W A R S Z A W S K A

W Y Ż S Z A S Z K O Ł A I N F O R M A T Y K I

P R A C A D Y P L O M O W A

S T U D I A P I E R W S Z E G O S T O P N I A

Grzegorz Furga

Numer albumu 8196

Inteligentny zegarek opracowany w oparciu o Raspberry Pi

Promotor:

dr inż. Piotr Kopciał

Praca spełnia wymagania stawiane pracom dyplomowym na studiach pierwszego stopnia.

W A R S Z A W A 2021

SPIS TREŚCI

1. Wprowadzenie do dziedziny

1.1. Wstęp

Historia Raspberry Pi rozpoczyna się w 2006 roku od grupy wykładowców z uniwersytetu w Cambridge. Eben Upton wraz z Robem Mullinsem, Jackiem Langiem i Alanem Mycroftem rozpoczęli prace nad tanim, niewielkim komputerem przeznaczonym dla programistów i majsterkowiczów. Założyciele fundacji Raspberry Pi tworzyli projekt z myślą zachęcenia dzieci i młodzieży do nauki programowania oraz wykorzystania nabytej wiedzy w praktyce. Jak się z czasem okazało – platforma z drobnej, skierowanej na konkretną grupę odbiorców, przemieniła się w coś bardziej poważnego, docenionego przez duże grono odbiorców i znajdującego coraz szersze zastosowanie.

Zastosowanie mikro-komputera Raspberry w niniejszej pracy umożliwiło zbudowanie inteligentnego zegarka wyświetającego wiele treści, takich jak:

- Godzina,
- Data,
- Dane pogodowe,
- Dowolny tekst,
- Temperatura w pomieszczeniu.

Dodatkowo system posiada możliwość tworzenia nowych modułów, takich jak np.:

- Licznik odwiedzin w portalu społecznościowym,
- Powiadomienia o nowej wiadomości @-mail,
- Kontrola inteligentnego domu.

Są to tylko przykłady, które użytkownik może stworzyć we własnym zakresie i zainstalować w urządzeniu w podobny sposób jak aplikacje na smartfonie.

Obraz 1 - Gotowe urządzenie. (źródło: opracowanie własne)

2. Cel i zakres pracy

Celem niniejszej pracy inżynierskiej było zaprojektowanie i zbudowanie inteligentnego zegarka wraz z oprogramowaniem i biblioteką do tworzenia dodatkowych modułów.

Całość opiera się na Raspberry Pi, a jako wyświetlacz została wykorzystana matryca diod LED [1] WS2412b. Oprogramowanie zostało napisane w języku C#. Zarządzanie systemem odbywa się przez przygotowany do tego panel administracyjny dostępny za pomocą przeglądarki internetowej.

[1] LED - ang. light-emitting diode: dioda emitująca światło

3. Analiza istniejących rozwiązań

3.1. Lametric Time

Urządzenie Lametric Time umożliwia wyświetlenie informacji pobranych z zainstalowanych w systemie aplikacji, pobranych za pośrednictwem smartfona. Projekt tego urządzenia został założony w Maju 2013 roku przez Nazara Bilous(CEO), Dmytra Baryskyya i Igora Kishchaka. Po roku prototypowania, w czerwcu 2014, LaMetric wypuścił swój pierwszy produkt na Kickstarterze i nazwał go LaMetric Time. Kampania crowdfundingowa otrzymała 370 001 USD od 2215 sponsorów z całego świata. W maju 2016 roku firma rozpoczęła sprzedaż międzynarodową, a we wrześniu tego roku LaMetric Time stał się częścią Amazon Launchpad w USA, Wielkiej Brytanii, Niemczech, Francji i Kanadzie. W październiku 2017 roku produkt przekroczył próg sprzedaży w 20 krajach, wchodząc na rynki Australii i Japonii.

Przykładowe aplikacje:

Obraz 2 - Zegar (źródło: https://lametric.com/)

Obraz 3 - Pogoda (źródło: https://lametric.com/)

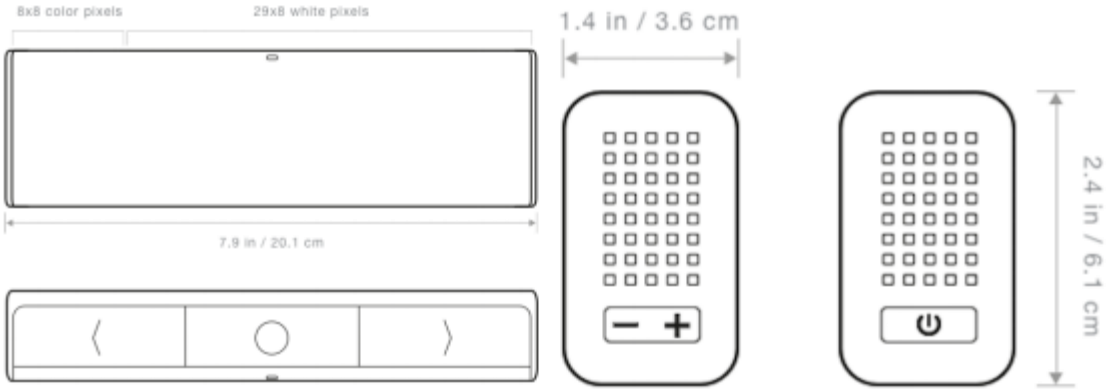
Obraz 4 - Minutnik (źródło: https://lametric.com/)

Obraz 5 - Powiadomienia ze smartfona (źródło: https://lametric.com/)

Specyfikacja techniczna:

Tabela 1 - Specyfikacja techniczna. (źródło: https://lametric.com/)

Lametric Time	
Model	LM 37X8
Waga	223 g
Wymiary	201 x 61 x 36 mm
Łączność	802. 11 b/g/n 2.4 GHz WiFi Bluetooth 4.0
Typ wyświetlacza	LED 37 x 8 białe piksele[2] + 8x8 kolorowe piksele
Wymagania	iPhone (iOS 10.0 lub nowszy) / Android (5.0 lub nowszy)
Audio	2 x 2W wbudowane głośniki
Zasilanie	5V, 2A



Rysunek 1 - Wymiary urządzenia Lametric Time (źródło: https://lametric.com/)

[2] Piksel - najmniejszy jednolity przedstawiający konkretny kolor element obrazu prezentowanego na wyświetlaczach

3.2. Divoom Tivoo

Divoom to firma zajmująca się urządzeniami audio, która projektuje, rozwija i produkuje głośniki bluetooth. Została założona w 2012 roku, z siedzibą znajdującą się w Wanchai, Inne, HK. Divoom Tivoo to urządzenie, które posiada programowalny panel LED 16×16 RGB, pełno zakresowy głośnik o mocy 6 W i Bluetooth 5.0. Sterowanie głośnikiem odbywa się przez aplikacje na smartfona. Na ekranie można wyświetlić proste obrazki i animacje, albo funkcje informacyjne, jak aktualny czas, czy minutnik. Ponadto można też uruchomić proste gry.

Obraz 6 - Podgląd urządzenia. (źródło: https://divoom.pl/)

Specyfikacja techniczna:

Tabela 2 - Specyfikacja techniczna. (źródło: https://divoom.pl/)

Divoom Tivoo	
Wyświetlacz	16 x 16 RGB LED
Waga	380g
Wymiary	100 x 83 x 83 mm
Audio	6W, ⌀ 45 mm, 80Hz - 20kHz
Akumulator	3000mA, do 6h działania,
Zasilanie	5V, 2A
Łączność	Bluetooth 5.0

4. Założenia i wymagania

4.1. Założenia

Wyświetlacz

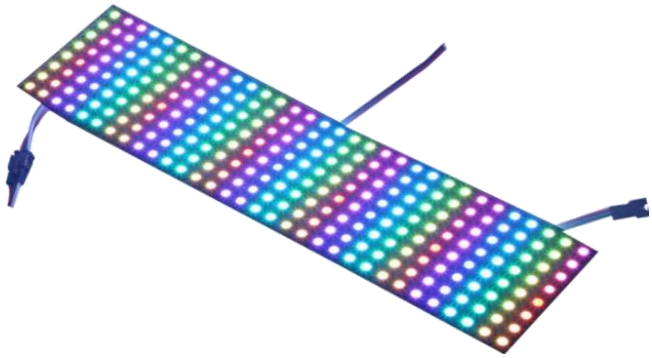
Aby system prawidłowo funkcjonował, powinien mieć czytelny i energooszczędny wyświetlacz. Na rynku istnieje wiele dostępnych rozwiązań wyświetlania obrazu, co zobrazowano w poniższej tabeli:

Tabela 3 - Analiza wyświetlaczy dostępnych na rynku (źródło: opracowanie własne)

Typ	Zalety	Wady
LCD - panel	● Czytelność ● Paleta barw	● Wymaga stałego podświetlania całego obszaru ● Cena
LCD - segmentowy	● Cena ● Energooszczędność	● Bardzo ograniczone możliwości wyświetlanego obrazu ● Monochromatyczny
OLED - panel	● Czytelność ● Paleta barw ● Jakość wyświetlanego obrazu ● Każdy piksel świeci własnym światłem	● Cena ● Podatność na wypalenia
OLED	● Cena ● Energooszczędność ● Każdy piksel świeci własnym światłem	● Rozmiary dostępne na rynku konsumenckim ● Monochromatyczny ● Podatność na wypalenia
E-Ink	● Energooszczędność ● Czytelność w pełnym słońcu	● Cena ● Ograniczona paleta barw ● Nieczytelny w nocy ● Problematyczność wyświetlania animacji
Matryca LED	● Cena ● Energooszczędność ● Paleta barw ● Każdy piksel świeci własnym światłem	● Rozdzielczość obrazu ● Ograniczone możliwości wyświetlanego obrazu

Po przeanalizowaniu dostępnych rozwiązań wybór padł na matrycę LED o wymiarach 32 x 8 [cm] z diodami w rozstawie co 10 [mm], dający łącznie 256 kolorowych pikseli.

Wyświetlacz składa się z diod WS2812b[3]. Panel jest zasilany napięciem stałym o wartości 5V. Taki wyświetlacz idealnie nadaję się do wyświetlania tekstu i prostych animacji.



Obraz 7 - Matryca LED (źródło: https://www.amazon.com/)

Obudowa

Obudowa urządzenia powinna być wykonana w jednej z następujących technologii:

- na drukarce 3D,
- z drewna,
- z metalu,
- z tworzywa sztucznego - wypraska[4],
- z tworzywa sztucznego poddanemu obróbce CNC.

Po przeanalizowaniu dostępnych rozwiązań wybór padł na obudowę wykonaną z PE-UHMW[5] 1000 poddanemu obróbce CNC[6].

Zasilanie

Urządzenie powinno pracować na zasilaniu zewnętrznym i wewnętrznym. Jako zasilanie zewnętrzne wybór padł na zasilacz stabilizowany 12V.

[3] WS2812b - adresowalne diody LED RGB połączone szeregowo
[4] Wypraska - jest uzyskiwana w procesie obróbki tworzyw sztucznych poprzez wtrysk tworzywa do formy
[5] PE-UHMW - typ polietylenu
[6] CNC - rodzaj obróbki ubytkowej polegający na usuwaniu poszczególnych warstw naddatku (źródło wikipedia.org)

Obraz 8 - Zasilacz 12V (źródło: https://elektronikadomowa.pl/)

Dostępne źródła zasilania wewnętrznego to:

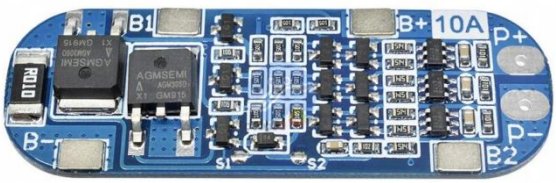
Obraz 9 - Akumulatorki paluszki typu AA (źródło: Komputer Świat)

Obraz 10 - Akumulator kwasowy (źródło: https://www.piekarz.pl/)

Obraz 11 - Ogniwo Li-ion (źródło: https://www.hurt.com.pl/

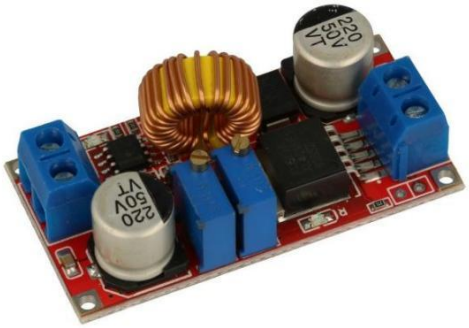
)

Po przeanalizowaniu dostępnych rozwiązań wybór padł na 3 ogniwa litowo jonowe w obudowie typu 18650 z zastosowaniem balansera[7].



