uogólniony, przez co niedużym nakładem pracy jest możliwe zmiana dostawcy usług lub dostosowanie do ewentualnych zmian w API. Rozpoczyna się on od zbudowania adresu URL poprzez podmianę ciągów znaków `{token},{lat}, {lon}` na wprowadzone przez użytkownika w ustawieniach tokeny serwisów i współrzędne geograficzne. Kolejnym krokiem jest ekstrakcja ciągu znaków oznaczającego *host* z adresu URL i nawiązanie połączenia (wykorzystując **ESP8266WiFi)**. W przypadku skutecznego nawiązania połączenia i prawidłowego sformułowanych parametrów (wykonano na podstawie dokumentacji API dla deweloperów z *Dark Sky*[82] i *WAQI*[83]), zapytanie zwraca pożądany rezultat w postaci pliku JSON. Problem przetwarzania tego pliku w systemie z ograniczonymi zasobami został rozwiązany przez wyodrębnienie istotnych jego fragmentów (aktualna pogoda, prognoza 4h i prognoza 24h), a następnie przez wykorzystanie narzędzi dostępnych w niezwykle użytecznej bibliotece **ArduinoJson.h**. W następstwie dokonywana jest walidacja ciągów znaków zawierających fragmenty odpowiedzi z danymi, które w zamiarze mają stanowić składniowo poprawne obiekty JSON[84]. W przypadku pomyślnego rezultatu, parsowane dane rozumiane jako wartości wielkości fizycznych zapisywane są do pól obiektów klasy `EnvVal`, o budowie zgodnej z rysunkiem nr 4.

81 https://github.com/gdaPythonProjects/weather2excel
82 https://darksky.net/dev/docs
83 https://aqicn.org/json-api/doc/
84 Blanchon B.,Mastering ArduinoJson - Efficient JSON serialization for embedded C++, str 60-67

Obiekty te zaś przechowywane są w strukturze `EnvData` pokazanej na rys. nr 47. W trakcie operacji sieciowych, które na urządzeniach o stosunkowo niewielkiej mocy obliczeniowej są czasochłonne, dla komfortu użytkownika na ekranie LCD prezentowany jest animowany pasek postępu wraz z jego aktualną procentową wartością (rys. nr 48).

```
struct EnvData {
  unsigned long weatherUpdateTime[3];
  unsigned long pollutionUpdateTime;
  ENVVAL pressure;
  ENVVAL humidity;
  ENVVAL temperature;
  ENVVAL apparentTemperature;
  ENVVAL cloudCover;
  ENVVAL windSpeed;
  ENVVAL windBearing;
  ENVVAL precipIntensity;
  ENVVAL precipAccumulation;
  ENVVAL visibility;
  ENVVAL uvIndex;
  ENVVAL CO;
  ENVVAL NO2;
  ENVVAL O3;
  ENVVAL PM10;
  ENVVAL SO2;
  ENVVAL PM25;
};
```

Rysunek 47: Struktura danych EnvData

Źródło: opracowanie własne



Rysunek 48: Prezentacja postępu operacji sieciowych w tle

Źródło: opracowanie własne

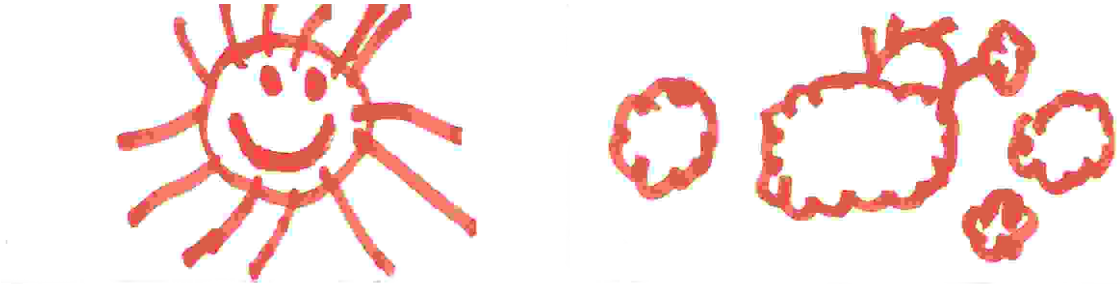
W trybie pogodowym, po pomyślnej aktualizacji danych środowiskowych, następuje ich prezentacja w tabeli z cyklicznie zmieniającą się zawartością. Z racji powtarzalnego zadania formatowania i wyświetlania, w klasie wielkości fizycznych `EnvVal` stworzono metodę `drawScreen`, której zadaniem jest właściwe wypełnianie tabeli. Iteracja po obiektach `EnvVal` struktury `EnvData` i wywołanie tej metody w zgrabny i skuteczny sposób pozwalają na efektywne wykonanie opisywanego zadania. Jego rezultat został uwidoczniony na ekranie LCD z rysunku nr 50. Aktualizacja danych następuje co 15 min. Niezależnie od trybu w jakim jest urządzenie, proces aktualizacji przerywa go, wyświetla postęp i aktywuje go ponownie po pobraniu i przetworzeniu danych z internetu. Dodatkowo, w celu podniesienia wartości użytkowych, w trakcie prezentacji pojedynczej tabeli z danymi trwającej 6 sekund, wzdłuż ekranu graficznego generowany jest pasek postępu.

Jedną z informacji pozyskiwanych ze źródła *Dark Sky* jest nazwa ikony określającej pogodę. Wykorzystując 10 rysunków dostarczonych przez zamawiającego (przykład na rys. 49), stworzono pliki o rozmiarze 128x64 piksele, do wyświetlenia na ekranie graficznym. Rysunki po zeskanowaniu zostały przetworzone do 1-bitowej palety kolorów za pomocą programu graficznego GIMP. Następnie powstały obraz rastrowy został wyeksportowany do pliku w formacie *.xbm*, którego zawartość mogła być użyta w zmiennej tablicowej definiującej poszczególne ikony w programie. Przykładowa deklaracja i inicjalizacja tablicy jedno-bajtowych wartości wyglądała następująco:

```
static const unsigned char clear_day[] PROGMEM= {0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x0c, 0x70, 0x00, 0x06, 0x78, 0x18 (…) }
```

Zastosowany modyfikator PROGMEM[85] powoduje już na etapie kompilacji zapis danych do pamięci flash i odczyt z niej w trakcie wykonywania programu zamiast domyślnie używanej pamięci SRAM. Działanie to skutecznie przyczynia się do efektywnego gospodarowania dostępną pamięcią operacyjną.

85 https://www.arduino.cc/reference/tr/language/variables/utilities/progmem/



Rysunek 49: Przykładowe skany obrazów pogody

Źródło: opracowanie własne



Rysunek 50: Widok urządzenia w trybie stacji pogodowej

Źródło: opracowanie własne

3.2.6 Monitoring pomieszczenia i transmisja danych do brokera MQTT

Według specyfikacji, produktami opisywanego trybu monitoringu są: temperatura, ciśnienie, wilgotność, natężenie oświetlenia, a także stężenie związków organicznych w pomieszczeniu w którym znajduje się urządzenie. Pozyskane z najbliższego otoczenia informacje powinny być wyświetlane na ekranie LCD. [F3]

Dane pobierane są z czujników BMP280, BH1750 oraz CCS811 przy użyciu dedykowanych bibliotek wyszczególnionych w podrozdziale 2.2. Aktualizacja danych następuje co 7 sekund, o ile aktywne jest wejście z czujnika ruchu lub w przypadku otrzymania stosownej komendy z aplikacji mobilnej.

Co ustalony czas 15min, dokładnie taki jak aktualizacja danych pogodowych, korzystając z nawiązanego wtedy połączenia z siecią bezprzewodową, następuje przesyłanie wartości mierzonych parametrów do serwera MQTT. Adres URL oraz dane logowania do brokera odczytywane są z ustawień użytkownika. Obsługę wiadomości przesyłanych za pomocą tego protokołu zaimplementowano używając biblioteki **PubSubClient**. Zestawienie przesyłanych wielkości fizycznych wraz z jednostkami oraz tematami w jakich są publikowane przedstawiono w tabeli nr 25.

Tabela 25 - Zestawienie danych publikowanych do brokera MQTT

LP.	Nazwa	Jednostka	Topic (Temat)
1	Temperatura	°C	ledclock/temperature
2	Ciśnienie	hPa	ledclock/pressure
3	Wilgotność względna	%	ledclock/humidity
4	Natężenie oświetlenia	lux	ledclock/illuminance
5	TVOC	ppb[86]	ledclock/tvoc
6	eCO_2	ppm[87]	ledclock/eco2
7	Ruch	-	ledclock/movement

Warte uwagi jest iż, celem uzyskania dokładniejszych rezultatów z czujnika CCS811, wykorzystano jego możliwość kompensacji wyników pomiarów

86 ppb – skrót z ang. parts per milion, stężenie wyrażone w ilości cząstek na milion

87 ppm – skrót z ang. parts per bilion, stężenie wyrażone w ilości cząstek na miliard

przez ciśnienie i temperaturę. Przed odczytem danych metodami `getTVOC()` i `getCO2()` wywoływana jest funkcja:

```
setEnvironmentalData( float relativeHumidity, float temperature )
```

w której za argumenty podstawiane są odpowiednio wartości wilgotności względnej i temperatury pozyskane z czujnika BMP280. Ostatecznie, jeden z dwóch widoków ekranu LCD w trybie monitoringu parametrów pomieszczenia, przedstawiono na rysunku nr 51.