Obraz 12 - Balanser do kontroli ładowania ogniw litowo-jonowych (źródło https://allegro.pl/)

Trzy ogniwa dają łącznie napięcie w przedziale 11.1-12.6 Volta, dlatego wymagane jest zastosowanie przetwornicy step-down w celu osiągnięcia napięcia 5 V wymaganego do zasilania mikro-kontrolera i wyświetlacza.



Obraz 13 - przetwornica step-down (źródło: https://www.piekarz.pl/)

Interfejs

Do poprawnego działania wymagana jest aplikacja pozwalająca na zarządzanie systemem, możliwe rozwiązania tego problemu to:

- Aplikacja mobilna,
- Aplikacja komputerowa,
- Aplikacja internetowa.

Z powyższych możliwości wybór padł na zastosowanie responsywnej aplikacji internetowej, dostępnej poprzez adres IP urządzenia.

Oprogramowanie

Do stworzenia oprogramowania systemu wybrano platformę .NET ze względu na dużą dostępność bibliotek do komunikacji z urządzeniami peryferyjnymi.

[7] Balanser - układ elektroniczny odpowiedzialny za kontrole ładowania i rozładowywania ogniw litowo jonowych

4.2. Wymagania

Zaprojektowany system powinien spełniać następujące wymagania:

- działać stabilnie przez długi czas,
- umożliwiać łatwą zmianę położenia,
- działać na zasilaniu bateryjnym,
- mieć prosty i czytelny interfejs,
- interfejs powinien być responsywny,
- umożliwiać łatwą rozszerzalność o nowe funkcjonalności.

Lista funkcji systemu

1. Zarządzanie modułami
    1.1. Instalacja modułów
    1.2. Usuwanie modułów
    1.3. Konfiguracja modułów
    1.4. Sortowanie modułów
2. Ustawienia jasności wyświetlacza
    2.1. Ustawienia jasności wyświetlacza
    2.2. Tryb nocny
    2.3. Ręczne/automatyczne przełączanie4.3.2.

Szczegółowe opisy funkcji

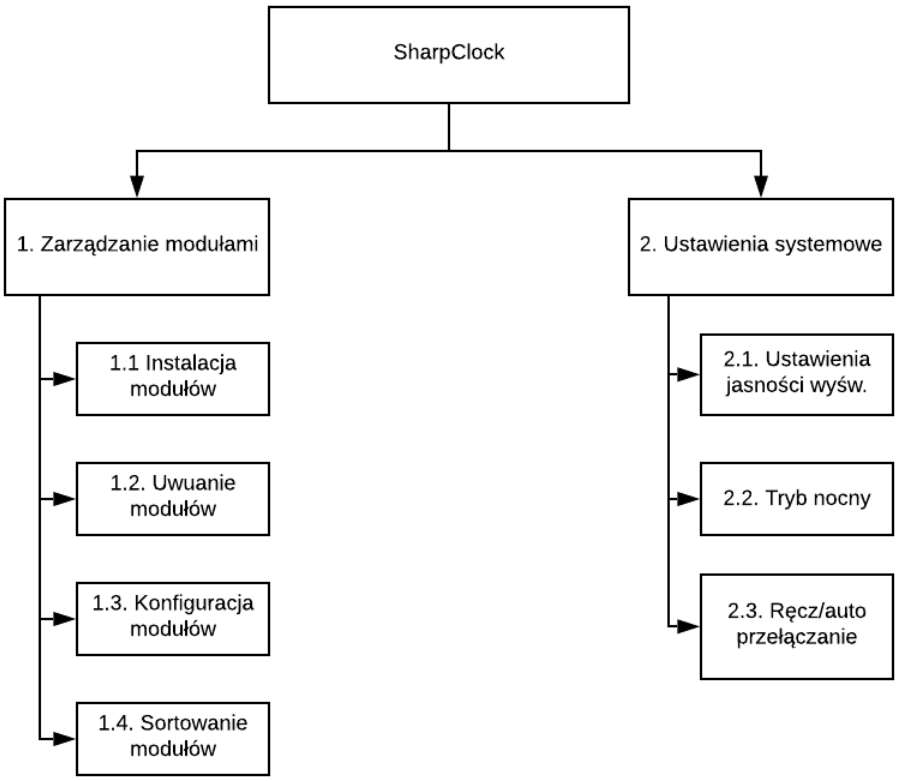
Tabela 4 - Opis funkcji Zarządzanie modułami (źródło: opracowanie własne)

Nazwa Funkcji	Zarządzanie modułami
Opis	Funkcja pozwala na instalacje nowych modułów, konfiguracje i usuwanie zainstalowanych
Dane wejściowe	Plik *.dll modułu, lub dane konfiguracyjne
Źródło danych wejściowych	Dane dostarczone przez użytkownika
Wynik	Dodanie/usunięcie lub zmiana konfiguracji modułu
Warunek wstępny	Poprawnie skompilowany moduł
Warunek końcowy	Moduł nie zwrócił żadnego błędu
Efekty uboczne	Brak
Powód wyróżnienia	Główna funkcja programu

Tabela 5 - Opis funkcji Ustawienia systemowe (źródło: opracowanie własne)

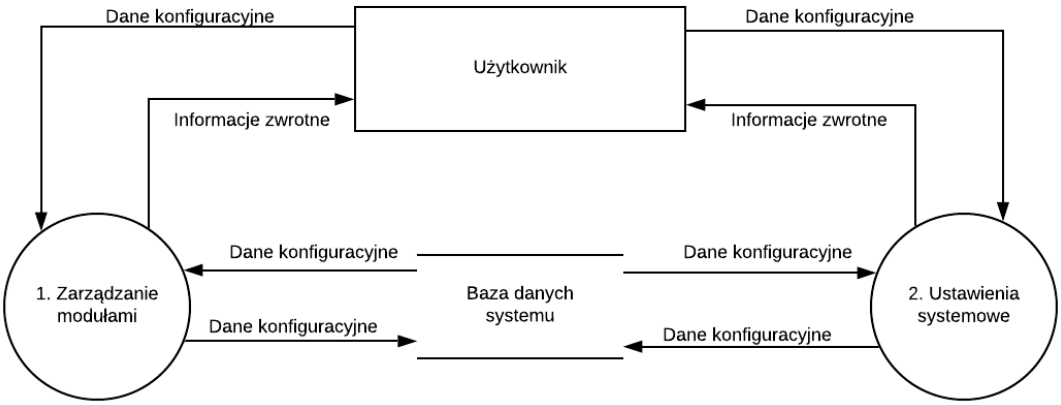
Nazwa Funkcji	Ustawienia systemowe
Opis	Funkcja pozwala na zmianę podstawowych zachowań systemu
Dane wejściowe	Dane konfiguracyjne
Źródło danych wejściowych	Dane dostarczone przez użytkownika
Wynik	Zmiana zachowań systemu
Warunek wstępny	Brak
Warunek końcowy	Brak
Efekty uboczne	Brak
Powód wyróżnienia	Główna funkcja programu

Diagram hierarchii funkcji



Rysunek 2 - Diagram hierarchii funkcji (źródło: opracowanie własne)

Diagram poziomu zerowego

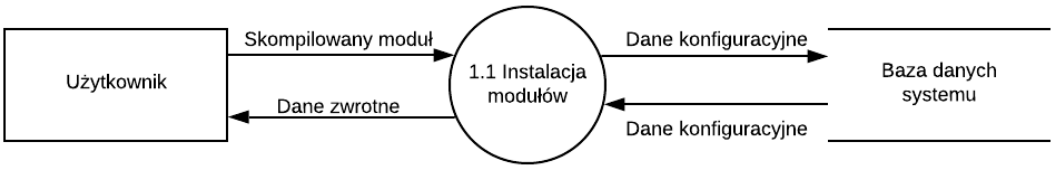


Rysunek 3 - Diagram poziomu zerowego (źródło: opracowanie własne)

Tabela 6 - Opis elementów poziomu zerowego (źródło: opracowanie własne)

Nazwa elementu	Typ elementu	Opis elementu
Zarządzanie modułami	Proces	Wprowadza zmiany w module
Ustawienia systemowe	Proces	Wprowadza zmiany w systemie
Użytkownik	Terminator	Konfiguruje moduły i odczytuje dane
Baza danych systemu	Magazyn danych	Przechowuje dane konfiguracyjne
Dane konfiguracyjne	Przepływ	Przepływ danych konfiguracyjnych
Informacje zwrotne	Przepływ	Przepływ danych zwrotnych z systemu/modułu

Diagram instalacji modułu

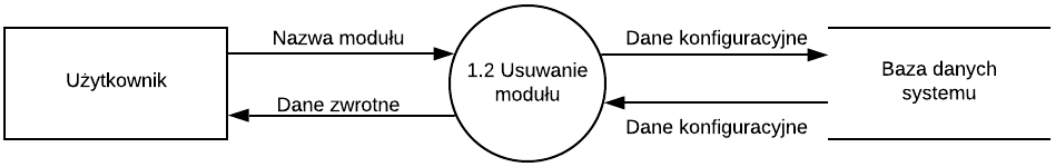


Rysunek 4 - Diagram instalacji modułu (źródło: opracowanie własne)

Tabela 7 - Opis elementów instalacji modułów (źródło: opracowanie własne)

Nazwa elementu	Typ elementu	Opis elementu
Instalacja modułów	Proces	Dodaje plik modułu do systemu
Użytkownik	Terminator	Konfiguruje moduły i odczytuje dane zwrotne
Baza danych systemu	Magazyn danych	Przechowuje dane konfiguracyjne
Dane konfiguracyjne	Przepływ	Przepływ danych konfiguracyjnych
Dane zwrotne	Przepływ	Przepływ danych zwrotnych z modułu
Skompilowany moduł	Przepływ	Przepływ danych zawierających bibliotekę modułu

Diagram usuwania modułów



Rysunek 5 - Diagram usuwania modułu (źródło: opracowanie własne)

Tabela 8 - Opis elementów usuwania modułu (źródło: opracowanie własne)

Nazwa elementu	Typ elementu	Opis elementu
Usuwanie modułu	Proces	Usuwa plik modułu z systemu
Użytkownik	Terminator	Konfiguruje moduły i odczytuje dane zwrotne
Baza danych systemu	Magazyn danych	Przechowuje dane konfiguracyjne
Dane konfiguracyjne	Przepływ	Przepływ danych konfiguracyjnych
Dane zwrotne	Przepływ	Przepływ danych zwrotnych z modułu
Nazwa modułu	Przepływ	Przepływ danych zawierających nazwę modułu do usunięcia

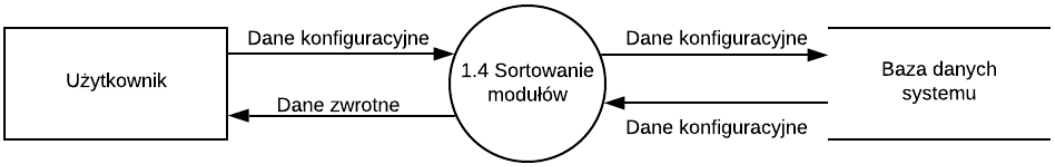
Diagram konfiguracji modułów

Rysunek 6 - Diagram konfiguracji modułów (źródło: opracowanie własne)

Tabela 9 - Opis konfiguracji modułów (źródło: opracowanie własne)

Nazwa elementu	Typ elementu	Opis elementu
Konfiguracja modułów	Proces	Proces przetwarzający dane konfiguracyjne
Użytkownik	Terminator	Konfiguruje moduły i odczytuje dane zwrotne
Baza danych systemu	Magazyn danych	Przechowuje dane konfiguracyjne
Dane konfiguracyjne	Przepływ	Przepływ danych konfiguracyjnych
Dane zwrotne	Przepływ	Przepływ danych zwrotnych z modułu

Diagram sortowania modułów



Rysunek 7 - Diagram sortowania modułów (źródło: opracowanie własne)

Tabela 10 - Opis elementów sortowania modułów (źródło: opracowanie własne)

Nazwa elementu	Typ elementu	Opis elementu
Sortowanie modułów	Proces	Proces sortujący moduły
Użytkownik	Terminator	Konfiguruje moduły i odczytuje dane zwrotne
Baza danych systemu	Magazyn danych	Przechowuje dane konfiguracyjne
Dane konfiguracyjne	Przepływ	Przepływ danych konfiguracyjnych
Dane zwrotne	Przepływ	Przepływ danych zwrotnych z modułu

Diagram ustawienia jasności wyświetlacza

Rysunek 8 - Diagram ustawienia jasności wyświetlacza (źródło: opracowanie własne)

Tabela 11 - Opis elementów ustawiania jasności (źródło: opracowanie własne)

Nazwa elementu	Typ elementu	Opis elementu
Ustawienia jasności wyświetlacza	Proces	Proces ustawiający jasność wyświetlacza
Użytkownik	Terminator	Konfiguruje i odczytuje dane zwrotne
Baza danych systemu	Magazyn danych	Przechowuje dane konfiguracyjne
Dane konfiguracyjne	Przepływ	Przepływ danych konfiguracyjnych
Dane zwrotne	Przepływ	Przepływ danych zwrotnych z systemu

Diagram ustawienia trybu nocnego

Rysunek 9 - Diagram ustawienia trybu nocnego (źródło: opracowanie własne)

Tabela 12 - Opis ustawienia trybu nocnego (źródło: opracowanie własne)

Nazwa elementu	Typ elementu	Opis elementu
Ustawienia trybu nocnego	Proces	Proces ustawiający właściwości trybu nocnego
Użytkownik	Terminator	Konfiguruje i odczytuje dane zwrotne
Baza danych systemu	Magazyn danych	Przechowuje dane konfiguracyjne
Dane konfiguracyjne	Przepływ	Przepływ danych konfiguracyjnych
Dane zwrotne	Przepływ	Przepływ danych zwrotnych z systemu

Diagram ustawienia automatycznego lub ręcznego przełączania

Rysunek 10 - Diagram ustawienia automatycznego przełączania (źródło: opracowanie własne)

Tabela 13 - Opis ustawienia automatycznego przełączania (źródło: opracowanie własne)

Nazwa elementu	Typ elementu	Opis elementu
Ustawienia przełączania	Proces	Proces ustawiający właściwości trybu przełączania
Użytkownik	Terminator	Konfiguruje i odczytuje dane zwrotne
Baza danych systemu	Magazyn danych	Przechowuje dane konfiguracyjne
Dane konfiguracyjne	Przepływ	Przepływ danych konfiguracyjnych
Dane zwrotne	Przepływ	Przepływ danych zwrotnych z systemu

5. Projekt i realizacja

W projekcie wykorzystano mikro-komputer Raspberry Pi w wersji Zero W. Ten model wyróżniają małe wymiary, niski pobór prądu, wbudowany moduł Wi-Fi oraz Bluetooth.

Obraz 14 - Mikro-komputer Raspberry Pi Zero W (źródło: https://botland.com.pl/)

Tabela 14 - Specyfikacja techniczna mikro-komputera.
(źródło:https://pl.wikipedia.org/wiki/Raspberry_Pi)

Raspberry Pi Zero W	
Procesor	Broadcom BCM2835 1 GHz ARM1176JZF-S
Układ graficzny	Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decode
Pamięć operacyjna	512 MB (współdzielona z GPU)
Pamięć masowa	Micro SD
Zasilanie	2.5A, 5V poprzez Micro USB lub GPIO[8]
Złącza	Micro SD, Mini HDMI[9], 2x Micro USB, 40x GPIO, Złącze kamery CSI[10], Złącze kompozytowe[11], Reset
Połączenia sieciowe	Wi-Fi (2,4 GHz 802.11n), Bluetooth 4.0
Wymiary	65 x 30 x 5 mm
Waga	9g
Obsługiwane systemy operacyjne	Raspbian, Debian, Fedora, Arch Linux, FreeBSD

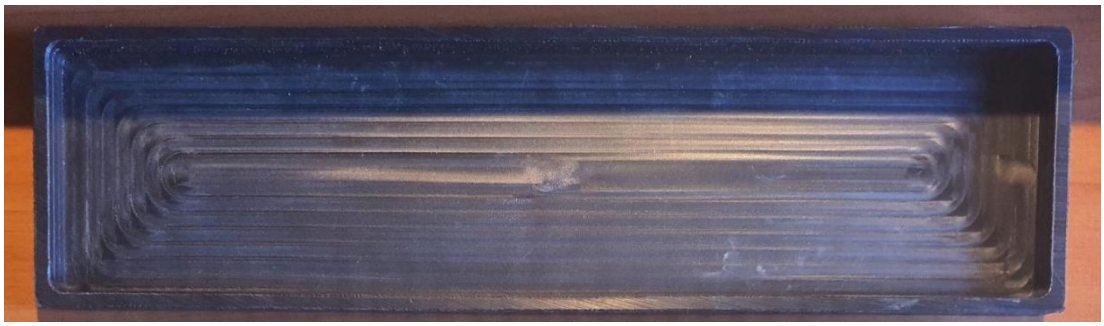
[8] GPIO - ang. general-purpose input/output, interfejs do komunikacji między elementami systemu komputerowego
[9] HDMI - ang. High Definition Multimedia Interface, multimedialny interfejs wysokiej rozdzielczości. interfejs służący do przesyłania cyfrowego, nieskompresowanego sygnału audio i wideo
[10] CSI - ang. Camera Serial Interface, interfejs do komunikacji z kamerą
[11] Composite - format analogowego sygnału wideo

5.1. Obudowa

Korpus obudowy został wykonany na frezarce CNC z polietylenu.

Obraz 15 - Korpus obudowy (źródło: opracowanie własne)

Tylna część obudowy została wykonana w taki sam sposób jak korpus.



Obraz 16 - Tylna część, zasłaniająca elektronikę (źródło: opracowanie własne)

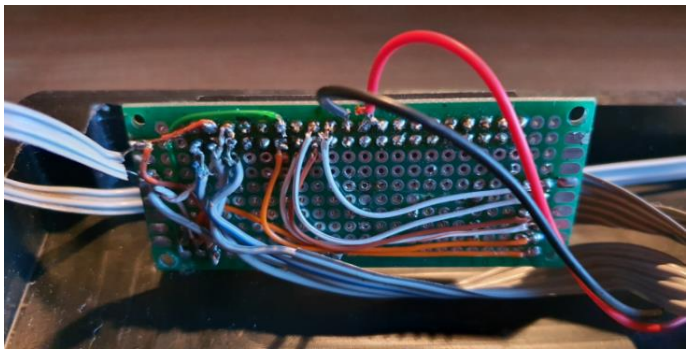
Część zasłaniająca wyświetlacz została wykonana z pleksi na frezarce CNC i pomalowana cienką warstwą czarnej farby w spreju.

Obraz 17 - Przednia część zasłaniająca ekran (źródło: opracowanie własne)

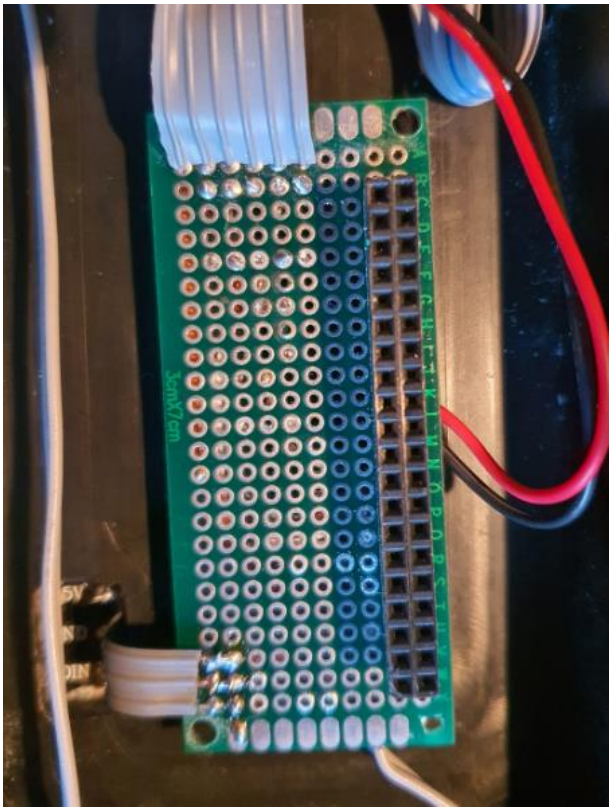
5.2. Elektronika

Główna płytka

Część elektroniczną wykonano na płytce uniwersalnej, tak aby część obliczeniowa była łatwo dostępna w razie ewentualnej awarii.

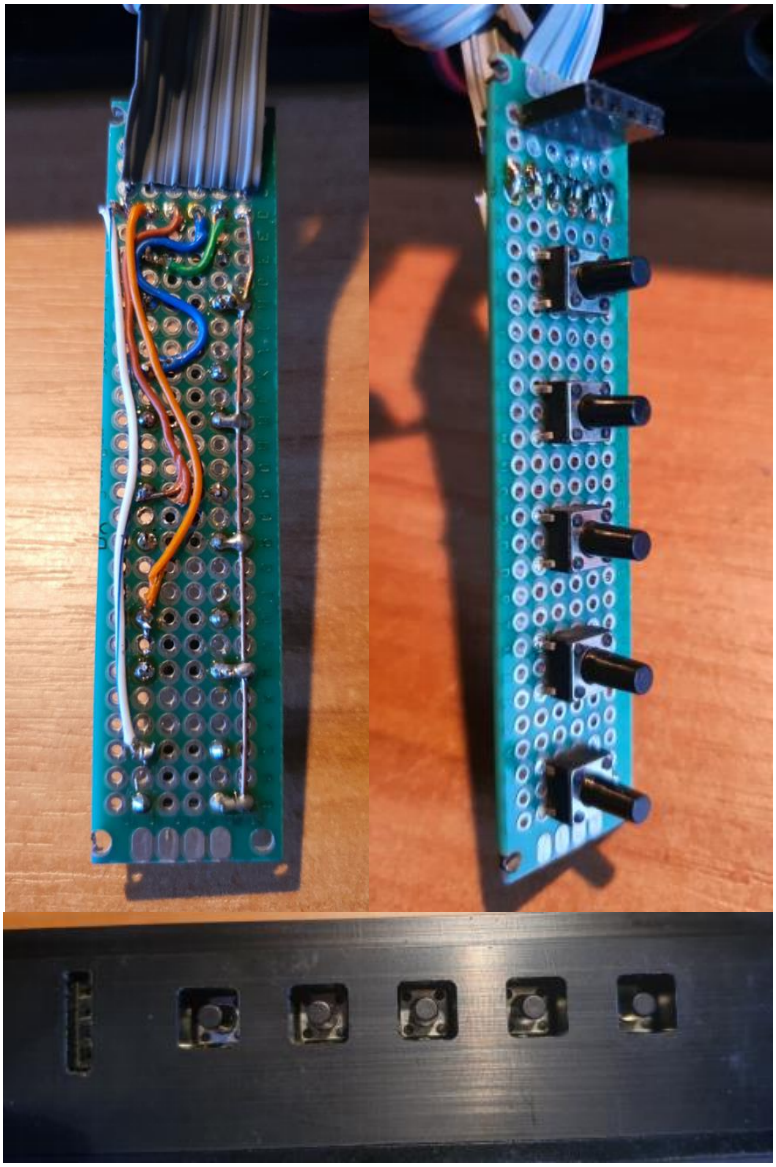


Obraz 18 - Spód głównej płytki (źródło: opracowanie własne)



Obraz 19 - Góra głównej płytki (źródło: opracowanie własne)

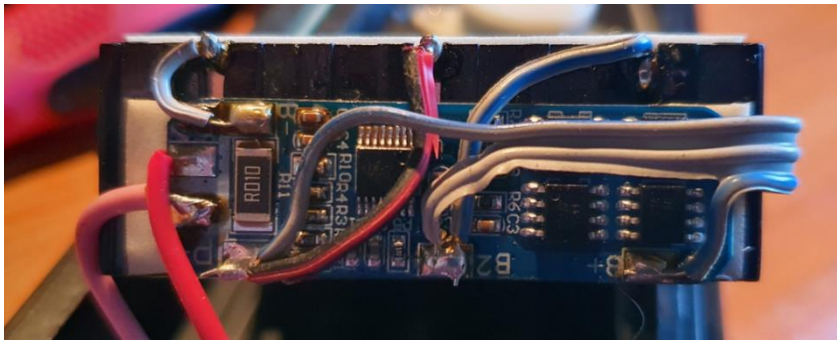
Sekcja przycisków



Obraz 20 - Sekcja przycisków (źródło: opracowanie własne)

Obraz 21 - Przyciski (źródło: opracowanie własne)

Sekcja zasilania

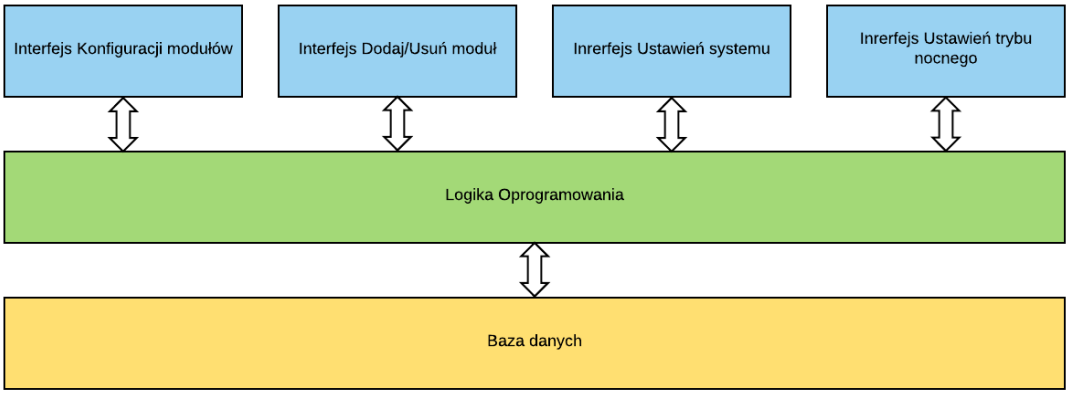


Obraz 22 - Kontroler ogniw (źródło: opracowanie własne)

Obraz 23 - Koszyczek na ogniwa i przetwornica (źródło: opracowanie własne)

5.3. Architektura oprogramowania SharpClock

Interfejsy zostały wykonane w oparciu o formularze HTML, które za pomocą mechanizmu HTTP[12] POST[13] komunikują się z logiką oprogramowania, gdzie zapytanie jest przetwarzane i zapisywane do bazy danych.