Rysunek 51: Widok ekranu LCD w trybie monitoringu pomieszczenia, widok 1/2

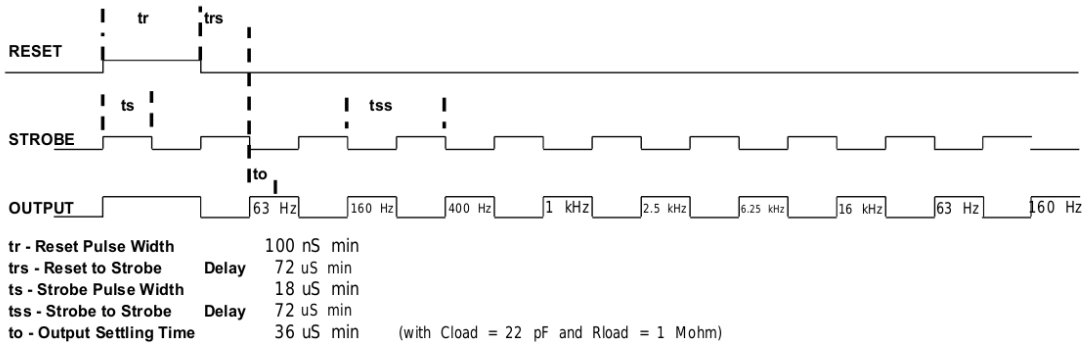
Źródło: opracowanie własne

3.2.7 Tryb efektów świetlnych

W celu wykonania zadania [F5] ze specyfikacji tj. generowania efektów świetlnych w takt dźwięków z otoczenia, wykorzystano zbudowany tor przetwarzania dźwięku. Składał się z modułu mikrofonu ze wzmacniaczem MAX9814, którego zmienny sygnał podawany był do 7 kanałowego equalizera MSGEQ7, a jego wyjście do konwertera analogowo-cyfrowego NodeMCU. Do obsługi tego equalizera wykorzystano dwa piny układu mikrokontrolera, pracujące w trybie wyjścia, sterując wejściami RESET i STROBE układu MAX9814.

Wzorując się na schemacie czasowym z karty katalogowej equalizera (rys. 52), stworzono procedurę do odczytu wartości proporcjonalnych do amplitud poszczególnych analizowanych częstotliwości. Na jego wejściu o nazwie RESET programowo wyzwalana jest zmiana sygnału z wysokiego na niski. Następnie postępuje sekwencyjny odczyt w pętli, z wyjścia OUTPUT poprzez wejście A0 mikrokontrolera, 7 wartości przekonwertowanych do wartości całkowitej z przedziału <0;1023>.

Pozyskane w ten sposób dane zapisywane są do 7-elementowej tablicy liczb 16 bitowych, a następie wykorzystywane do wyzwalania sekwencji świetlnych.



Rysunek 52: Schemat czasowy equalizera MSGEQ7

Źródło: Seven Band Graphic Equalizer Data Sheet, 9/2004, http://mix-sig.com

Do wysterowania diod LED RGB wykorzystano dwie biblioteki, obsługujące zintegrowane źródła światła WS2812B, a mianowicie **FastLED** oraz **WS2812FX**. Każda z nich posiadała zbiór gotowych do użycia przykładów animacji diod, co też zostało wykorzystane. Ostatecznie efekty od numeru 0 do 14 pochodzą z **FastLED**, kolejne zaś od numeru 15 do 70 zaczerpnięto z **WS2812FX**. Ostatni efekt nr 71 został wymyślony i zaimplementowany przez autora pracy.

Efekty animacji wykorzystują odczytane wartości amplitud dla niskich częstotliwości 63Hz i 160Hz. Czas wyzwolenia kolejnej klatki animacji jest odwrotnie proporcjonalny do wartości amplitudy najniższej częstotliwości. Wyzwolenie następuje także w przypadku, gdy średnia arytmetyczna amplitud 63Hz i 160Hz przekroczy stałą zdefiniowaną wartość progową, co ma na celu dodatkowe zwiększenie szybkości dla dźwięków z niskiego pasma. Poza tym, klika efektów wykorzystuje zmierzone wartości do wysterowania jasności wyświetlanych sekwencji.

Autorski efekt nr 71, zapala na dwóch półokręgach ilości diod proporcjonalne do amplitud najniższej (63Hz) oraz najwyższej (16kHz) częstotliwości. Wykorzystano kolory odpowiednio czerwony i zielony. Poza tym, dla każdej częstotliwości zaprogramowano opóźniony ruch opadający piksela w kolorze białym, który jest „podbijany" przez wymienione kolorowe diody. Widok zegara w trybie generowania jednego z efektów przedstawiono na poniższym rysunku. Kilka z pozostałych zostało nagranych na filmie demonstracyjnym na dołączonej do pracy płycie CD.

Równolegle do efektów świetlnych z pierścienia 60 diod LED, na ekranie LCD prezentowany jest equalizer graficzny. Występuje on w opcji poziomej i pionowej wybieranej w ustawieniach aplikacji. Analogicznie do własnych symboli pogodowych z podrozdziału 3.2.4, zdefiniowano 8 znaków przedstawiających wypełniający się prostokąt 8x5 od wysokości 1 linii aż do 8 linii poziomych. Natomiast w opcji equalizera poziomego prostokąt składał się od 1 do 5 linii pionowych. Zostało to następnie wykorzystane do wyświetlenia pasów o długości proporcjonalnej do amplitudy, reprezentujących poszczególne częstotliwości.



Rysunek 53: Widok urządzenia w trybie efektów świetlnych

Źródło: opracowanie własne

3.2.8 Tryb edukacyjny

W trybie edukacyjnym dostępne są zagadki w jednej z trzech kategorii na różnych poziomach, co zostało przedstawione w poniższej tabeli.

Tabela 26 - Rodzaje zagadek w trybie edukacyjnym

Rodzaj zagadki	Nazwa poziomu	Opis
Zagadki językowe	Łatwy	Wielkie litery alfabetu łacińskiego
	Normalny	Wielkie litery alfabetu łacińskiego + polskie znaki
	Trudny	Wielkie i małe litery + polskie znaki
	Bardzo trudny	Wielkie i małe litery, cyfry, znaki specjalne
	Extra	Słowo od 2 do 5 liter
Zagadki matematyczne	Poziom 1	Dodawanie
	Poziom 2	Odejmowanie
	Poziom 3	Mnożenie
	Poziom 4	Dzielenie
	Poziom 5	Mniejsze, większe, równe
Zagadki graficzne	Zwierzęta	Kształty zwierząt

W trakcie losowania w obrębie wybranej zagadki, na ekranie graficznym pojawiają się losowe symbole, a diody mienią się losowymi kolorami obejmując coraz większą część obwodu, aż do osiągnięcia pełnego okręgu. Po wyświetleniu wylosowanej wartości na ekranie graficznym, diody są przez 3 sekundy stopniowo wygaszane.

Od strony implementacji, wszystkie zagadki, wykorzystują liczby pseudolosowe generowane po stronie zegara. W zależności od rodzaju zagadki, wartości te są używane do wyświetlenia odpowiedniej litery, liczby, działania matematycznego i dwóch liczb oraz znaku graficznego. Tablica znaków, której podzbiory są używane do określenia kolejnych podzbiorów została przedstawiona poniżej:

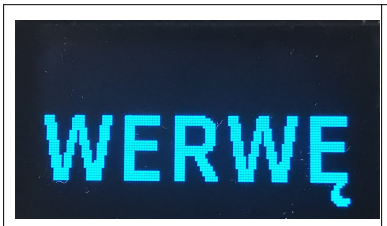
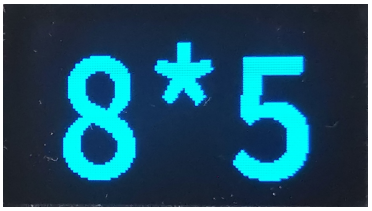
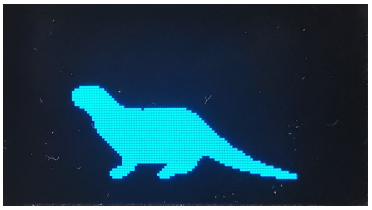
"ABCDEFGHIJKLMNOPQRSTUVWXYZÓĄĆĘŁŃŚŹŻabcdefghijklmnopqrstuvwxyzóąćęłńśźż+-*=.,!?@$0123456789"

Warto podkreślić, że fragment słownika[88] (wyrazy od 2 do 5 liter, z ręcznie wykluczonymi wulgaryzmami) został zawarty w zasobach aplikacji mobilnej i tam wylosowane słowo jest przesyłane do zegara. Jest to plik tekstowy o wielkości 259.4 kB zawierający 36159 słów. Sytuacja ta dotyczy zagadek językowych na najtrudniejszym poziomie o nazwie „Extra".

Szczególną uwagę zwraca ostatni z wymienionych rodzajów, przeznaczony dla najmłodszych dzieci. Źródłem kształtów zwierząt jest czcionka *animal.ttf* [89] przekonwertowana narzędziem wiersza poleceń *bdfconv*[90] do postaci odpowiedniej tablicy znaków, wykorzystywanym przez bibliotekę **U8g2** dedykowaną wyświetlaczowi graficznemu. Analogiczne działania zostały wykonane w stosunku do czcionki *Falling-Sky-Condensed* [91]. Została ona poddana konwersji i zapisana w 3 rozmiarach, aby właściwie wyświetlić długie ciągi znaków. Zaimplementowano to w zagadkach matematycznych w przypadku działaniach na liczbach dwucyfrowych oraz na wspomnianym poziomie „Extra" dla słów 5-literowych. Wspomniane zabiegi dopasowywania czcionek mają na celu maksymalne wykorzystanie powierzchni ekranu i jak największą czytelność wyświetlanych znaków.

Jako podsumowanie omawianego trybu przedstawiono 3 zdjęcia ekranu graficznego, prezentujące zrealizowaną funkcjonalność [F8].

Tabela 27 - Przykładowe zagadki w trybie edykacyjnym

		
Zagadka językowa	Zagadka matematyczna	Zagadka graficzna

88 https://sjp.pl/ - Słownik języka polskiego, ortograficzny, wyrazów obcych i słownik do gier
89 https://www.dafont.com/animals.font
90 https://github.com/olikraus/u8g2/tree/master/tools/font/bdfconv
91 https://www.whatfontis.com/FF_Falling-Sky-Condensed.font

3.2.9 Tryb budzika dźwiękowego i świetlnego

O zadanym w ustawieniach przez użytkownika czasie (HH:MM) w dni powszednie wyzwalany jest budzik dźwiękowy. Urządzenie emituje wtedy dźwięki melodii utworu „Take on me" zespołu A-ha. Jedynym sposobem na wyłączeniem budzika, poza odłączeniem zasilania, jest długie wciśnięcie przycisku funkcyjnego.