Rysunek 11 - Architektura oprogramowania (źródło: opracowanie własne)

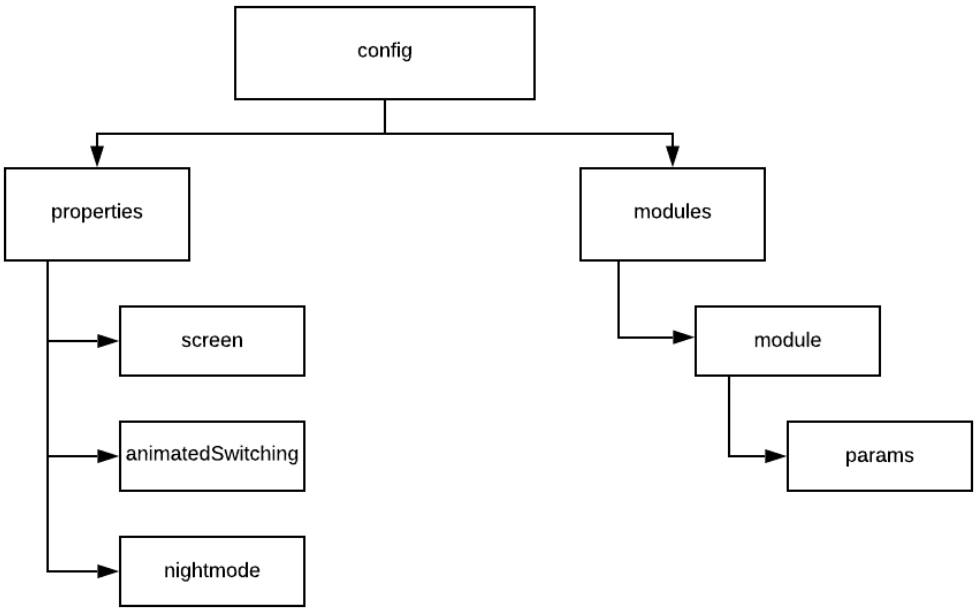
[12] HTTP - (ang. Hypertext Transfer Protocol) Za pomocą protokołu HTTP przesyła się żądania udostępnienia dokumentów WWW

[13] POST - metoda przesyłania danych w sieci internet. Istnieje w ramach protokołu HTTP i wykorzystywana jest najczęściej do wysłania informacji z formularza znajdującego się na stronie internetowej.

Program SharpClock napisano w środowisku Visual Studio 2019 z wykorzystaniem technologii .NET Framework. Jako środowiska uruchomieniowego użyto oprogramowania Mono Develop działającego pod systemem Linux. Oprogramowanie działa w oparciu o XML'ową baze danych, z powodu niskich wymagań tego typu rozwiązania.

5.4. Projekt bazy danych

Baza danych została oparta o model hierarchiczny. Schemat hierarchii bazy danych przedstawiono na rys. 12. Poszczególne elementy widoczne na rysunku opisano w tabeli 15.



Rysunek 12 - Schemat hierarchii bazy danych (źródło: opracowanie własne)

Szczegółowy opis znaczników przedstawiono w tabeli poniżej.

Tabela 15 - Szczegółowy opis znaczników (źródło: opracowanie własne)

L.p	Nazwa znacznika	Przynależność	Opis
1	Properties	Config	Zawiera dane ustawień systemu
2	Modules	Config	Zawiera dane konfiguracji modułów
3	Screen	Properties	Zawiera dane ustawień ekranu, np: jasność
4	animatedSwitching	Properties	Zawiera informacje o tym czy przełączanie modułów jest płynne
5	nightmode	Properties	Zawiera dane o trybie nocnym, np: status włączenia, aktywne godziny
6	module	Modules	Zawiera dane dotyczące poszczególnych modułów: Nazwa modułu, status modułu
7	params	module	Zawiera dane konfiguracyjne modułu zdefiniowane przez programistę modułu. Atrybuty systemowe to: Timer i widoczność

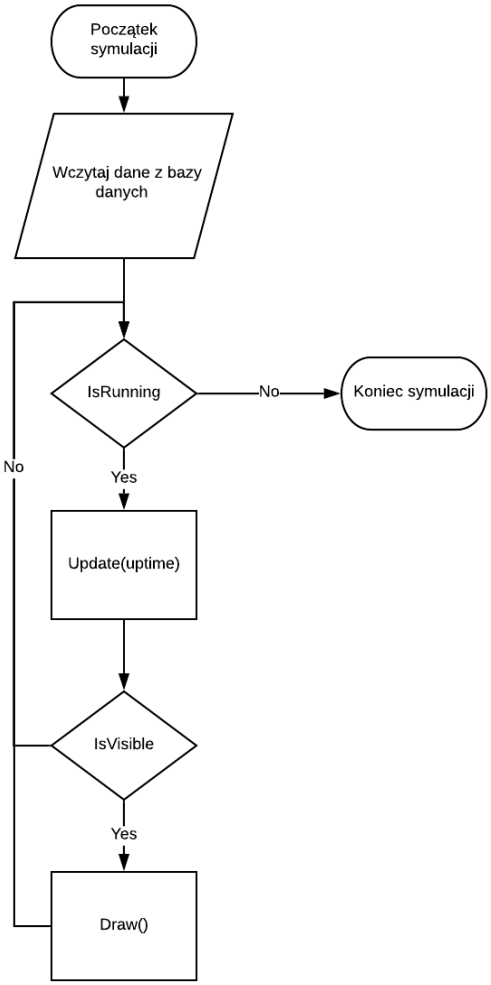
5.5. Projekt logiki działania

5.5.1. Schemat blokowy zachowania modułu

Podczas startu modułu system pobiera z bazy danych konfiguracje modułu np: czy jest uruchomiony, czy jest widoczny, czas wyświetlania na wyświetlaczu, dane lokalizacji w przypadku modułu pogodowego, kolor tekstu w przypadku modułu wyświetlającego tekst.

Działanie modułu odbywa się asynchronicznie w dwóch pętlach:

- Metoda Update() odpowiedzialna za działanie logiki modułu np: pobranie z api[14] danych pogodowych co 10 minut,
- Metoda Draw() odpowiedzialna za wysłanie instrukcji do wyświetlacza, system stara się wykonać metodę 30 razy na sekundę, w celu zachowania płynności obrazu.

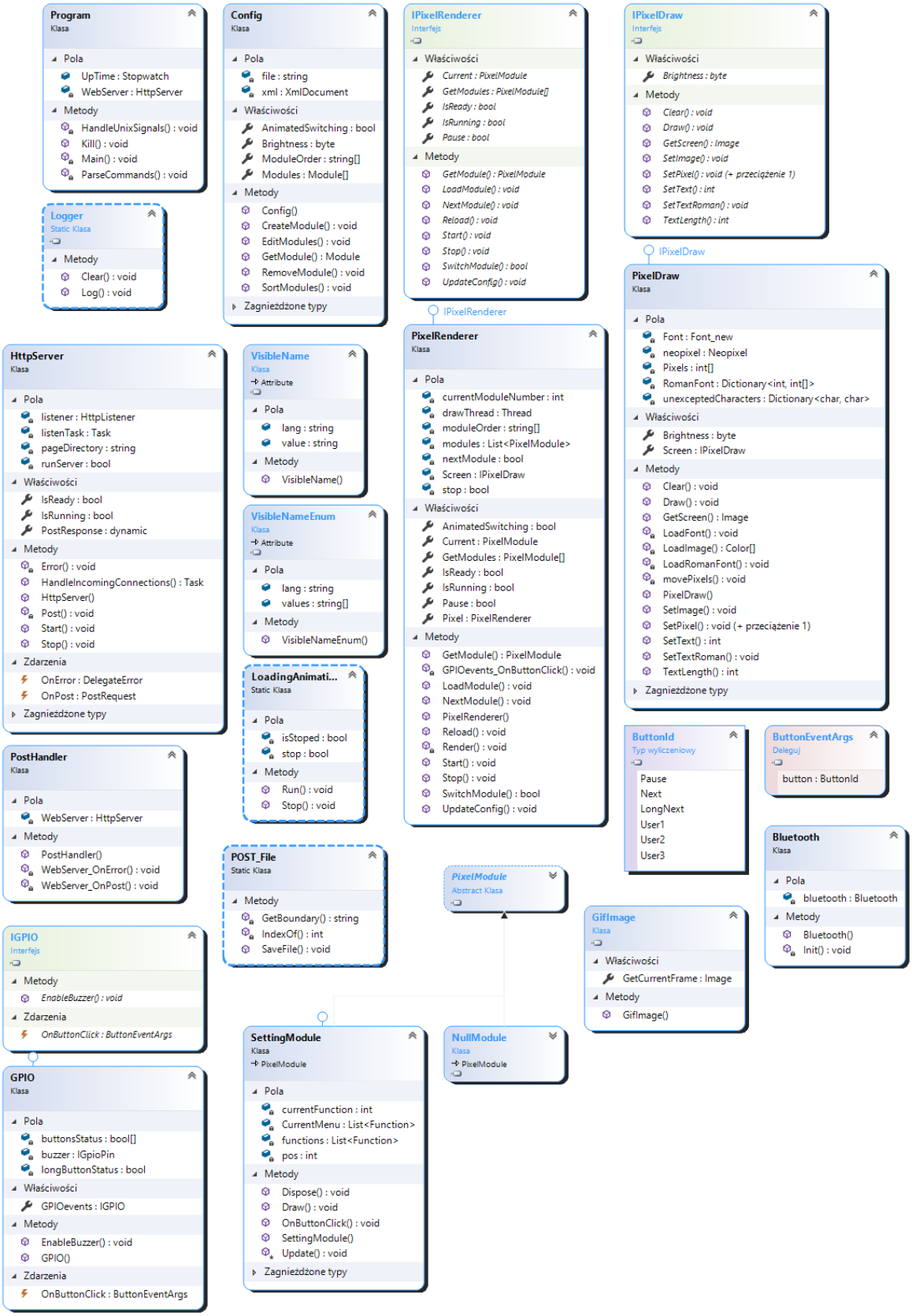


Rysunek 13 - Schemat blokowy zachowania modułu (źródło: opracowanie własne)

[14] API - (ang. application programming interface) - zbiór reguł ściśle opisujący, w jaki sposób programy lub podprogramy komunikują się ze sobą.

5.5.2. Diagram i opisy klas

Aplikacja została napisana w paradygmacie obiektowym. Na poniższym diagramie przedstawiono strukturę klas zaimplementowanych w systemie.



Rysunek 14 - Diagram klas (źródło: opracowanie własne)

Opis klasy Program

Tabela 16 - *Opis klasy program* (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	UpTime	Stopwatch	public	Liczy czas od uruchomienia programu
2	WebServer	HttpServer	private	Klasa obsługująca serwer www do zarządzania aplikacją
3	HandlUnixSignal()	void	private	Obsługa unixSignal
4	Kill()	void	public	Służy do poprawnego zatrzymania programu
5	Main()	void	private	Główny punkt wejścia
6	ParseCommands(string[] args)	void	private	Obsługa argumentów podanych przy starcie programu

Opis klasy Logger

Tabela 17 - *Opis klasy Logger* (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	Clear()	void	public	Czyści log systemu
2	Log(params object[] args)	void	public	Zapisuje do pliku logi systemu

Opis klasy HttpServer

Tabela 18 - Opis klasy HttpServer (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	listener	HttpListener	private	Obsługuje zapytania http
2	listenTask	Task	private	Task obsługujący zapytania http
3	pageDirectory	String	private	Zawiera ścieżkę do katalogu z plikami strony
4	runServer	bool	private	Utrzymuje pętle serwera
5	IsReady	bool	public	Zwraca informacje czy serwer jest gotowy
6	IsRunning	bool	public	Zwraca informacje czy serwer jest uruchomiony
7	PostResponse	dynamic	public	Pole zawiera dane, które serwer ma przekazać klientowi
8	Error()	void	private	Zostaje wywołane gdy serwer napotka błąd
9	HandleIncomingConnections()	Task	private	Obsługuje zapytania http, powiązane z polem listenTask
10	Post(string path, NameValueCollection query)	void	private	Zostaje wywołane gdy klient wykona zapytanie typu POST
11	Start()	void	public	Uruchamia serwer
12	Stop()	void	public	Zatrzymuje serwer
13	OnError	DelegateError	public	Zdarzenie wywołane gdy serwer napotka błąd

14	OnPost	PostRequest	public	Zdarzenie wywołane gdy klient wykona zapytanie typu post

Opis klasy POST_File

Tabela 19 - Opis klasy POST_File (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	GetBoundry	string	private	Zwraca granice zapytania
2	IndexOf	int	private	Zwraca pozycje granicy
3	SaveFile	void	public	Zapisuje plik odebrany POSTem

Opis klasy Config

Tabela 20 - Opis klasy Config (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	file	string	private	Zawiera ścieżkę do pliku bazy danych
2	xml	XmlDocument	private	Obsługuje komunikacje z bazą danych
3	AnimatedSwitching	bool	public	Właściwość odpowiedzialna za ustawienie typu przewijania: płynne czy szybkie
4	Brightness	byte	public	Właściwość odpowiedzialna za ustawienie jasności wyświetlacza
5	ModuleOrder	String[]	public	Zwraca listę modułów w kolejności jakiej będą wyświetlane
6	Modules	PixelModule[]	public	Zwraca listę modułów
7	CreateModule(PixelModule module)	void	public	Dodaje do bazy dane nowego modułu
8	EditModules(PixelModules[ ] modules)	void	public	Zmienia ustawienia modułów
9	GetModule(string name)	PixelModule	public	Zwraca moduł po nazwie
10	RemoveModule(string name)	void	public	Usuwa dane modułu z bazy
11	SortModules(string[] names)	void	public	Ustawia kolejność wyświetlania modułów

Opis klasy LoadingAnimation

Tabela 21 - Opis klasy LoadingAnimation (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub metody	Typ	Dostęp	Przeznaczenie
1	isStoped	bool	private	Zwraca informacje o tym czy animacja jest aktywna
2	stop	bool	private	Utrzymuje pętle animacji
3	Run()	void	public	Startuje animacje
4	Stop()	void	public	Zatrzymuje animacje

Opis klasy GifImage

Tabela 22 - Opis klasy GifImage (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	GetCurrentFrame	Image	public	Zwraca pojedynczą ramke obrazu z gifu

Opis klasy PostHandler

Tabela 23 - Opis klasy PostHandler (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	WebServer	HttpServer	private	Zawiera referencje do serwera WWW
2	WebServer_OnError	void	private	Obsługuję błąd serwera www
3	WebServer_OnPost	void	private	Obsługuje zapytania POST

Opis interfejsu IPixelRender

Tabela 24 - Opis interfejsu IPixelRenderer (źródło: opracowanie własne)

L.p	Nazwa właściwości lub nagłówek metody	Typ
1	Current	PixelModule
2	GetModules	PixelModule[]
3	IsReady	bool
4	IsRunning	bool
5	Pause	bool
6	GetModule(string name)	PixelModule
7	LoadModule(string path)	void
8	NextModule()	void
9	Reload	void
10	Start	void
11	Stop	void
12	SwitchModule(PixelModule module, bool forcePause)	bool
13	UpdateConfig	void

Opis klasy PixelRenderer

Tabela 25 - Opis klasy PixelRenderer (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	currentModuleNumber	int	private	Zwraca numer aktualnie wyświetlanego modułu
2	drawThred	Thred	private	Wątek odpowiedzialny za rysowanie modułu na wyświetlaczu
3	moduleOrder	String[]	private	Tablica zawierająca nazwy modułów w kolejności wyświetlania
4	modules	List <PixelModule>	private	Lista modułów
5	nextModule	bool	private	Jeżeli wartość jest true wymusza przełączenie na następny moduł

6	Screen	IPixelDraw	private	Odpowiada za komunikacje z wyświetlaczem
7	stop	bool	private	Utrzymuje pętle systemu
8	AnimatedSwitching	bool	public	Właściwość odpowiedzialna za zmianę animacji przewijania
9	Current	PixelModule	public	Zwraca aktualny moduł
10	GetModules	PixelModule[]	public	Zwraca listę aktualnie załadowanych modułów
11	IsReady	bool	public	Zwraca informacje czy ststem jest gotowy
12	IsRunning	bool	public	Zwraca informacje czy system jest uruchomiony
13	Pause	bool	public	Odpowiada za wstrzymanie automatycznego przewijania
14	Pixel	PixelRenderer	public	Singleton PixelRenderer
15	GetModule(string name)	PixelModule	public	Zwraca moduł po nazwie
16	GPIOevents_OnButton Click(ButtonId button)	void	public	Zostaje wywołane w momencie naciśnięcia przycisku na obudowie
17	LoadModule(string path)	void	public	Wczytuje plik modułu do systemu
18	NextModule()	void	public	Przełącza na następny moduł
19	Reload()	void	public	Przeładowuje system
20	Render()	void	private	Odpowiada za wykonanie zadań modułu i narysowanie go na ekranie
21	Start()	void	public	Startuje system
22	Stop()	void	public	Zatrzymuje system
23	SwitchModule(PixelMo dule module, bool forcePause)	bool	public	Przełącza system na podany moduł
24	UpdateConfig()	void	public	Zapisuje aktualną konfiguracje

Opis interfejsu IPixelDraw

Tabela 26 - Opis interfejsu IPixelDraw (źródło: opracowanie własne)

L.p	Nazwa właściwości lub nagłówek metody	Typ
1	Brightness	byte
2	Clear()	void
3	Draw(int offset)	void
4	GetScreen()	Image
5	SetImage(Image image, int x, int width, int height)	void
6	SetPixel(int number, Color c)	void
7	SetPixel(Point point, Color c)	void
8	SetText(string text, Color c, int x, int spaces)	void
9	SetTextRoman(int number, Color c, int x)	void
10	TextLength(string text, int spaces, int x)	int

Opis klasy PixelDraw

Tabela 27 - Opis klasy PixelDraw (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	Font	Font_new	private	Przechowuje informacje o czcionce
2	neopixel	Neopixel	private	Klasa z zewnętrznej biblioteki, służy do niskopoziomowej komunikacji z wyświetlaczem
3	Pixels	Int[]	private	Tablica przekształca pozycje pixeli na użyteczne dla programu
4	RomanFont	Dictionary<int, int[]>	private	Przechowuje informacje na temat liczb rzymskich
5	unexceptedCharacters	Dictionary<char,char>	private	Zamienia polskie znaki
6	Brightness	byte	public	Pozwala ustawić jasność wyświetlacza
7	Screen	IPixelDraw	public	Singleton PixelDraw
8	Clear()	void	public	Zeruje stan ekranu
9	Draw(int offset)	void	public	Rysuje dane z pamięci na wyświetlaczu
10	GetScreen()	Image	public	Zwraca obraz aktualnego stanu wyświetlacza
11	LoadFont(string path)	void	public	Wczytuje plik czcionki
12	LoadImage(Image image, Point point, int width, int height)	Color[]	public	Zamienia obraz na tablice punktów
13	LoadRomanFont(string path)	void	public	Wczytuje czcionkę liczb rzymskich
14	movePixels(int offset)	void	public	Przesuwa obraz na wyświetlaczu n wierszy
15	SetImage(Image image, int x, int width, int height)	void	public	Wczytuje obraz do pamięci wyświetlacza
16	SetPixel(int number, Color c)	void	public	Ustawia pixel na określony kolor
17	SetPixel(Point point, Color c)	void	public	Ustawia pixel na określony kolor
18	SetText(string text, Color c, int x, int spaces)	void	public	Ustawia tekst na wyświetlaczu
19	SetRomanText(int number, Color c, int x)	void	public	Ustawia cyfrę rzymską na wyświetlaczu
20	TextLength(string text, int spaces, int x)	int	public	Zwraca ilość kolumn jaką zajmie tekst na wyświetlaczu

Opis interfejsu IGPIO

Tabela 28 - Opis interfejsu IGPIO (źródło: opracowanie własne)

L.p	Nazwa właściwości lub nagłówek metody	Typ
1	EnableBuzzer(int amount, int howLong, int interval)	void
2	OnButtonClick	ButtonEventArgs

Opis klasy GPIO

Tabela 29 - Opis klasy GPIO (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	buttonsStatus	Bool[]	private	Przechowuje stan przycisków
2	buzzer	IGpioPin	private	Przechowuje numer pinu do którego podłączony jest buzzer
3	longButtonStatus	bool	private	Zwraca prawdę jeżeli przycisk został przytrzymany
4	GPIOevents	IGPIO	public	Singleton klasy GPIO
5	EnableBuzzer(int amount, int howLong, int interval)	void	public	Włącza buzzer
6	OnButtonClick	ButtonEventArgs	public	Zdarzenie wywoływane w momencie naciśnięcia przycisku

Opis klasy Bluetooth

Tabela 30 - Opis klasy Bluetooth (źródło: opracowanie własne)

L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	bluetooth	Bluetooth	private	Przechowuje referencje do urządzenia bluetooth
2	Init()	void	private	Inicjalizacja urządzenia bluetooth

Opis klasy abstrakcyjnej PixelModule

Tabela 31 - Opis klasy abstrakcyjnej PixelModule (źródło: opracowanie własne)

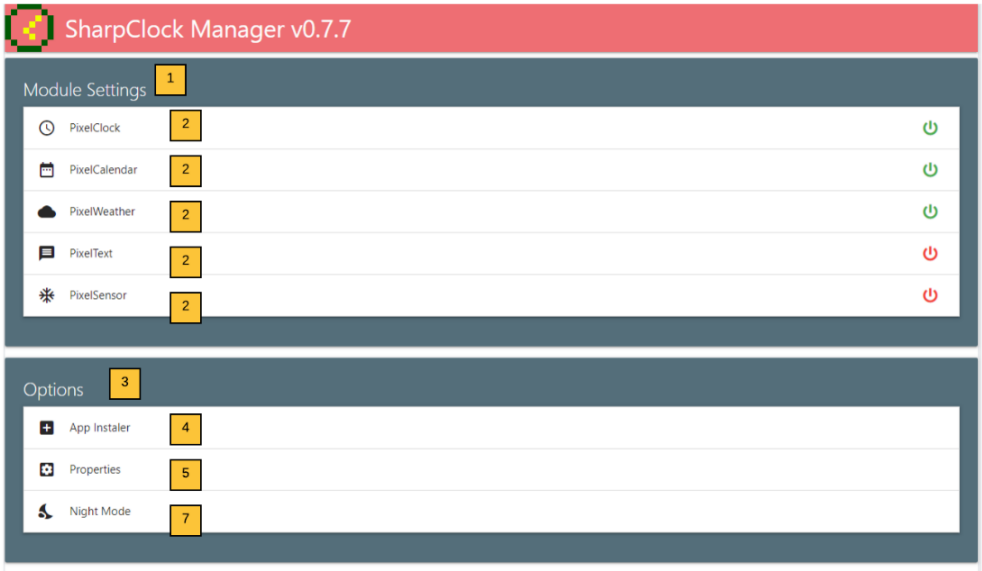
L.p	Nazwa zmiennej lub nagłówek metody	Typ	Dostęp	Przeznaczenie
1	GPIOevents	IGPIO	protected	Pozwala na używanie buzzera i przycisków przez moduł
2	Icon	string	public	Nazwa ikonki do wyświetlenia na stronie
3	IsRunning	bool	public	Zwaraca informacje o tym czy moduł jest uruchomiony
4	Name	string	public	Zwraca nazwe modułu
5	pixelRender	IPixelRender	protected	Daje dostęp do klasy PixelRender wewnątrz modułu
6	Screen	IPixelDraw	protected	Daje dostęp do wyświetlacza
7	Tickrate	int	protected	Ustawia czas praczy logiki modułu

8	Timer	int	public	Ustawia czas wyświetlania modułu
9	Visible	bool	public	Ustala czy moduł jest widoczny czy nie
10	Draw(Stopwatch stopwatch)	void	public	Rysuje moduł na wyświetlaczu
11	OnButtonClick(ButtonId button)	void	public	Zostaje wywołane w momencie naciśnięcia przycisku na obudowie
12	Reload	void	public	Przeładowuje moduł
13	SetGPIO(IGPIO gpio)	void	public	Przekazuje referencje GPIO do modułu
14	SetRenderer(IPixelRenderer pixelRender)	void	public	Przekazuje referencje PixelRender do modułu
15	SetScreen(IPixelScreen screen)	void	public	Przekazuje referencje PixelScreen do modułu
16	Start(Stopwatch stopwatch)	void	public	Uruchamia logikę modułu
17	Stop()	void	public	Zatrzymuje logikę modułu
18	Update(Stopwatch stopwatch)	void	public	Wykonuje logikę modułu

5.6. Projekt interfejsów graficznych

Menu główne

Widok bezpośrednio po uruchomieniu panelu przedstawiono na rys. 15. Poszczególne elementy widoczne na rysunku ponumerowano, a ich funkcje opisano w tabeli 32.