Dostępna w module zegara RTC funkcjonalność w postaci dwóch alarmów[92] (z dokładnością do sekund lub minut) nie mogła zostać wykorzystana, gdyż jego wyjście SQW zostało skonfigurowane jako sygnał prostokątny o częstotliwości 1Hz (por. 3.2.4 „Tryb zegara"). W tej sytuacji sprawdzanie warunku wyzwalania budzika dźwiękowego zostało zaimplementowane w procedurze ISR `one_second`, dla wywołań w których aktualna sekunda ma wartość 0 (tzn. pełne minuty). Aranżacja melodii została wykonana przez Edwarda Truonga, a implementacja dokonana przez Robsona Couto[93]. Melodia została zapisana w tablicy w postaci 2 bajtowych liczb całkowitych (oznaczających poszczególne nuty) naprzemiennie z interwałami. Następnie wykorzystywana jest dostępna na platformie Arduino funkcja **`tone`** do generowania dźwięków na podstawie zdefiniowanych nut i **`delay`** do wykonania interwałów. Pomimo, iż jest to funkcja blokująca, nie zaburza ona działania urządzenia, gdyż w trakcie generowania melodii nie są wykonywane żadne animacje, a przerwania działają w sposób niezakłócony[94].

Urządzenie „**Zegar LED**" posiada opcjonalną funkcję budzika świetlnego. Jej włączenie oraz domyślny czas trwania 15 minut można zmienić w ustawieniach (por. 3.2.10.4). Celem funkcji jest stopniowe rozświetlenie tarczy diod LED w zadanym czasie przed budzikiem dźwiękowym, jako imitacja wschodu słońca. Uzasadnienie powstania tej funkcji jest głębiej niż tylko wizualnie umotywowane, gdyż oko poza przekazywaniem mózgowi sygnałów związanych z obrazem, może zarządzać zegarem biologicznym człowieka[95]. Budzik świetlny może też wspomagać łagodzenie sezonowych zaburzeń afektywnych[96]. Uwzględniając zjawisko większego rozproszenia

92 Karta katalogowa układu DS3231 str. 9,
Źródło: https://datasheets.maximintegrated.com/en/ds/DS3231.pdf

93 https://github.com/robsoncouto/arduino-songs/blob/master/takeonme/takeonme.ino

94 https://www.arduino.cc/en/pmwiki.php?n=Reference/Delay

95 Kuc W., Światło regulujące zegar biologiczny, Zawód:architekt, Wydanie: Z:A_03_2010, 2010, dodatek specjalny, s. 20-21

96 Miller M. C., Seasonal affective disorder: bring on the light, Harvard Health Publishing

najkrótszych widzialnych fal świetlnych – fioletowych i niebieskich[97], kiedy to podczas wschodu słońca mają w atmosferze najdłuższą drogę do przebycia, stworzono gradient kolorów imitujących wschód słońca. Przedstawiono go na rysunku nr 54.



Rysunek 54: Gradient kolorów podczas imitacji wschodu słońca

W implementacji gradientu wykorzystano obiekt klasy `CRGBPalette16` z biblioteki **FastLED**, do przechowywania 16 kolorów w postaci RGB (3 bajty, `0xRRGGBB`). Wykorzystując dostępną metodę `DEFINE_GRADIENT_PALETTE` zdefiniowano gradient, poprzez wyspecyfikowanie 5 kolorów wraz z ich pozycjami z przedziału <0%;100%>. Proporcjonalnie do upływu czasu w trakcie symulowanego wschodu słońca, stosowny kolor pobierany jest z obiektu z wygenerowanym gradientem i wyświetlany na okręgu LED. Jasność zmienia się również proporcjonalnie do czasu, od 20% dla koloru czerwonego, aż do 100% dla koloru białego.

W celu umożliwienia przyszłej integracji, w uzupełnieniu funkcjonalności budzika świetlnego, poza specyfikacją, ale za zgodą interesariuszy, zaimplementowano przekazywanie aktualnej wartości koloru RGB do domowego serwera za pomocą protokołu MQTT. Z każdym cyklem, podczas „wschodu słońca" aktualny kolor publikowany jest w temacie **ledclock/sunrise**. Przesyłane wartości mogą posłużyć do wysterowania innego oświetlenia LED RGB, w celu wzmocnienia zarówno efektu jak i zwiększenia skuteczności rozświetlania pomieszczenia.

W ten sposób wykonano funkcjonalności [F6] i [F7]. Dla dobrego zobrazowania, na poniższym rysunku nr 55, przedstawiono 4 fazy wschodu słońca.



Rysunek 55: Fazy symulowanego wschodu słońca

97 Ackerman S., What Determines Sky's Colors At Sunrise And Sunset?, University of Wisconsin – Madison

3.2.10 Aplikacja mobilna „Zegar Led”

3.2.10.1 Budowa i działanie aplikacji

W celu zapewnienia zdalnego sterowania zegara, w ramach realizacji funkcjonalności [FZ], stworzono aplikację na telefony komórkowe pod kontrolą najpopularniejszego systemu operacyjnego - Android. Jego udział w 2019r. wśród dostarczanych na rynek globalny nowych urządzeń wyniósł 86.6%[98].

Jednym z założeń aplikacji była wysoka kompatybilność z użytkowanymi na rynku urządzeniami w ramach wybranej platformy systemowej. Przyjmując za minimalną wersję API 19 (*Kitkat*, wersja 4.4) osiągnięto zgodność z 96.5% urządzeń opartych o platformę Android[99]. Kompilacji dokonano przy wykorzystaniu aktualnej w czasie tworzenia projektu wersji API 29 (*Android Q*), a kod został napisany w języku Kotlin (oficjalnym język programowania dla platformy Android od maja 2017r.[100])

Zgodnie ze specyfikacją, głównym celem aplikacji miało być uruchamianie 5 podstawowych trybów pracy zegara. Podczas prac przygotowawczych do implementacji, zainspirowani klienci chcąc wykorzystać potencjał drzemiący w ekranie dotykowym, wyspecyfikowali dodatkowe wymaganie. Dotyczyło ono wyświetlania na tarczy z diod LED, koloru wybranego w aplikacji. Autor zgodził się uwzględnić je w specyfikacji – stało się ono szóstym trybem pracy, wyzwalanym wyłącznie z aplikacji, nazwanym „paleta barw”.

W sposób spójny w obrębie całej aplikacji, prezentacji opcji wyboru dokonano za pomocą popularnego i użytecznego wzorca projektowego – przewijanej listy elementów. Każdy z nich został wystylizowany zgodnie ze zdefiniowanym *layoutem*, składającym się z obrazu (ikony) odzwierciedlającej opcję, tytułu i krótkiego opisu. Do tego zadania wykorzystano komponent *CardView,* zawarty w androidowej bibliotece *support library*, zapewniającej kompatybilność wsteczną, aż do założonego API 19. Wygląd ekranu początkowego aplikacji oraz listę zagadek trybu edukacyjnego zaimplementowane w powyżej opisany sposób przedstawiono na rysunkach 56 i 57. Wszystkie rysunki w tym podrozdziale od nr 56 do 66 są opracowaniem własnym.

98 https://www.idc.com/promo/smartphone-market-share/os
99 https://developer.android.com/about/dashboards
100 https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/



Rysunek 56: Ekran główny aplikacji



Rysunek 57: Tryb edukacyjny

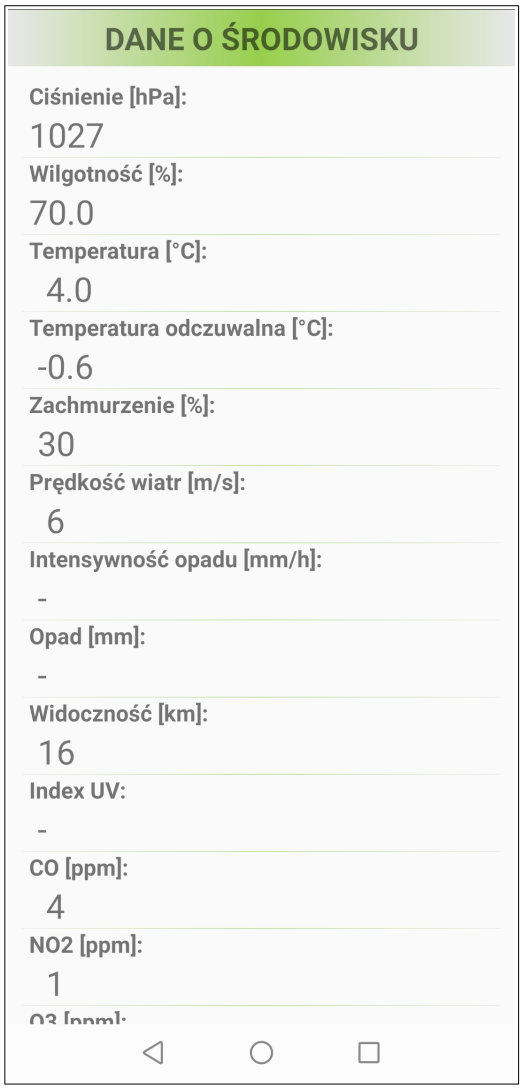
Ekran główny aplikacji „**Zegar Led**” (rys. nr 56) składa się z:

a) paska statusu do wyświetlania informacji o możliwych do podjęcia działaniach lub wychwyconych wyjątkach dotyczących operacji sieciowych,

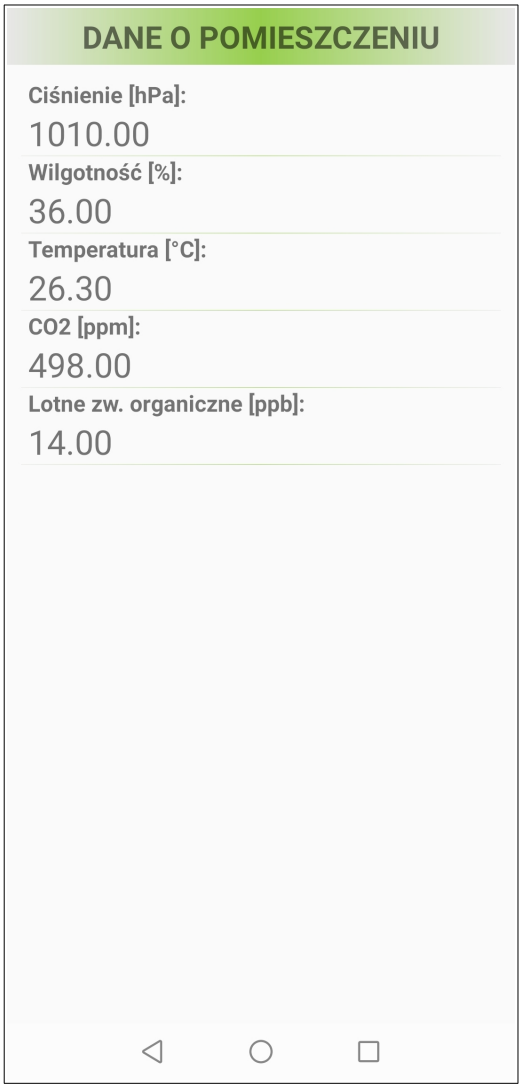
b) przewijanej listy 8 elementów, podzielonej na 6 trybów pracy zegara oraz „pozostałe opcje”: ustawienia i informacje o urządzeniu i aplikacji,

c) animowanego *loadera,* pojawiającego się w dolnej części ekranu w trakcie wykonywania operacji w tle.

Przełączenie zegara do wybranego z 6 trybów następuje po naciśnięciu żądanej pozycji na liście ekranu głównego. W przypadku opcji „Zegar LED” dochodzi jedynie do zmiany trybu, w pozostałych zaś przypadkach uruchamiane są, poza tym, odpowiednie aktywności w aplikacji. Dla opcji „Środowisko” wyświetlana jest lista wielkości fizycznych ze środowiska zewnętrznego (rys. nr 58), dla opcji „Dom” zaś wielkości zmierzone wewnątrz pomieszczenia (rys. nr 59). Aplikacja przetwarza w tych dwóch trybach pliki *json,* wysyłane z zegara jako odpowiedź na żądanie i wyświetla je w postaci list.



Rysunek 58: Dane o środowisku



Rysunek 59: Dane o pomieszczeniu

Kolejne zrzuty ekranu przedstawione na rysunkach 60 i 61 dotyczą odpowiednio opcji „Efekty świetlne” oraz „Zagadki”. Zawartość 1. listy jest generowana na podstawie pliku tekstowego zawartego w zasobach aplikacji z numerami, nazwami i opisami efektów. Druga zaś jest listą statyczną, zdefiniowaną w pliku *layoutu* aktywności *Zagadki*.



Rysunek 60: Efekty świetlne



Rysunek 61: Tryb edukacyjny

3.2.10.2 Komunikacja aplikacji z zegarem

Opierając się na dobrych praktykach Google dotyczących tworzenia aplikacji mobilnych[101], operacje sieciowe są wykonywane w tle nie blokując interfejsu użytkownika, który też nie jest z ich poziomu modyfikowany. Obsługuje to,

101 https://developer.android.com/guide/components/processes-and-threads

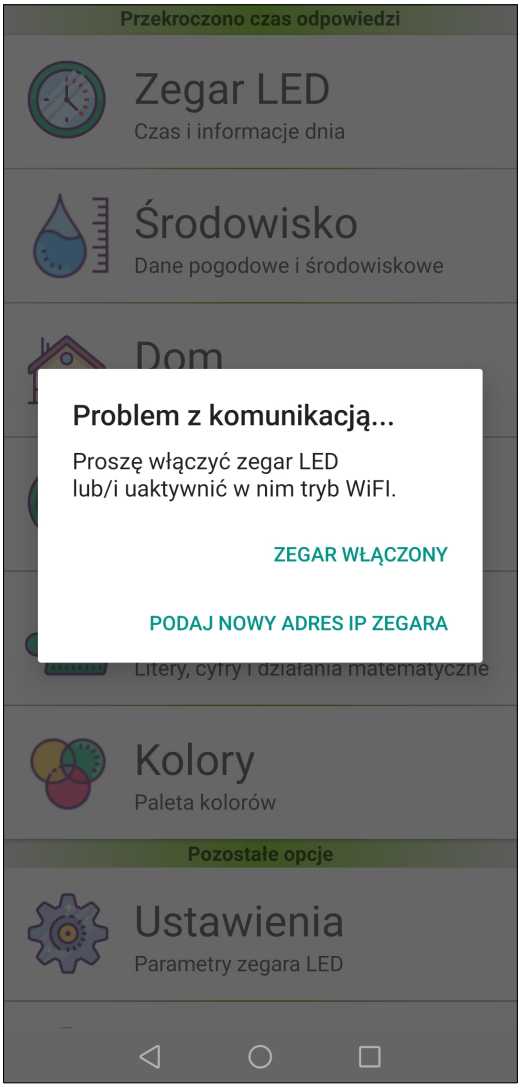
samodzielnie zaprojektowana i zaimplementowana, publiczna klasa `AsyncTaskHelper` inicjalizowana w kontekście `context` aktywności ją wywołującej. Wykorzystanie słabej referencji, umożliwia prawidłowe działanie programu, także w sytuacji kiedy aktywność wywołująca została usunięta przez *memory manager* np. w sytuacji, gdy w trakcie odbierania danych z zegara, użytkownik otworzy inną pochłaniającą zasoby aplikację. Argumentem przekazywanym do instancji klasy jest ciąg znaków rozumiany jako komenda dla zegara, przesyłany za pomocą protokołu UDP. Brak odpowiedzi od zegara w ustalonym na poziomie 6s czasie *timeout* lub inne błędy sieci komputerowej np. związane z gniazdami lub adresami, są wychwytywane i zamieniane na stosowne komunikaty. Tak zdefiniowana nieprawidłowa odpowiedź, a także jej przeciwieństwo, są zwracane z obiektu klasy `AsyncTaskHelper` poprzez implementację interfejsu `onFinished`. Dzięki temu są dostępne w aktywności wywołującej i używane w dalszym działaniu aplikacji (wyświetlenie komunikatu o błędzie lub zmiana aktywności zgodnie z zadaną komendą). Graficzny przykład błędu komunikacyjnego, występującego w trakcie działania aplikacji, został przedstawiony na rysunku nr 62. Użytkownik ma możliwość potwierdzenia, że zegar został włączony lub została w nim aktywowane połączenie z siecią WiFi poprzez opcję „zegar włączony”. Alternatywnie w pasku statusu ukazuje się edytowalne pole tekstowe (rysunek nr 63) do wpisania adresu IP zegara (wyświetlanego przez 3 sekundy na ekranie LCD, po każdym uruchomieniu urządzenia).

Komunikaty przesyłane z aplikacji do urządzenia mają następującą budowę:

```
<rozkaz>#<wartość1>(#<wartość2>)
gdzie <rozkaz> = {MODE, SETTINGS, COLOR, PING}
```

„MODE” to rozkaz do zmiany trybu zegara – w tym przypadku `<wartość1>` to liczba odpowiadająca numerowi trybu: 1-zegar, 2-środowisko, 3-pomieszczenie, 4-efekty świetlne, 5-tryb edukacyjna oraz 6-paleta barw. Opcjonalny ciąg znaków `<wartość2>` jest wykorzystywany w trybie 4 i 5, kiedy to zawarty jest w nim numer efektu świetlnego bądź rodzaju zagadki. „SETTINGS” jest rozkazem, po którym z pamięci flash zegara odczytywany jest plik konfiguracyjny i przesyłany do aplikacji w celu edycji przez użytkownika. Kolejny dopuszczalny napis „COLOR” w polu pierwszej

wartość uzupełniany jest zgodnie z formatem `#AARRGGBB` – gdzie poszczególne pary bajtów oznaczają heksadecymalne wartości komponentów koloru: przezroczystości, czerwonego, zielonego i niebieskiego. Ostatni rozpoznawany łańcuch znaków „PING” służy do weryfikacji, czy moduł WiFi ESP8266 jest *online*.



Rysunek 62: Błąd komunikacyjny



Rysunek 63: Edycja adresu IP

3.2.10.3 Tryb palety barw, preferencje użytkownika

Nowy, wymieniony powyżej, szósty tryb pracy *paleta barw*, utworzono wykorzystując ogólnodostępną bibliotekę *ColorWheel*. Stworzenie obiektu `colorWheel` i oprogramowanie jego interfejsu nasłuchującego zmiany wskazanego koloru, pozwoliło przetwarzać jego wartość. Analogicznie postąpiono w przypadku pionowego paska

`seekbar`, regulującego przezroczystość koloru w zakresie od 0 do 255. Zmiana na kole kolorów jak również na wspomnianym pasku jest momentalnie odzwierciedlana na utworzonym powyżej polu podglądu koloru i przesyłana przez sieć komputerową do zegara. W dobrych warunkach propagacji, w niezauważalnie krótkim czasie, tarcza LED zegara rozświetla się wybraną barwą. Wygląd opisywanej aktywności przedstawiono na rysunku 65. Tło zostało celowo ustawiono jako czarne w celu uzyskania kontrastu z wybieranymi kolorami.

Poprzez implementację dostępnego interfejsu `SharedPreferences`, umożliwiono zapis i odczyt „preferencji" użytkownika do pamięci telefonu. Dokonano tego w stosunku do dwóch wartości. Jeden z łańcuchów znaków, zawierał informację o adresie IP zegara, w celu umożliwienia próby automatycznego nawiązania połączenia po uruchomieniu aplikacji. Drugi łańcuch znaków zawierał ostatnio wybrany kolor z palety barw w formacie `#AARRGGBB`, który jest automatycznie przesyłany do zegara po wybraniu trybu o numerze 6.