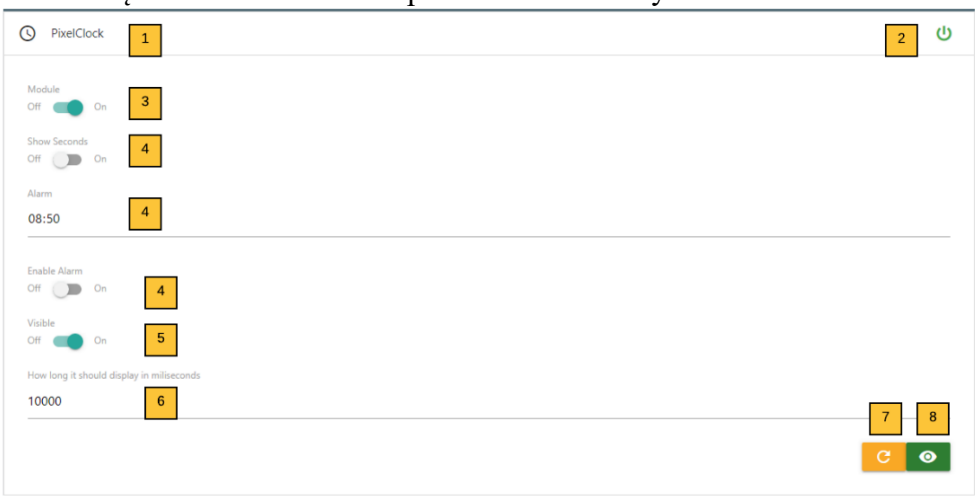
Rysunek 15 - Menu główne (źródło: opracowanie własne)

Tabela 32 - Opis elementów menu (źródło: opracowanie własne)

L.p	Typ obiektu	Treść	Funkcja
1	Lista	Module Settings	Wyświetla listę zainstalowanych modułów
2	Moduł	Nazwa modułu	Udostępnia opcje do personalizacji ustawieni modułu i wyświetla status modułu
3	Lista	Options	Wyświetla listę ustawień systemu
4	Zakładka	App Instaler	Pozwala na instalacje i usuwanie modułów z systemu
5	Zakładka	Properties	Udostępnia opcje ustawień systemu
6	Zakładka	Night Mode	Udostępnia opcje ustawień trybu nocnego

Zarządzanie modułem

Widok po rozwinięciu ustawień modułu przedstawiono na rys. 16.



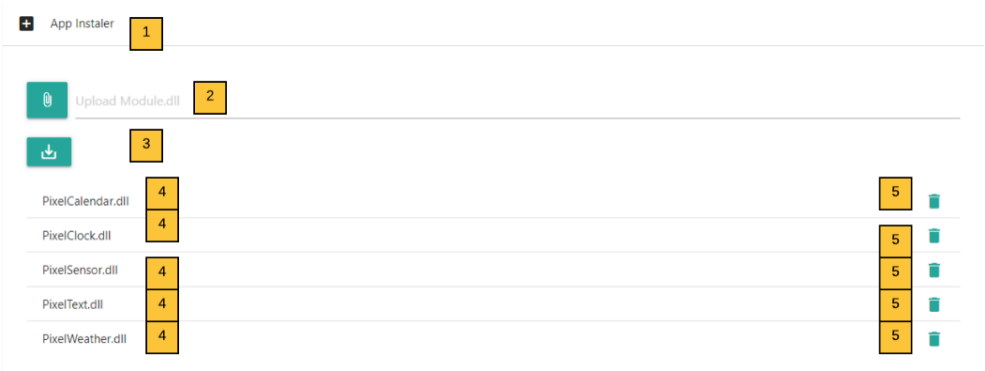
Rysunek 16 - Konfiguracja modułu (źródło: opracowanie własne)

Tabela 33 - Opis elementów konfiguracji modułu

L.p	Typ obiektu	Treść	Funkcja
1	Label	Nazwa modułu	Wyświetla nazwę modułu
2	Icon	Status modułu	Wyświetla status modułu
3	Switch	Module	Udostępnia opcje uruchomienia bądź zatrzymania modułu
4	Button, Label, TextBox, Radio, Switch	Nazwa opcji do ustawienia	Wyświetla opcje zdefiniowaną przez twórce modułu
5	Switch	Visible	Udostępnia opcje ukrycia modułu
6	TextBox	How long it should display in milisecond	Udostępnia opcje ustawienia czasu wyświetlania modułu
7	Button	Reload	Udostępnia opcje przeładowania modułu
8	Button	Select	Przełącza system na wybrany moduł

Instalacja/Usuwanie modułu z systemu

Widok zakładki usuwania modułów przedstawiono na rys. 17.



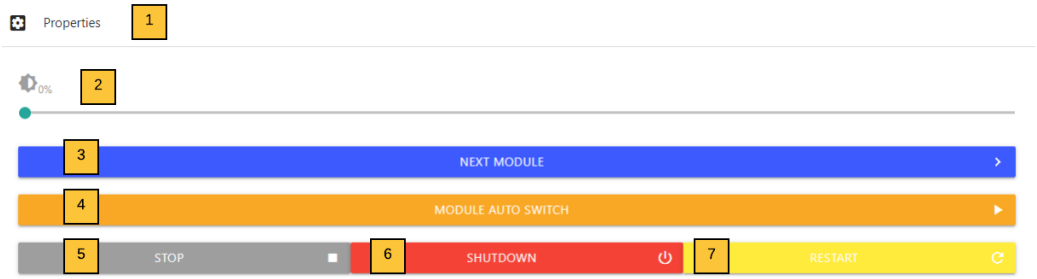
Rysunek 17 - Instalacja/usuwanie modułu (źródło: opracowanie własne)

Tabela 34 - *Opis elementów instalacji/usuwania modułu* (źródło: opracowanie własne)

L.p	Typ obiektu	Treść	Funkcja
1	Label	App Instaler	Wyświetla listę opcji
2	File Selector	Upload Module.dll	Pozwala wysłać plik modułu od urządzenia
3	Button	Submit	Zatwierdza plik do wysłania
4	Label	Nazwa zainstalowanego modułu	Wyświetla nazwę zainstalowanego modułu
5	Button	Delete	Udostępnia opcje usunięcia modułu z systemu

Konfiguracja systemu

Widok zakładki podstawowych ustawień systemu przedstawiono na rys. 18.



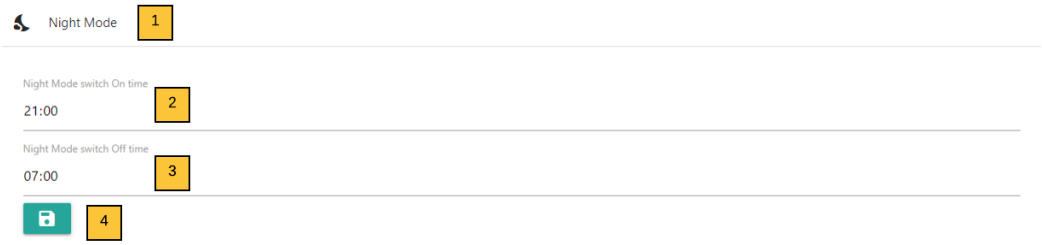
Rysunek 18 - Konfiguracja systemu (źródło: opracowanie własne)

Tabela 35 - *Opis elementów ustawień* (źródło: opracowanie własne)

L.p	Typ obiektu	Treść	Funkcja
1	Label	Properties	Wyświetla listę opcji
2	Range	Brightness	Pozwala ustawić jasność wyświetlacza
3	Button	Next Module	Przełącza na następny moduł
4	Button	Stop	Zatrzymuje aplikacje
5	Button	Shutdown	Wyłącza system
6	Button	Restart	Restartuje aplikacje

Tryb nocny

Widok zakładki ustawień trybu nocnego przedstawiono na rys. 19.



Rysunek 19 - *Ustawienia trybu nocnego* (źródło: opracowanie własne)

Tabela 36 - Opis elementów ustawienia trybu nocnego (źródło: opracowanie własne)

L.p	Typ obiektu	Treść	Funkcja
1	Label	Properties	Wyświetla listę opcji
2	Time	Night Mode enable on time	Pozwala ustawić godzinę włączenia trybu nocnego
3	Time	Night Mode disable on time	Pozwala ustawić godzinę wyłączenia trybu nocnego
4	Button	Save	Zapisuje ustawienia

5.7. Implementacja bazy danych

Skrypt tworzący bazę danych

Poniższy skrypt jest odpowiedzialny za wczytanie pliku konfiguracyjnego; a jeżeli plik nie istnieje - tworzy nowy pusty plik.

```
public Config()
{
        xml = new XmlDocument();
        try
        {
                xml.Load(file);
        }
        catch (Exception)
        {
                xml.LoadXml("<?xml version=\"1.0\" encoding=\"UTF8\"?>
                <config>
                        <properties>
                                <screen brightness = \"10\" />
                                <animatedSwitching      enabled=\"false\" /><nightmode enable =
                        \"False\"       />
                        </properties>
                        <modules></modules
                </config>");
                xml.Save(file);
        }
}
```

Skrypt dodający nowy moduł

Poniższy skrypt jest odpowiedzialny za pobranie wszystkich wartości z modułu i zapisanie ich do pliku konfiguracyjnego.

```
public void CreateModule(PixelModule module)
{
    XmlElement m = xml.CreateElement(string.Empty, "module", string.Empty);

    Logger.Log(ConsoleColor.Yellow, $"Module Name: {module.Name} Start: True");
    m.SetAttribute("class", module.Name);
    m.SetAttribute("start", "True");

    XmlElement p = xml.CreateElement(string.Empty, "params", string.Empty);
    foreach (var prop in module.GetType().GetProperties())
    {
        try
        {
            Logger.Log(ConsoleColor.Yellow, $"Prop: {prop.Name} Value:
{prop.GetValue(module).ToString()}");
            if (prop.Name == "Name" || prop.Name == "IsRunning")
            {
                continue;
            }
            else if (prop.PropertyType == typeof(Color))
            {
                var c = (Color)prop.GetValue(module);
                p.SetAttribute(prop.Name,
$"#{c.R.ToString("X2")}{c.G.ToString("X2")}{c.B.ToString("X2")}");
            }
            else
            {
                p.SetAttribute(prop.Name, prop.GetValue(module).ToString());
            }
        }
        catch (Exception)
        {
            Logger.Log(ConsoleColor.Yellow, $"Prop: {prop.Name} Value: ", ConsoleColor.Red, "NULL");
        }


    }

    m.AppendChild(p);
    xml.DocumentElement["modules"].AppendChild(m);
    xml.Save(file);
}
```

Skrypt do edytowania wartości modułu w bazie danych

Poniższy skrypt jest odpowiedzialny za zmianę stanu konfiguracji modułu w pliku konfiguracyjnym.

```
public void EditModules(PixelModule[] modules)
{
    //xml.DocumentElement["modules"].InnerText = "";
    foreach (var module in modules)
    {
        XmlElement m = null;
        foreach (XmlElement moduleElement in xml.DocumentElement["modules"])
        {
            if (moduleElement.GetAttribute("class") == module.Name)
                m = moduleElement;
        }
        if (m == null)
        {
            Logger.Log("Coś nie pykło");
            break;
        }
```

```
        m.SetAttribute("start", module.IsRunning.ToString());

        XmlElement p = m["params"];

        foreach (var prop in module.GetType().GetProperties())
        {
            if (prop.Name == "Name" || prop.Name == "IsRunning" || prop.Name == "Icon")
            {
                continue;
            }
            else if (prop.PropertyType == typeof(Color))
            {
                var c = (Color)prop.GetValue(module);
                p.SetAttribute(prop.Name,
$"#{c.R.ToString("X2")}{c.G.ToString("X2")}{c.B.ToString("X2")}");
            }
            else
            {
                p.SetAttribute(prop.Name, prop.GetValue(module).ToString());
            }
        }
        m.AppendChild(p);
        xml.DocumentElement["modules"].AppendChild(m);
    }
    xml.Save(file);
}
```

Skrypt pobierający dane modułu z bazy

Poniższy skrypt jest odpowiedzialny za pobranie danych konfiguracyjnych konkretnego modułu z pliku.

```
public Module GetModule(string name)
{
    foreach (XmlElement module in xml.DocumentElement["modules"])
    {
        if (module.GetAttribute("class") == name)
            return new Module(module);
    }
    return null;
}
```

Skrypt usuwający moduł z bazy danych

Poniższy skrypt jest odpowiedzialny za usunięcie konfiguracji podanego modułu.

```
public void RemoveModule(string name)
{
    XmlElement tmp = null;
    foreach (XmlElement module in xml.DocumentElement["modules"])
    {
        if (module.GetAttribute("class") == name)
            tmp = module;
    }
    xml.DocumentElement["modules"].RemoveChild(tmp);
    xml.Save(file);
}
```