3.2.10.4 Ustawienia – edycja parametrów konfiguracyjnych

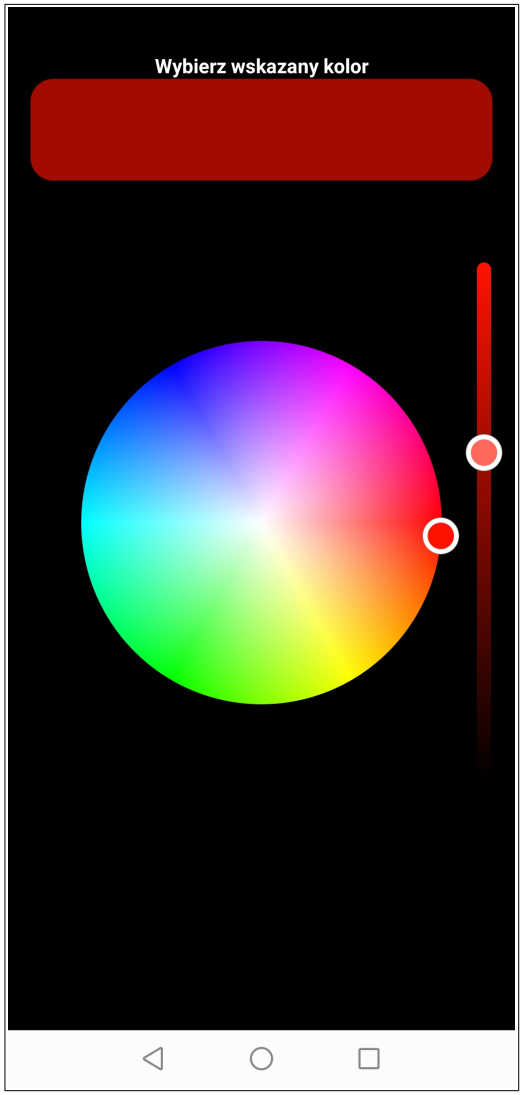
Drugim co do ważności zadaniem aplikacji, zaraz po zdalnym sterowaniu, jest umożliwienie użytkownikowi edycji parametrów konfiguracyjnych zegara w przystępny sposób. Po wybraniu opcji „Ustawienia", użytkownik uzyskuje listę nazw parametrów i ich aktualnych wartości, wraz z możliwością ich edycji. Przedstawiono to na rysunku nr 64. Lista ta tworzona jest dynamicznie na podstawie zawartości pliku JSON, który jest przesyłany z zegara po każdorazowym wybraniu opcji „Ustawienia". Taki zabieg daje możliwość uniwersalnego sposób wprowadzania zmian w konfiguracji, nawet w przypadku zmiany oprogramowania zegara, bez potrzeby ponownej kompilacji aplikacji mobilnej. W trakcie parsowania pliku, zawartość klucza jest przetwarzana zgodnie ze wzorcem:

`<nazwa klucza> ! <maksymalna długość> <rodzaj>`

Natomiast odpowiadająca mu wartość wpisywana jest do edytowalnego pola typu `EditText`. Przykładowy klucz „`lambda!010f`" (por. rys. nr 36) oznaczający długość geograficzną jest parsowany zgodnie z regułami pokazanymi w tabeli nr 28.



Rysunek 64: Ustawienia



Rysunek 65: Paleta barw

Tabela 28 - Budowa kluczy z pliku config.json - parametry konfiguracyjne zegara

fragment	wyjaśnienie
lambda	identyfikator nazwy pola [ciąg znaków do separatora]
!	separator
010	maksymalna długość wartości (w systemie dziesiętnym) [trzy kolejne cyfry po separatorze]
f	rodzaj wartości ze zbioru {f, s, d, y, h} [kolejny, 4 znak po separatorze] gdzie: f-liczna zmiennoprzecinkowa float, s – łańcuch znakowy, d – liczba całkowita, y – prawda/fałsz, h – godzina

Dzięki dostępności informacji dotyczących ograniczeń, realizacja algorytmu aplikacji jest w stanie dokonać podstawowej walidacji wprowadzanych przez użytkownika danych. Przedstawiona koncepcja nie została wstępnie zaplanowana w opisany powyżej sposób. Pierwotnie zamierzano wykorzystać tablice notacji JSON i obok nazwy parametru i jego wartości przesyłać pary klucz-wartość definiujące ograniczenia. Koncepcja ta została jednak zarzucona, aby lepiej gospodarować pamięcią operacyjną NodeMCU i przede wszystkim w celu wykonalności zadania. Próby wykazały, że budowa łańcucha znaków powyżej 900 bajtów kończyła się resetem mikrokontrolera z powodu niemożliwości zaalokowania kolejnych, wymaganych zasobów pamięci.

Ostatnią pozycję na liście ekranu głównego stanowią „Informacje", gdzie poza ikoną, nazwą i wersją aplikacji wymieniono jej funkcje. Zgodnie z licencjami, umieszczono tam też linki do strony źródłowej[102] wykorzystanych grafik oraz biblioteki *ColorWheel*[103] obsługującej paletę barw.



Rysunek 66: Informacje o aplikacji

102 https://icons8.com
103 https://github.com/AntonPopoff/android-color-wheel

3.3 Przeprowadzone testy oprogramowania

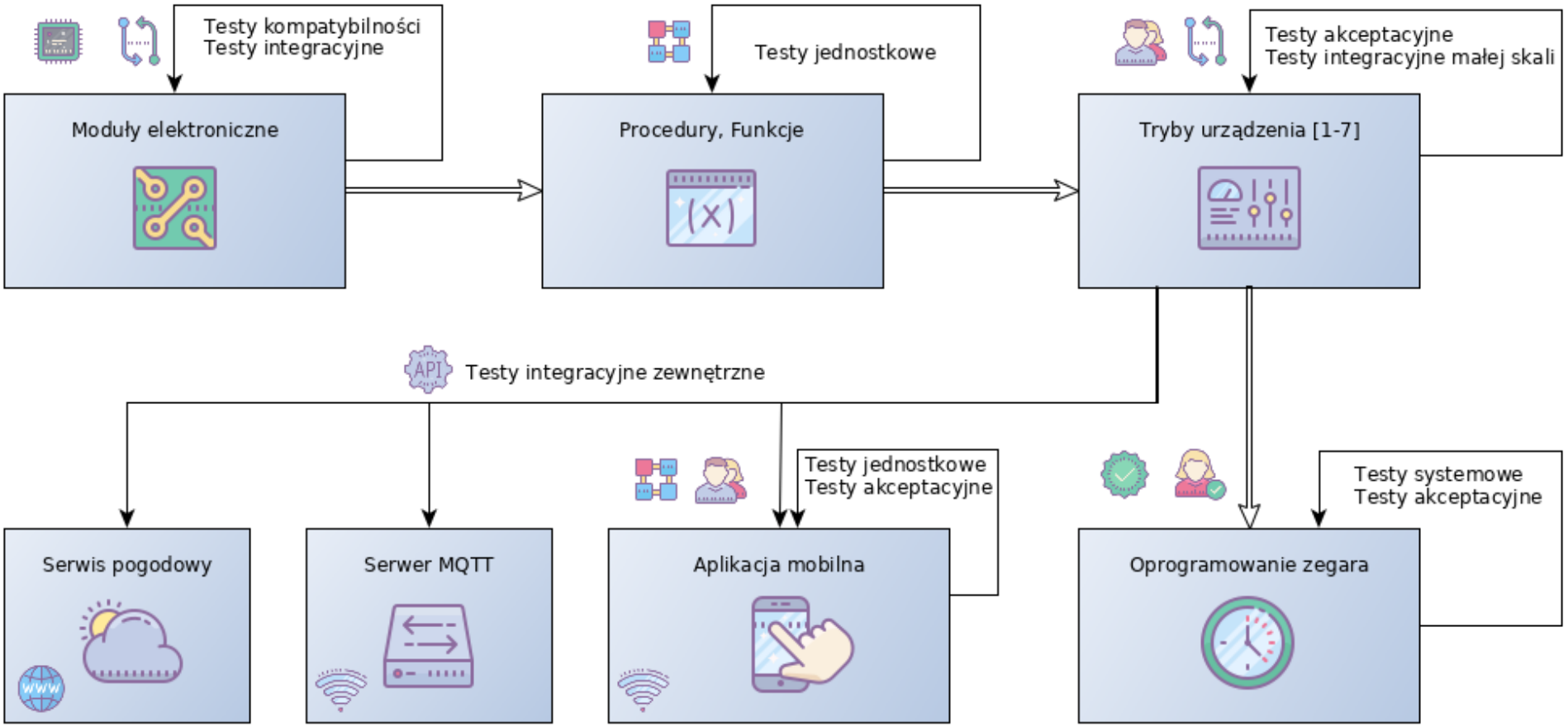
Już w fazie projektowana urządzenia, stworzono wstępny plan dotyczący poziomów przeprowadzanych testów funkcjonalnych i niefunkcjonalnych w odniesieniu do produktu w poszczególnych fazach powstawania. Został on przedstawiony na rysunku nr 67. Równolegle do implementacji tworzono scenariusze testowe do testów akceptacyjnych, opartych o przypadki użycia.

Na wstępnym etapie realizacji projektu, kiedy powstawał on na płytce stykowej, ostatecznie zweryfikowano kompatybilność poszczególnych modułów z platformą NodeMCU w oparciu o biblioteki programistyczne służące do ich obsługi. Przejście do następnego etapu nastąpiło po pomyślnym sprawdzeniu urządzeń wejścia/wyjścia i ich stabilnym działaniu.

Ze względu na dobrą praktykę w postaci wczesnego wyeliminowania usterek, a także na modułowy charakter oprogramowania (podział pracy urządzenia na odrębne tryby), testowanie działającego kodu dokonywano już w trakcie implementacji. Były to zautomatyzowane testy jednostkowe dotyczące dla przykładu klasy `EnvVal` (wielkość fizyczna, rysunek nr 4), metod wspomagających obsługę połączenia sieciowego czy szereg funkcji pomocniczych do formatowania daty i czasu.

Zgodnie z przyjętym procesem wytwarzania oprogramowania w postaci metodyki *Feature Driven Development,* po implementacji każdy z trybów był przedstawiony zamawiającym na prototypie urządzenia. Wykorzystując utworzoną na początku projektu specyfikację dokonywano testów akceptacyjnych. Uwagi użytkowników w postaci niezgodności ze specyfikacją lub po prostu błędów zapisywane były w arkuszu kalkulacyjnym. Po ich podjęciu i naprawie w procesie debugowania, zamykano zgłoszenia, dodając krótki komentarz o podjętych działaniach.

Z racji, że większość danych wyjściowych wyświetlanych było na ekranach urządzenia i na okręgu diod LED, ostatecznie ocena prawidłowości testów wymagała użycia zmysłu wzroku. Jednakże, w oparciu o dostępny interfejs sieciowy urządzenia w postaci otwartego portu serwera nasłuchującego datagramów UDP, stworzono dedykowane komendy realizujące proste scenariusze testowe. Pozwoliło to na skrócenie czasu testowania i umożliwiło sprawniejszą realizację kolejnych poleceń z procedur testowych. Do stworzenia opisanej automatyzacji wykorzystano środowisko



Rysunek 67: Poziomy testów w podziale na moduły tworzące lub współpracujące z urządzeniem

Źródło: opracowanie własne

Grafika: https://icons8.com

uruchomieniowe Node.js[104] zainstalowane na serwerze w sieci lokalnej, ze skryptami napisanymi w języku JavaScript.