Skrypt sortujący moduły w bazie

Poniższy skrypt jest odpowiedzialny za posortowanie modułów, tak aby na liście w panelu wyświetlały się w kolejności określonej przez użytkownika.

```
public void SortModules(string[] names)
{
    List<XmlElement> tmp = new List<XmlElement>();
    foreach (XmlElement module in xml.DocumentElement["modules"])
    {
        tmp.Add(module);
    }
    xml.DocumentElement["modules"].InnerText = "";
```

```
    foreach (string name in names)
    {
        xml.DocumentElement["modules"].AppendChild(tmp.Find((x) =>
        x.GetAttribute("class") == name));
    }
    xml.Save(file);
}
```

Skrypt pobierający listę modułów z bazy danych

Poniższy skrypt zwraca tablice konfiguracji wszystkich modułów zainstalowanych w systemie.

```
public Module[] Modules
{
    get
    {
        var modules = new List<Module>();
        foreach (XmlElement module in xml.DocumentElement["modules"])
        {
            modules.Add(new Module(module));
        }
        return modules.ToArray();
    }
}
```

Skrypt pobierający kolejność modułów z bazy danych

Poniższy skrypt zwraca tablice nazw modułów w kolejności jakiej posortował je użytkownik.

```
public string[] ModuleOrder
{
    get
    {
        List<string> tmp = new List<string>();
        foreach (XmlElement module in xml.DocumentElement["modules"])
        {
            tmp.Add(module.GetAttribute("class"));
        }
        return tmp.ToArray();
    }
}
```

Skrypt pobierający/zmieniający wartość jasności wyświetlacza z bazy danych

Poniższy skrypt odpowiada za zapisanie i pobranie danych jasności wyświetlacza z pliku.

```
public byte Brightness
{
    get => byte.Parse(xml.DocumentElement["properties"]["screen"].GetAttribute("brightness"));
    set
    {
        xml.DocumentElement["properties"]["screen"].SetAttribute("brightness", value.ToString());
        xml.Save(file);
    }
}
```

5.8. Implementacja logiki działania

Zestawienie zaimplementowanych klas i interfejsów systemu przedstawiono w tabeli 37.

Tabela 37 - Zestawienie zaimplementowanych klas i interfejsów (źródło: opracowanie własne)

L.p	Nazwa klasy	Projekt	Implementacja
1	Program	+	+
2	Logger	+	+
3	HttpServer	+	+
4	POST_File	+	+
5	Config	+	+
6	LoadingAnimation	+	+
7	GifImage	+	+
8	PostHandler	+	+
9	IPixelRenderer	+	+
10	PixelRenderer	+	+
11	IPixelDraw	+	+
12	PixelDraw	+	+
13	IGPIO	+	+*
14	GPIO	+	+*
15	Bluetooth	+	+*
16	PixelModule	+	+
17	NullModule	+	+
18	SettignModule	+	+

UWAGA! Symbol +* oznacza, że klasa została częściowo zaimplementowana.

Zestawienie deklaracji interfejsów przedstawiono w tabeli 38.

Tabela 38 - Zestawienie deklaracji interfejsów (źródło: opracowanie własne)

L.p	Nazwa metody/właściwości	Typ zwracany	Projekt	Implementacja
IPixelRenderer				
1	Current	PixelModule	+	+
2	GetModules	PixelModule[]	+	+
3	IsReady	bool	+	+
4	IsRunning	bool	+	+
5	Pause	bool	+	+
6	GetModule	PixelModule	+	+
7	LoadModule	void	+	+
8	NextModule	void	+	+
9	Reload	void	+	+
10	Start	void	+	+
11	Stop	void	+	+
12	SwitcxhModule	bool	+	+
13	UpdateConfig	void	+	+
IPixelDraw				
1	Brightness	byte	+	+
2	Clear	void	+	+
3	Draw	void	+	+

4	GetScreen	Image	+	+
5	SetImage	void	+	+
6	SetPixel	void	+	+
7	SetText	int	+	+
8	SetTextRoman	void	+	+
9	TextLength	int	+	+
IGPIO				
1	EnableBuzzer	void	+	+
2	OnButtonClick	ButtonEventArgs	+	+

Zestawienie zaimplementowanych metod i właściwości przedstawiono w tabeli 39.

Tabela 39 - Zestawienie zaimplementowanych metod i właściwości (źródło:opracowanie własne)

L.p	Nazwa metody/właściwości	Typ zwracany	Projekt	Implementacja
Logger				
1	Clear	void	+	+
2	Log	void	+	+
HttpServer				
1	IsReady	bool	+	+
2	IsRunning	bool	+	+
3	PostResponse	dynamic	+	+
4	Error	void	+	+
5	HandleIncomingConnections	Task	+	+
6	Post	void	+	+
7	Start	void	+	+
8	Stop	void	+	+
POST_File				
1	GetBoundry	string	+	+
2	IndexOf	int	+	+
3	SaveFile	void	+	+
Config				
1	AnimatedSwitching	bool	+	+
2	Brightness	byte	+	+
3	ModuleOrder	String[]	+	+
4	Modules	Module[]	+	+
5	CreateModule	void	+	+
6	EditModules	void	+	+
7	GetModule	Module	+	+
8	RemoveModule	void	+	+
9	SortModule	void	+	+
LoadingAnimation				
1	Run	void	+	+
2	Stop	void	+	+
PostHandler				
1	WebServerOnError	void	+	+*
2	WebServerOnPost	void	+	+*
PixelRenderer				
1	AnimatedSwitching	bool	+	+

2	Current	PixelModule	+	+
3	GetModules	PixelModule[]	+	+
4	IsReady	bool	+	+
5	IsRunning	bool	+	+
6	Pause	bool	+	+
7	Pixel	PixelRenderer	+	+
8	GetModule	PixelModule	+	+
9	GPIOevents_OnButtonClick	void	+	+
10	LoadModule	void	+	+
11	NextModule	void	+	+
12	Reload	void	+	+
13	Render	void	+	+*
14	Start	void	+	+
15	Stop	void	+	+
16	SwitchModule	bool	+	+
17	UpdateConfig	void	+	+
PixelDraw				
1	Brightness	byte	+	+
2	Screen	IPixelDraw	+	+
3	Clear	void	+	+
4	Draw	void	+	+
5	GetScreen	Image	+	+
6	LoadFont	void	+	+
7	LoadImage	Color[]	+	+
8	LoadRomanFont	void	+	+
9	MovePixels	void	+	+*
10	SetImage	void	+	+
11	SetPixel	void	+	+
12	SetText	void	+	+
13	SetTextRoman	void	+	+
14	TextLength	int	+	+
GPIO				
1	GPIOevents	IGPIO	+	+
2	EnableBuzzer	void	+	+
Bluetooth				
1	Init	void	+	+*
PixelModule				
1	Screen	IPixelDraw	+	+
2	GPIOevents	IGPIO	+	+
3	pixelRenderer	IPixelRenderer	+	+
4	SetScreen	void	+	+
5	SetGPIO	void	+	+
6	SetRenderer	void	+	+
7	Name	string	+	+
8	Icon	string	+	+
9	Visible	bool	+	+
10	Timer	int	+	+
11	Tickrate	int	+	+
12	IsRunning	bool	+	+

13	OnButtonClick	void	+	+
14	Start	void	+	+
15	Stop	void	+	+
16	Reload	void	+	+
17	Update	void	+	+
18	Draw	void	+	+
NullModule				
1	Draw	void	+	+
2	Update	void	+	+
SettingModule				
1	Draw	void	+	+
2	Update	void	+	+
3	OnButtonClick	void	+	+

UWAGA! Ze względu na obszerność tematu nie wszystkie metody zostały jeszcze zaimplementowane. Metody częściowo zaimplementowane oznaczone są symbolem +*.

5.9. Tworzenie nowego modułu

Aby stworzyć nowy moduł należało utworzyć nowy projekt - Biblioteka klas (.NET Framework), zaimportować *PixelModule.dll,* a następnie utworzyć publiczną klasę dziedziczącą po *SharpClock.PixelModule*. Każda publiczna właściwość klasy została wykorzystana podczas generowania formularza w panelu sterowania modułem. W konstruktorze należało ustawić zmienną *Tickrate*, która określa interwał pomiędzy wykonaniem metody *Update(),* ustawić ikonę modułu, załadować obrazki, które moduł będzie wyświetlał. W metodzie *Update()* zaimplementowano logikę modułu, np: pobranie aktualnego czasu systemowego, pobieranie danych z internetu, aktualizacji danych do wyświetlenia. W metodzie *Draw()*zaimplementowano tylko instrukcje odpowiedzialne za wyświetlanie wcześniej zainicializowanych zmiennych. Implementacja instrukcji wymagających więcej czasu na wykonanie wpływa negatywnie na działanie systemu, ponieważ metoda Draw() wykonywana jest z częstotliwością 30Hz[15]. Moduł może korzystać z trzech przycisków zainstalowanych na górze obudowy. Implementacja odbywała się poprzez nadpisanie metody *OnButtonClick().* Pierwsze dwa przyciski są zarezerwowane dla pauzowania przewijania i przełączania na następny moduł.

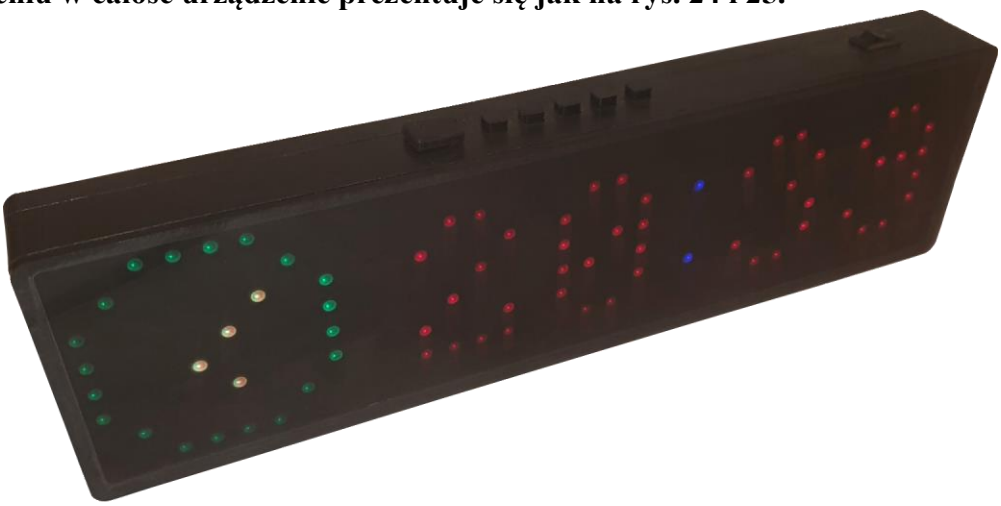
```
public class PixelNewModule : PixelModule
{
    public PixelNewModule()
    {
      Konstruktor - miejsce na inicjalizacje modułu
    }
    protected override void Update(Stopwatch stopwatch)
    {
```

[15] Herc - jednostka miary częstotliwości w układzie SI. Definiuje się ją jako liczbę cykli na sekundę

```
        Metoda Update wykonywana jest w pętli co 1000 milisekund, wartość można
        zmienić poprzez edycje zmiennej Tickrate
    }
    public override void Draw(Stopwatch stopwatch = null)
    {
        Metoda Draw wykonywana jest w pętli, system stara się utrzymać
        rysowanie na poziomie 30 razy na sekunde.
        Do rysowania na ekranie służą metody:
            ● Screen.SetText();
            ● Screen.SetPixel();
            ● Screen.SetImage();
    }
    public override void OnButtonClick(ButtonId button)
    {
        base.OnButtonClick(button);
        if(button == ButtonId.User1)
        {
            Implementacja kodu wykonywanego po naciśnięciu 3 przycisku
        }
        else if(button == ButtonId.User2)
        {
            Implementacja kodu wykonywanego po naciśnięciu 4 przycisku
        }
        else if(button == ButtonId.User3)
        {
            mplementacja kodu wykonywanego po naciśnięciu 5 przycisku
        }
    }
}
```

6. Uruchomienie i testowanie

Po złożeniu w całość urządzenie prezentuje się jak na rys. 24 i 25.