Po pomyślnej integracji z serwisami dostarczającymi dane środowiskowe, przystąpiono do weryfikacji działania aplikacji mobilnej. Poza fizycznie dostępnym telefonem komórkowym z przekątną ekranu 5” (Android 9) oraz tabletem o przekątnej dwukrotnie większej (Android 4.4), wygląd i działanie aplikacji zostały sprawdzone na emulatorach dostępnych w środowisku deweloperskim Android Studio. Utworzone tam wirtualne urządzenia (*virtual devices*) obejmowały pośrednie wersje tego systemu operacyjnego oraz różne wielkości ekranów. Poza tym aplikację sprawdzono, pod kątem użyteczności. Szybka nauka jej obsługi dzięki intuicyjności oraz łatwość użytkowania, zostały potwierdzone nawet przez dzieci w wieku 4 oraz 6 lat.

Natomiast wstęp do prób obciążeniowych nastąpił niespodziewanie właśnie wtedy, kiedy najmłodszy użytkownik w trybie palety barw fantazyjnymi i szybkimi ruchami, rozpoczął zalewanie zegara pakietami UDP. Powstałe opóźnienia w wyświetlaniu wybranych barw, potwierdzone później w próbach automatycznych, były na poziomie poniżej 1.5s w ekstremalnych przypadkach i uznane zostały za dopuszczalne.

Ostatecznie przeprowadzono testy systemowe na urządzeniu w obudowie. Potwierdziły one wysokopoziomową funkcjonalność stworzonego systemu[105] pracującego na zbudowanym urządzeniu. W szerszym gronie, pomyślnie przeprowadzono wcześniej stworzone scenariusze testowe. W rezultacie, produkt w istniejącej wersji został zaakceptowany przez zamawiających.

Testowanie i uwzględnianie jego czasem niepomyślnych wyników zajęły zdecydowanie więcej czasu niż samo kodowanie. Jednakże wpłynęło ono pozytywnie na jakość oprogramowania zbudowanego urządzenia. Czas poświęcony na analizę kodu źródłowego, wielokrotnie zaowocował jego poprawie, często na tyle, że zapobiegało to przyszłym potencjalnym problemom. Poza oczywistym celem jakim było znajdowanie usterek, testowanie pozwoliło nabrać zaufania do jakości wytworzonego kodu.

104 https://nodejs.org/en/

105 Roman A., Testowanie i jakość oprogramowania, PWN, Warszawa 2015, str. 157-158

Zakończenie

Postępując w myśl zasad metodyki **Feature-Driven Development** zebrano i uzgodniono wymagania projektu zarówno w warstwie sprzętowej i oprogramowania. Na ich podstawie stworzono specyfikację wymagań, dzieląc ją na odrębne cechy. Etap modelowania każdej z nich był bardzo angażujący i wyniku długotrwałego, intensywnego procesu, trzykrotnie nastąpił powrót do specyfikacji wymagań w celu udoskonalenia i uczynienia projektu bardziej użytecznym. Posługując się funkcjonalnym podziałem, w dwutygodniowych sprintach dostarczano zamawiającym do testowania kolejne tryby pracy budowanego zegara. Średnio, w czasie jednego tygodnia, uwzględniano uwagi i błędy zgłoszone w testach akceptacyjnych każdej oddanej funkcjonalności. Dwukrotnie wykorzystano elastyczność wybranej zwinnej metodyki FDD, kiedy to znacząco zmienione zostały wymagania (np. tryb palety barw) i powrócono do specyfikacji, a następnie zaktualizowano zakres prac programistycznych. Ostatecznie wykorzystanie wspomnianej metodyki zarządzania projektami informatycznymi okazało się bardzo korzystną decyzją dla realizowanego przedsięwzięcia z punktu widzenia ogólnego, zamawiających, a także wykonawcy.

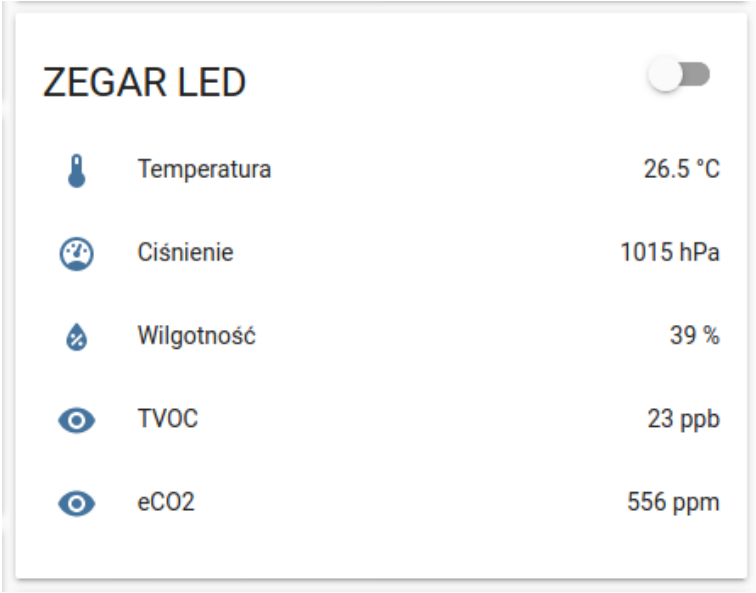
Kod źródłowy w trakcie tworzenia oprogramowania zegara był archiwizowany przy użyciu linuksowego programu *rdiff-backup*. Poza tym regularnie tworzono jego kopie bezpieczeństwa. W tym projekcie okazało się to skuteczne i wystarczające, jednak lepszym rozwiązaniem wg autora pracy byłoby wykorzystanie systemu kontroli wersji.

Na płaszczyźnie sprzętowej, po testach kompatybilności modułów elektronicznych, projekt z płytek stykowych względnie szybko został przeniesiony na pierwszą wersję płytki PCB. Stworzyło to stabilnie działającą platformę sprzętową, z modułami podłączonymi przez połączenia rozłączalne, na której prezentowane było oprogramowanie rozbudowywane o kolejne funkcjonalności. Przytoczone podejście w postaci budowy udoskonalanych prototypów sprzętowych, mimo dodatkowych nakładów finansowych, opłaciło się, eliminując problemy natury teorii obwodów i tworząc solidną podstawę do rozwoju oprogramowania.

Napisanie dedykowanej aplikacji mobilnej wykreowało wygodny interfejs urządzenia, umożliwiając jego **zdalne sterowanie** (z założenia w zasięgu sieci WiFi) oraz wygodne wprowadzanie parametrów konfiguracyjnych. Dodatkowo, wykorzystując możliwości ekranu dotykowego telefonu umożliwiono wybór kolorów

w trybie palety barw (zaimplementowane wyłącznie w aplikacji). W ostatecznym rozrachunku sterowanie zegarem możliwe jest w sposób trzytorowy. Poza aplikacją pracującą w systemie Android oprogramowano fizyczny przycisk funkcyjny na obudowie, a także czujnik gestów dłoni. Tak zwielokrotniona możliwość wydawania rozkazów z pewnością wpłynie na wygodę użytkowania oraz będzie stanowić dobrą alternatywę podczas ewentualnej awarii w jednym z kanałów sterowania.

Dane z lokalnych pomiarów temperatury, ciśnienia, wilgotności, natężenia oświetlenia, a także stężenia lotnych związków organicznych zostały wykorzystane nie tylko do celów informacyjnych na wyświetlaczach urządzenia. Używając lekkiego protokołu transportowego, raportowane są one regularnie do serwera **MQTT** i wykorzystane w systemie automatyzacji domowej **Home Assistant**. Wyniki pomiarów dostępne są w systemie w zakładce informacyjnej pomieszczenia, co zostało zobrazowane na poniższym rysunku.



Rysunek 68: Dane z zegara w systemie Home Assistant

Przede wszystkim jednak, uzyskane wartości wykorzystywane są przez skrypty automatyzujące. Temperatura, jako jedna ze składowych, posłużyła do wysterowania termostatu grzejnika w pokoju dziecięcym w okresie jesienno-zimowym. Przekroczenie dopuszczalnych wielkości stężenia lotnych związków organicznych lub wilgotności generuje okresowy alarm dźwiękowy w głośnikach sieciowych i jest poprzedzone

wysłaniem wiadomości e-mail z ostrzeżeniem. Zbyt duże natężenie światła w pomieszczeniu w przedziale od godziny 22 do najbliższego świtu nautycznego, skutkuje wysłaniem powiadomienia *push* na telefony z aplikacją systemu zarządzania domem.

Poza powyższym, została otwarta furtka do integracji z budzikiem świetlnym. Aktualne kolor i jasność podczas porannego rozświetlenia są przekazywane do brokera MQTT i mogą zostać użyte do wzmocnienia efektu symulowanego wschodu słońca np. przez powielenie na zewnętrznych taśmach LED RGB.

Przedstawione w pracy zaplanowane i skoordynowane działania, doprowadziły do stworzenia **wielofunkcyjnego zegara** elektronicznego, naturalnie wpisującego się w kategorię internetu rzeczy **IoT**. Finalnie osiągnięto więc **cel główny** projektu, którego funkcje mają charakter:

1) informacyjny – tryb zegara, tryb stacji pogodowej oraz tryb budzików,
2) edukacyjny – tryb zagadek językowych i matematycznych,
3) rozrywkowy – tryb efektów świetlnych oraz tryb palety barw,
4) pomiarowy – tryb monitoringu parametrów pomieszczenia.

Dodatkowo, osiągnięto także niepisany cel poboczny projektu. Podczas jego tworzenia, tak samo jak i niniejszej pracy, nie użyto ani jednego oprogramowania znanej firmy z Redmond (Waszyngton, USA). Postawiono za to na solidne umocowane **wolne oprogramowanie**[106], w kilku przypadkach zaś na dostępne do użytku bezpłatnego, które zostało wyszczególnione w tabeli nr 29.