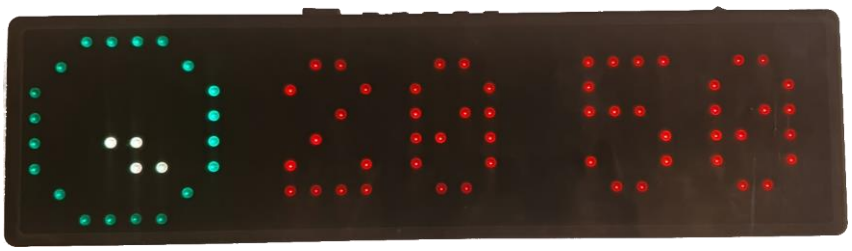


Obraz 24 - Urządzenie złożone w całość (źródło: opracowanie własne)

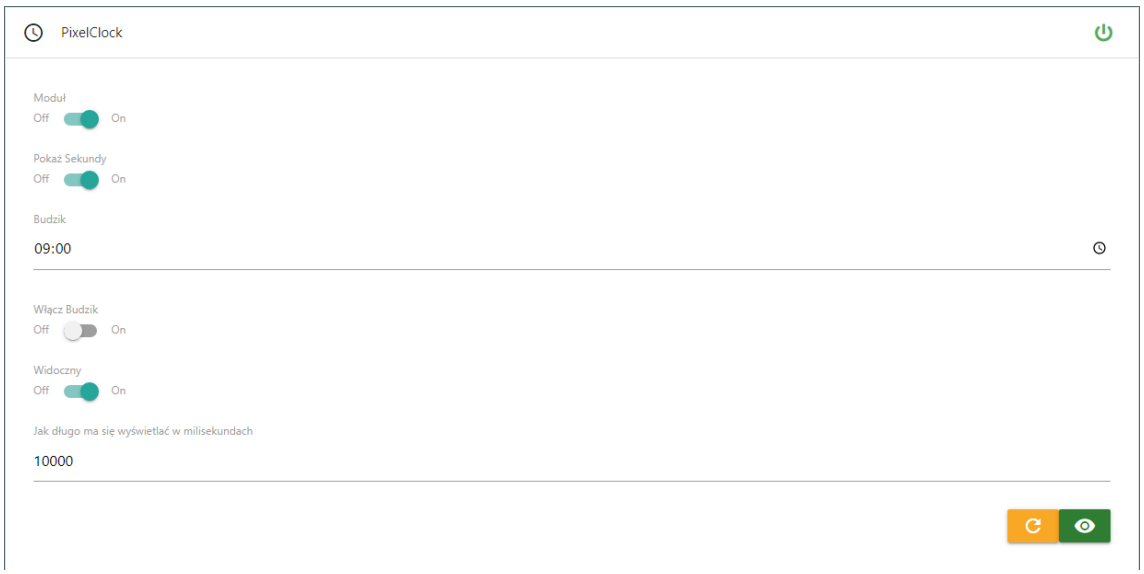
Obraz 25 - Urządzenie złożone w całość - wnętrze (źródło: opracowanie własne)

Na urządzeniu zainstalowano moduły do wyświetlania:

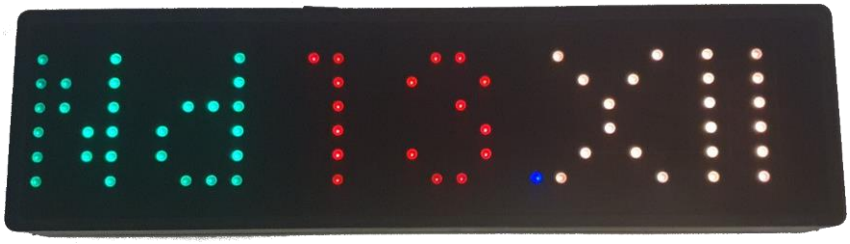
- Godziny,
- Daty,
- Pogody,
- Tekstu wprowadzonego przez użytkownika,
- Odczytu wartości temperatur z czujnika.



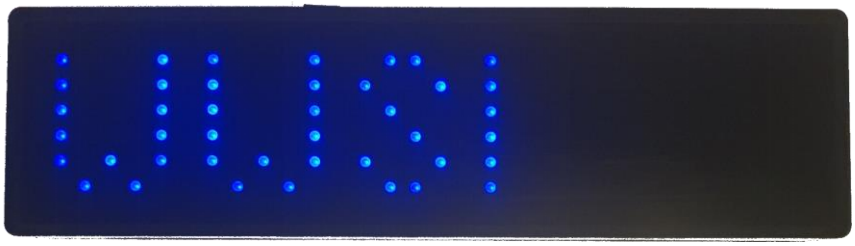
Obraz 26 - Moduł zegara (źródło: opracowanie własne)



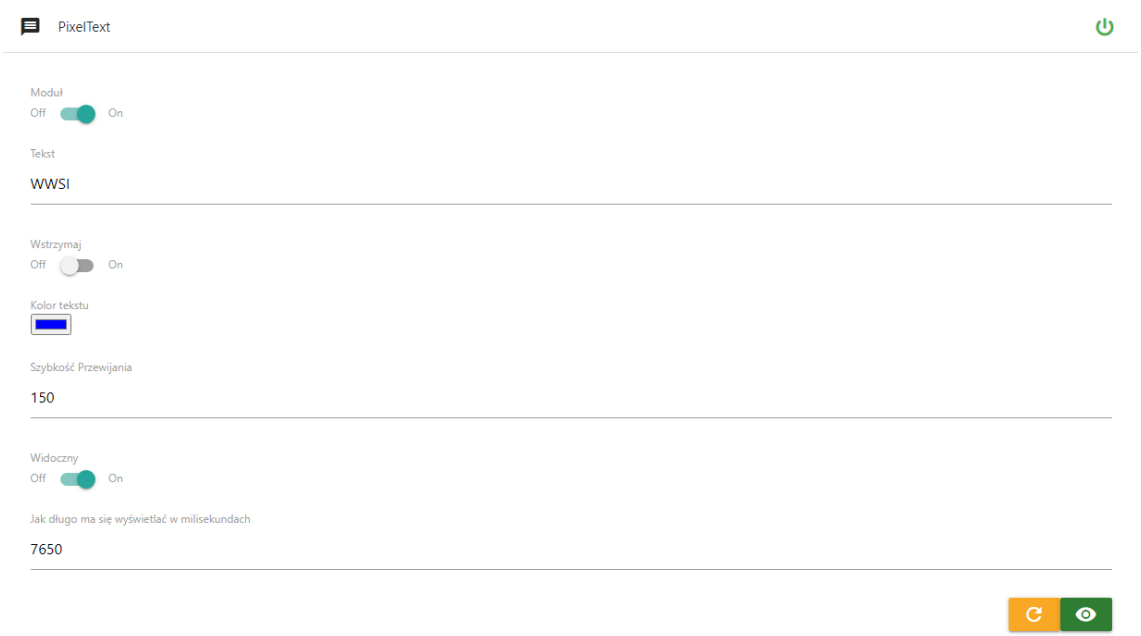
Obraz 27 - Sterowanie modułem zegara



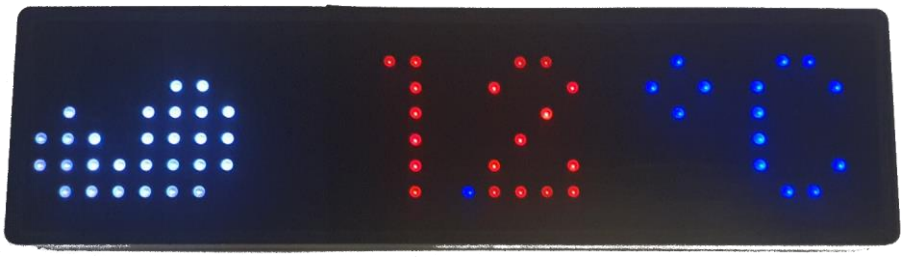
Obraz 28 - Moduł kalendarza (źródło: opracowanie własne)



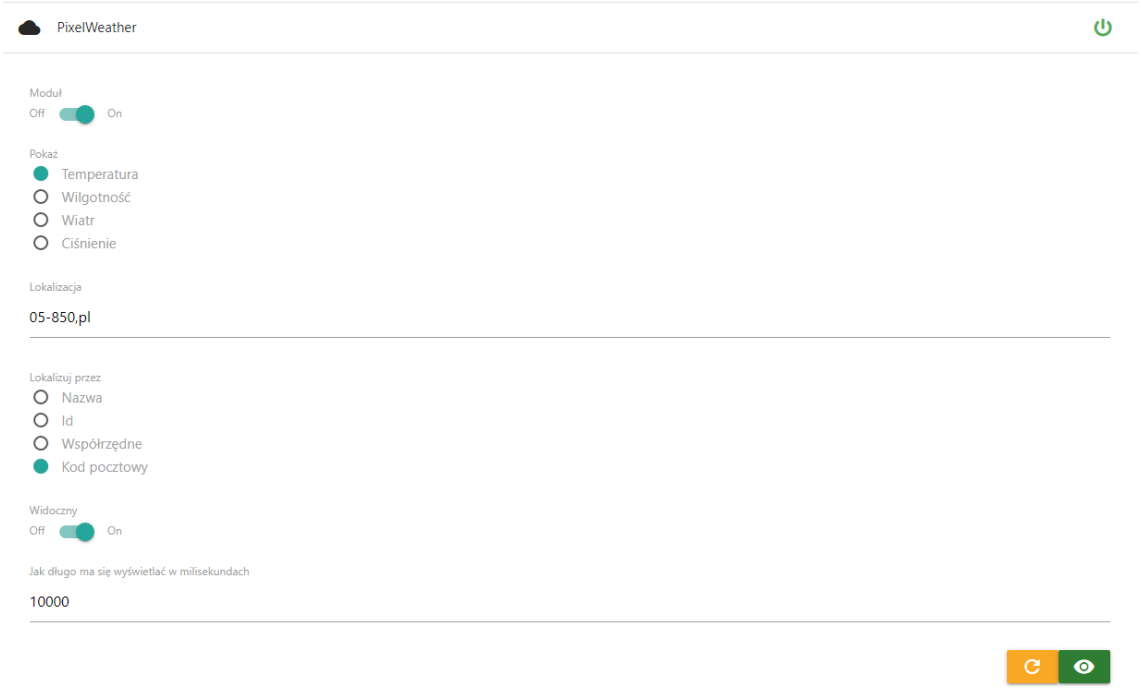
Obraz 29 - Moduł wyświetlania tekstu (źródło: opracowanie własne)



Obraz 30 - Sterownie modułem wyświetlania tekstu (źródło: opracowanie własne)



Obraz 31 - Moduł pogodowy (źródło: opracowanie własne)



Obraz 32 - Sterowanie modułem pogodowym (źródło: opracowanie własne)

7. Podsumowanie i wnioski

Celem pracy było zaprojektowanie i wdrożenie inteligentnego zegarka w oparciu o mikro-kontroler Raspberry Pi Zero. W pierwszej części opisano zarys historii powstania kontrolera, na którym bazuje cały system.

Opracowany projekt spełnił założenia i wymagania postawione w tej pracy. Jedyną funkcjonalnością, której nie udało się w pełni zaimplementować jest tryb nocny i obsługa Bluetooth.

System wzbogacono o aplikację webową, która umożliwia zarządzanie i konfigurację z poziomu przeglądarki internetowej. Dzięki temu rozwiązaniu gotowy projekt może być wykorzystywany przez przeciętnego użytkownika bez znajomości języków programowania. Prosty interfejs i minimalizm aplikacji nie przytłacza użytkownika. Pozwala on w szybki sposób na skonfigurowanie urządzenia i dostęp do najważniejszych funkcjonalności.

Największą trudnością w trakcie realizacji niniejszej pracy okazało się wykonanie obudowy. W pierwszej wersji urządzenia obudowę zbudowano z drewnianej sklejki, do której przymocowano wszystkie elementy. Jednak taka konstrukcja okazała się narażona na uszkodzenia delikatnych podzespołów. W rezultacie zaprojektowano obudowę tak, aby całe urządzenie było kompaktowe i estetyczne, a cała elektronika była bezpiecznie ukryta wewnątrz. Obróbka CNC umożliwiła precyzyjne docięcie części tak aby całość była idealnie spasowana.

Aby utworzony system wykorzystać w celach komercyjnych, powinien on zostać poddany kilku udoskonaleniom. M.in. płytki z elektroniką należałoby wykonać maszynowo, natomiast obudowa powinna być tłoczona z formy, a nie frezowana z bloku.

8. Literatura i źródła

1. Jon Skeet (2012) C# od podszewki. Wydanie II
2. Akkana Peck (2019) Raspberry Pi Zero W. Kontrolery, czujniki, sterowniki i gadżety
3. Brian Ward (2015) Jak działa Linux. Podręcznik administratora. Wydanie II
4. https://lametric.com/
5. https://wikitia.com/wiki/LaMetric
6. https://divoom.pl/
7. https://botland.com.pl/
8. https://docs.microsoft.com/pl-pl/dotnet/
9. https://www.raspberrypi.org/documentation/
10. https://github.com/klemmchr/rpi_ws281x.Net
11. https://github.com/unosquare/raspberryio
12. https://www.newtonsoft.com/json/help/html/Introduction.htm
13. https://www.w3schools.com/
14. https://materializecss.com/
15. https://jquery.com/