Zaangażowanie oraz silna motywacja autora pracy doprowadziła w rezultacie do powstania estetycznego produktu o wielu walorach użytkowych. Przejście przez cały proces od pomysłu przez zbieranie wymagań, specyfikację projektu, projektowanie, aż po realizację, zapisało się jako bardzo owocne i dające satysfakcję doświadczenie. Wzbudzono innowacyjność i kreatywność potrzebną architektowi, poszerzono wiedzę, która potrzebna jest każdemu inżynierowi, wzmocniono cierpliwość i konsekwencję tak cenne u testerów, a także dobrą organizację i sumienność wskazane u programistów. Na przestrzeni 7 miesięcy osiągnięto coś więcej, niż tylko realizacja celu projektowego, ale przecież... Czas oraz jego upływ są względne.

106 https://www.gnu.org/philosophy/free-sw.pl.html

Spis wykorzystanej literatury

1. Ackerman S., *What Determines Sky's Colors At Sunrise And Sunset?*, University of Wisconsin – Madison, odczyt z dnia 24.02.2019, https://www.sciencedaily.com/releases/2007/11/071108135522.htm
2. AI-Thinker team, *ESP-12E WiFi Module*, Version1.0, https://components101.com/sites/default/files/component_datasheet/ESP12E%20Datasheet.pdf, odczyt z dnia 31.10.2019
3. Barr M., Massa A., *Programming Embedded Systems, Second Edition with C and GNU Development Tools*, O'Reilly Media, Sebastopol, 2006
4. Bassett L., *Introduction to JavaScript Object Notation*, O'Reilly Media, Sebastopol, 2015
5. Blanchon B., *Mastering ArduinoJson - Efficient JSON serialization for embedded C++*, https://arduinojson.org/book/, odczyt z dnia 21.02.2019
6. ECMA Technical Committee 39, *The JSON Data Interchange Syntax, Standatd ECMA-404,* Ecma International 2017
7. Espressif IOT Team, *ESP8266 Technical Reference version 1.2*, www.espressif.com, 2016r.
8. Espressif Systems, *ESPRESSIF SMART CONNECTIVITY PLATFORM: ESP8266*, https://espressif.com, 2013r., odczyt z dnia 03.11.2019
9. Falck D., Collette B., *FreeCAD [How-to],* Packt Publishing, Birmingam, 2012
10. Goyal Sadhna, *Major Seminar On Feature Driven Development*, Technical University Munich, WS 2007/08
11. Halliday D. [i in.], *Podstawy Fizyki*, Wydawnictwo Naukowe PWN, Warszawa 2003, Tom 4
12. Hillar G., *MQTT Essentials - A Lightweight IoT Protocol*, Packt Publishing, Birmingam, 2017
13. Jayakumar M., *The inernet of things with esp8266 Hands on approach*, CreateSpace Independent Publishing Platform, 2017r.
14. Karta katalogowa equalizera MSGEQ7*, Seven Band Graphic Equalizer Data Sheet, 9/2004,* http://mix-sig.com
15. Karta katalogowa złącz JST, *Katalog JST Catalog vol. 120e,* https://www.jst-mfg.com

16. Kolban N., *Kolban's Book on the ESP32 & ESP8266*, http://neilkolban.com/tech/esp8266/, odczyt z dnia 11.02.2019
17. Kuc W., *Światło regulujące zegar biologiczny*, w: Zawód:architekt, Wydanie: Z:A_03_2010, 2010, dodatek specjalny, s. 20-21
18. Margolis M., *Arduino Cookbook*, O'Reilly Media, Sebastopol, 2017
19. Mazurkiewicz A., *Analiza jakości wydruku elementu z termoplastu ABS wykonanego w technologii FDM,* w: „Autobusy : technika, eksploatacja, systemy transportowe", 2017r., R. 18, nr 6, str. 956-960
20. Mielczarek W., *Szeregowe interfejsy cyfrowe*, Wydawnictwo Helion, Gliwice, 1993r.
21. Miller M. C., *Seasonal affective disorder: bring on the light*, Harvard Health Publishing, https://www.health.harvard.edu/blog/seasonal-affective-disorder-bring-on-the-light-201212215663 , odczyt z dnia 24.02.2019
22. Monk S., *Electronics Cookbook. Practical Electronic Recipes with Arduino and Raspberry Pi*, O'Reilly Media, Sebastopol, 2017
23. Oficjalna dokumentacja oprogramowania układowego urządzenia NodeMCU, https://nodemcu.readthedocs.io, odczyt z dnia 13.11.2019r.
24. Oficjalna strona internetowa Centrum Informatycznego Trójmiejskiej Akademickiej Sieci Komputerowej, https://task.gda.pl, odczyt z dnia 10.11.2019
25. Oficjalna strona internetowa Dark Sky Company, dokumentacja API dla deweloperów, https://darksky.net/dev/docs, odczyt z dnia 24.02.2019
26. Oficjalna strona internetowa firmy Nebulon (w której pracował autor metodyki FDD) , http://www.nebulon.com/fdd/, odczyt z dnia 24.02.2019
27. Oficjalna strona internetowa i repozytorium kart katalogowych układów zintegrowanych firmy Maxim-Integrated, https://www.maximintegrated.com/
28. Oficjalna strona internetowa Instytut Inżynierów Elektryków i Elektroników IEEE, Standardy budowy bezprzewodowych sieci komputerowych IEEE 802.11 „Wireless Local Area Networks", www.ieee802.org/11/, odczyt z dnia 31.10.19
29. Oficjalna strona internetowa International Data Corporation https://www.idc.com/promo/smartphone-market-share/os, odczyt z dnia 22.01.2020r.

30. Oficjalna strona internetowa Internet Engineering Task Force (IETF), Network Time Protocol Version 4: Protocol and Algorithms Specification, https://tools.ietf.org/html/rfc5905, odczyt z dnia 14.11.2019r.
31. Oficjalna strona internetowa języka programowania Lua, https://www.lua.org/, odczyt z dnia 31.10.2019
32. Oficjalna strona internetowa mobilnego systemu operacyjnego Android dedykowana deweloperom,https://developer.android.com, odczyt z dnia 31.10.2019
33. Oficjalna strona internetowa Network Time Protocol, http://ntp.org/, odczyt z dnia 12.11.2019r.
34. Oficjalna strona internetowa firmy New Line Technologies, FDD, odczyt z dnia 24.02.2019r., https://newline.tech/blog/feature-driven-development-methodology/,
35. Oficjalna strona internetowa projektu Arduino, https://www.arduino.cc, odczyt z dnia 31.10.2019
36. Oficjalna strona internetowa projektu Espruino, https://www.espruino.com/, odczyt z dnia 03.11.2019
37. Oficjalna strona internetowa społeczności inżynierów i twórców pasjonujących się elektroniką „Circut Digest", Top Hardware Platforms for Internet of Things (IoT), https://circuitdigest.com/article/top-hardware-platforms-for-internet-of-things-iot, odczyt z dnia 31.10.2019
38. Oficjalna strona internetowa World Air Quality Index, dokumentacja API dla deweloperów, https://aqicn.org/json-api/doc/ , odczyt z dnia 24.02.2019
39. Oficjalna strona internetowa Zakładu Inteligentnych Systemu Wspomagania Decyzji Politechniki Poznańskiej, Optymalizacja Wielokryterialna, http://www.cs.put.poznan.pl/mmika/Optymalizacja%20wielokryterialna.pdf
40. Oficjalna strona internetowa zintegrowanego środowiska deweloperskiego Arduino, https://www.arduino.cc, 14.11.2019r.
41. Oficjalne repozytorium internetowe github projektu NodeMCU, https://github.com/nodemcu/nodemcu-firmware, odczyt z dnia 03.11.2019
42. Oficjalne repozytorium projektu Arduino, https://github.com/arduino/Arduino/blob/master/build/shared/manpage.adoc, 14.11.2019r.
43. Pęczak A., *NTP i synchronizacja czasu*, w: IT w Administracji, Nr 10/2018r.
44. Prata S., *Szkoła programowania. Język C. wyd 5*, Helion, Gliwice, 2006

45. Pulver T., *Hands-On Internet of Things with MQTT*, Packt Publishing, Birmingam, 2019
46. Purdum J., *Beginning C for Arduino*, Apress Media, New York, 2012
47. Roman A., *Testowanie i jakość oprogramowania*, PWN, Warszawa 2015
48. Rybus R., *Feature Driven Development – lekka metodyka tworzenia oprogramowania*,https://www.e-informatyka.pl/index.php/pimio/metodyki-lekkie/feature-driven-development-lekka-metodyka-tworzenia-oprogramowania/, odczyt z dnia 08.06.2019
49. Schwartz M., *ESP8266 Internet of Things Cookbook*, Packt Publishing, Birmingham, 2017
50. Smith B., *Beginning JSON*, Apress Media, New York, 2015
51. Sommerville I., *Software Engineering, Ninth Edition*, Addison Wesley, Boston 2011
52. Techniczna strona internetowa dotycząca systemów wbudowanych, http://www.mbeddedc.com/, odczyt z dnia 08.06.2019
53. UpSkill Learning, *ESP8266: Programming NodeMCU Using Arduino IDE*, CreateSpace Independent Publishing Platform, 2016r.
54. Wheat D., *Arduino Internals*, Apress Media, New York, 2011
55. Wrycza S., Marcinkowski B., Wyrzykowski K., *Język UML 2.0 w modelowaniu systemów informatycznych*, Wydawnictwo Helion, Gliwice, 2005
56. Yorik van Havre, FreeCAD Community, *FREECAD A MANUAL,* https://legacy.gitbook.com/book/yorikvanhavre/a-freecad-manual/
57. Young R., *The Requirements Engineering Handbook*, Artech House, Boston, 2004

Spis tabel

Tabela 1 - Lista funkcjonalności urządzenia po negocjacjach.....11
Tabela 2 - Lista cech urządzenia po negocjacjach.....12
Tabela 3 - Wagi dla kryteriów przy wyborze platformy sprzętowej.....23
Tabela 4 - Zestawienie peryferyjnych modułów elektronicznych dla NodeMCU.....33
Tabela 5 - Podsumowanie informacji o wyświetlaczu OLED SSD1306.....34
Tabela 6 - Podsumowanie informacji o wyświetlaczu LCD 4x20.....34
Tabela 7 - Podsumowanie informacji o mikrofonie Adafruit MAX9814.....35
Tabela 8 - Podsumowanie informacji o korektorze graficznym MSGEQ7.....35
Tabela 9 - Podsumowanie informacji o czujniku BH1750.....36
Tabela 10 - Podsumowanie informacji o pierścieniu LED RGB (WS2812B).....36
Tabela 11 - Podsumowanie informacji o czujniku PIR HC-SR501.....37
Tabela 12 - Podsumowanie informacji o czujniku BMP280.....37
Tabela 13 - Podsumowanie informacji o module PAJ7620U2.....38
Tabela 14 - Podsumowanie informacji o głośniku Grove Speaker.....38
Tabela 15 - Podsumowanie informacji o module przycisku.....39
Tabela 16 - Podsumowanie informacji o zegarze czasu rzeczywistego RTC DS3231....39
Tabela 17 - Podsumowanie informacji o czujniku CCS811.....40
Tabela 18 - Zestawienie elementów do montażu na płytce głównej zegara.....55
Tabela 19 - Parametry zasilacza zegara.....56
Tabela 20 - Urządzenia i strumienie wejścia.....69
Tabela 21 - Urządzenia i strumienie wyjścia.....70
Tabela 22 - Komendy sterujące w zależności od kanału sterowania.....71
Tabela 23 - Stan urządzeń wyjścia w zależności od aktualnego trybu pracy.....72
Tabela 24 - Porównanie usług serwisów pogodowych.....78
Tabela 25 - Zestawienie danych publikowanych do brokera MQTT.....82
Tabela 26 - Rodzaje zagadek w trybie edukacyjnym.....86
Tabela 27 - Przykładowe zagadki w trybie edykacyjnym.....87
Tabela 28 - Budowa kluczy z pliku config.json - parametry konfiguracyjne zegara.....97
Tabela 29 - Spis oprogramowania wykorzystanego w projekcie i niniejszej pracy.....113
Tabela 30 - Spis bibliotek programistycznych użytych w projekcie.....114

Spis rysunków

Rysunek 1: 5 procesów FDD wraz z ich produktami....9
Rysunek 2: Schemat blokowy projektowanego urządzenia....17
Rysunek 3: Diagram stanów - modelowanie historii życia programu....19
Rysunek 4: Struktura klasy *Wielkość fizyczna*....20
Rysunek 5: Algorytm uruchamiania aplikacji mobilnej i kontroli połączenia....21
Rysunek 6: Platforma sprzętowa NodeMCU z ESP-12-E....25
Rysunek 7: Porty ESP-12-E zmapowane na płytce NodeMCU....26
Rysunek 8: Schemat budowy pinu GPIO....28
Rysunek 9: Schemat budowy magistrali I2C z nadajnikami i odbiornikami....30
Rysunek 10: Wyświetlacz SSD1306....34
Rysunek 11: Wyświetlacz LCD 4x20....34
Rysunek 12: Adafruit MAX9814....35
Rysunek 13: Equalizer MSGEQ7....35
Rysunek 14: Czujnik BH1750....36
Rysunek 15: Pierścień LED RGB....36
Rysunek 16: Czujnik ruchu PIR....37
Rysunek 17: Czujnik BMP280....37
Rysunek 18: Moduł PAJ7620U2....38
Rysunek 19: Głośnik z interfejsem cyfrowym Grove Speaker....38
Rysunek 20: Moduł przycisku....39
Rysunek 21: Moduł czasu RTC DS3231....39
Rysunek 22: Moduł czujnika czystości powietrza CCS811....40
Rysunek 23: Interfejs graficzny Arduino IDE....42
Rysunek 24: 32 bitowy format czasu NTP (skrócony znacznik czasu)....44
Rysunek 25: 64 bitowy format czasu NTP (znacznik czasu)....44
Rysunek 26: Format nagłówka pakietu NTP....45
Rysunek 27: Budowa nagłówka datagramu UDP....46
Rysunek 28: Role w protokole MQTT oraz wyjaśnienie wzorca *Publish-Subscribe*....48
Rysunek 29: Zagnieżdżony tekst w formacie JSON wraz z obiektową reprezentacją...51
Rysunek 30: Sygnały w złączu JST....53
Rysunek 31: Schemat elektryczny płytki głównej zegara....54

Rysunek 32: Projekt płytki głównej zegara – wersja finalna.....58
Rysunek 33: Projekt tabliczki znamionowej urządzenia.....60
Rysunek 34: Model frontowej części obudowy.....60
Rysunek 35: Widok izometryczny modelu zegara - front.....61
Rysunek 36: Widok izometryczny modelu zegara - tył.....61
Rysunek 37: Widok modelu zegara z góry.....62
Rysunek 38: Przekrój przez środkową płaszczyznę poziomą modelu zegara.....62
Rysunek 39: Widok tylnej pokrywki modelu zegara.....62
Rysunek 40: Rozmieszczenie czujników i innych komponentów wewnątrz obudowy. 63
Rysunek 41: Algorytm inicjalizacji zegara LED.....65
Rysunek 42: Strona edycji danych dostępowych w trybie konfiguracyjnym.....67
Rysunek 43: Ekran LCD w trybie konfiguracyjnym.....67
Rysunek 44: Domyślna zawartość pliku konfiguracyjnego config.json.....69
Rysunek 45: Zawartość ekranu LCD w trybie zegara.....75
Rysunek 46: Widok urządzenia w trybie zegara.....77
Rysunek 47: Struktura danych EnvData.....79
Rysunek 48: Prezentacja postępu operacji sieciowych w tle.....79
Rysunek 49: Przykładowe skany obrazów pogody.....81
Rysunek 50: Widok urządzenia w trybie stacji pogodowej.....81
Rysunek 51: Widok ekranu LCD w trybie monitoringu pomieszczenia, widok 1/2.....83
Rysunek 52: Schemat czasowy equalizera MSGEQ7.....84
Rysunek 53: Widok urządzenia w trybie efektów świetlnych.....85
Rysunek 54: Gradient kolorów podczas imitacji wschodu słońca.....89
Rysunek 55: Fazy symulowanego wschodu słońca.....89
Rysunek 56: Ekran główny aplikacji.....91
Rysunek 57: Tryb edukacyjny.....91
Rysunek 58: Dane o środowisku.....92
Rysunek 59: Dane o pomieszczeniu.....92
Rysunek 60: Efekty świetlne.....93
Rysunek 61: Tryb edukacyjny.....93
Rysunek 62: Błąd komunikacyjny.....95
Rysunek 63: Edycja adresu IP.....95

Rysunek 64: Ustawienia.....97
Rysunek 65: Paleta barw.....97
Rysunek 66: Informacje o aplikacji.....98
Rysunek 67: Poziomy testów w podziale na moduły tworzące lub współpracujące z urządzeniem.....100
Rysunek 68: Dane z zegara w systemie Home Assistant.....103

Spis oprogramowania wykorzystanego w projekcie i niniejszej pracy

Tabela 29 - Spis oprogramowania wykorzystanego w projekcie i niniejszej pracy

LP.	Nazwa	Wersja	Rodzaj	Licencja
1	Linux Mint	19.2	System operacyjny	GPL
2	LibreOffice Writer	6.4.0.3	Edytor tekstu	MPLv2.0
3	LibreOffice Calc	6.0.7	Arkusz kalkulacyjny	MPLv2.0
4	yEd Graph Edytor	3.19	Edytor graficzny diagramów	Freeware
5	Arduino IDE	1.8.7	Zintegrowane środowisko deweloperskie	LGPL
6	FreeCAD	0.18	Modelowanie CAD 3D	LGPLv2+
7	EasyEDA	6.3.22	Projektowanie płytek PCB	Freeware
8	Android Studio	3.5.3	Zintegrowane środowisko deweloperskie	Freeware
9	GIMP	2.8.22	Edytor obrazów	GPL v3.0+
10	bdfconv	2.27.6	Konwerter czcionek (wiersz poleceń)	BSD 2 klauz.
11	rdiff-backup	1.3.3	Archiwizacja (wiersz poleceń)	GPL-2.0

Spis bibliotek programistycznych użytych w projekcie

Tabela 30 - Spis bibliotek programistycznych użytych w projekcie

LP.	Nazwa	Opis	Lic.	Link
1	U8g2.h	Wyświetlacz graficzny OLED 1.3"	BSD 2 klauz.	https://github.com/olikraus/u8g2
2	LiquidCrystal_I2C.h	Wyświetlacz LCD 20x4	LGPL v3.0	https://github.com/johnrickman/LiquidCrystal_I2C
3	BH1750.h	Czujnik natężenia światła BH1750	MIT	https://github.com/PeterEmbedded/BH1750FVI
4	FastLED.h	Pierścień LED RGB 60	MIT	https://github.com/FastLED/FastLED
5	WS2812FX.h	Pierścień LED RGB 60	MIT	https://github.com/kitesurfer1404/WS2812FX
6	BME280I2C.h	Czujnik temperatury, wilg. i ciśnienia BMP280	GPL v3.0	https://github.com/finitespace/BME280
7	paj7620.h	Moduł wykrywania gestów PAJ7620U2	MIT	https://github.com/Seeed-Studio/Gesture_PAJ7620
8	RtcDS3231.h	Moduł czasu RTC - DS3231	LGPL v3.0	https://github.com/Makuna/Rtc
9	takeonme.ino	Melodie do odtwarzania przez Arduino	MIT	https://github.com/robsoncouto/arduino-songs
10	ColorWheel	Paleta barw [w aplikacji na system Android]	MIT	https://github.com/AntonPopoff/android-color-wheel
11	ConfigManager.h	Obsługa trybu konfiguracji zegara	MIT	https://github.com/nrwiersma/ConfigManager
12	SparkFunCCS811.h	Moduł czujnika czystości powietrza	MIT	https://github.com/sparkfun/SparkFun_CCS811_Arduino_Library/

Załączniki

Na nośniku CD dołączonym do niniejszej pracy zawarto:

1. Film demonstrujący działanie urządzenia we wszystkich trybach,
2. Pliki projektowe płytki PCB – **pcb_gerber.zip,**
3. Pliki projektowe obudowy – **cover.zip,**
4. Kod źródłowy programu zegara – **source.zip**,
5. Karty katalogowe modułów elektronicznych – **datasheets.zip**.