

Dokumentacja do projektu z przedmiotu:

Technologie Bezprzewodowe w Automatyce i Robotyce

**Sterownik BlueTooth
do Lampy LED
KwadraLampa**



LP1 C1, EADI-3
Bełch Jakub

Rzeszów, IV 2017

1. Cel i zakres projektu

Projekt miał na celu wykonanie sterownika do pokojowej stojącej Lampy LED, można go również użyć do innych zastosowań. Zakres obejmował wykonanie firmwaru do mikrokontrolera oraz klienta na Android i PC. Z braku czasu na tę chwilę zrezygnowałem z wykonania klienta na PC.

Podstawowe założenia:

- włączanie/wyłączanie lamp
- regulacja jasności
- włączanie/wyłączanie po zadany czasie – `TIMER`
- włączanie/wyłączanie o zadanej godzinie – `CLOCK`

Dodatkowo w trybach `CLOCK` i `TIMER` można ustawić opcję ze zamiast nagłego wyłączenia po zadany czasie lampa ściemnia/rozjasnia się stopniowo do 0/255 w ciągu tego czasu.

Na razie w trybach *timer* i *clock* działa tylko wyłączanie, muszę pomyśleć jak zrealizować to rozróżnienie

Komunikacja bezprzewodowa realizowana jest przez dowolny moduł BT w trybie SPP

2. Ogólny opis rozwiązania

Zadanie było w miarę proste jednak wymagało trochę czasu żeby wszystko działało wygodnie. sterowanie realizowane jest w dwóch trybach `MANUAL` i `BLUTUTU`. Obsługa obu lamp jest identyczna.

2.1. tryb Manual:

Wykorzystałem dostępny fizyczny interfejs tzn. dwie galki, z braku potencjometrów z wyłącznikiem, wyłączanie polega na skreśleniu potencjometrów na 0, regulacja polega na kręceniu galkami. Lampa ma być prosta intuicyjna w obsłudze i wygodna dla postronnych osób dlatego nie mnożyłem dalszych kontrolek

2.2. tryb Bluetooth:

Realizowany jest na zasadzie przesyłania kilkunastu komunikatów sterujących płytka. Dostępne opcje:

- gaszenie/zapalenie przyciskiem
- regulacja jasności
- włączanie/wyłączanie po zadany czasie – `timer`
- włączanie/wyłączanie o zadanej godzinie – `clock`

Dodatkowo w trybach `clock` i `timer` można ustawić opcję ze zamiast nagłego wyłączenia po zadany czasie lampa ściemnia się stopniowo do 0 w ciągu tego czasu (nieliniowo ponieważ jasność diody nie maleje liniowo wraz z prądem).

Przed podłączeniem konieczne jest sparowanie urządzenia z telefonem

Aby móc korzystać z trybu `clock` konieczna jest znajomość rzeczywistego czasu, ponieważ nie jest to jakos szczególnie ważne zrezygnowałem z jakiegos modułu podtrzymywania/pobierania go, a rozwiązałem to w ten sposób że po połączeniu Klientem na którym mamy aktualny czas jest on przesyłany i uaktualniany w sterowniku,

Czas w sterowniku przekreśla się co ok 40 dni, i kasuje po zaniku zasilania, jednak przy sterowaniu ręcznym i tak jest zbędny a po połączeniu z BT jest ustawiany więc ten problem uznałem za rozwiązany

Komunikaty sterujące:

3. Komunikaty do sterownika

Długość komunikatu 11 znaków (ważne! aczkolwiek firmware dopełnia sam do 11 znaków jeśli dostał krótszy komunikat, jeśli dostał dłuższy, sprawdza ostatnie 11 znaków)

Liczby lepiej byłoby przesyłać jako bajty (krociej) ale potem trzeba by konwertować na stringi, na etapie prototypowania prościej wysyłać jako stringi/char.

Tak samo tryby i w ogóle komunikaty, docelowo lepiej przerobić na typ ENUM i wtedy wysyłamy jako jeden bajt cały string – ale trzeba by wtedy pamiętać w terminalu te "kody" trybów

Spis komunikatów:

Zapytania do sterownika:	
::HM:?	Przesłanie aktualnego czasu ze sterownika do klienta
::L1:?	Odsyła stan lampy do klienta
::L1:DIM?	Odsyła stan opcji ściemniania do klienta
::L1:BRGHT?	Odsyła aktualną jasność lampy
Ustawienia sterownika:	
::HM: <i>hhmm</i>	Przesłanie aktualnego czasu do sterownika
::L1: on ::L1: off	Wyl/zal lampy 1
::L1: PWM <i>5</i> ::L1: PWM <i>55</i> ::L1: PWM <i>255</i>	Przesłanie zadanej jasności
::L1: DIM on ::L1: DIM off	Wyl/zal ściemnianie opcji ściemniania dla trybu timer i clock
::L1: CL <i>hhmm</i> ::L1: CL <i>hhmm</i>	Wyl/zal lampy o zadanej godzinie
::L1: TIM <i>5</i> ::L1: TIM <i>55</i> ::L1: TIM <i>255</i>	Wyl/zal lampy po zadany czasie
::L1: TI cnc l	Anulowanie timera
::L1: CL cnc l	Anulowanie clock
::L1: TGL on	Po zadany czasie/godzinie lampa się włącza
::L1: TGL off	Po zadany czasie/godzinie lampa się wyłącza

Analogicznie dla lampy L2, Szczegółowy opis firmware – dalej

4. Sprzęt

4.1. Lampa

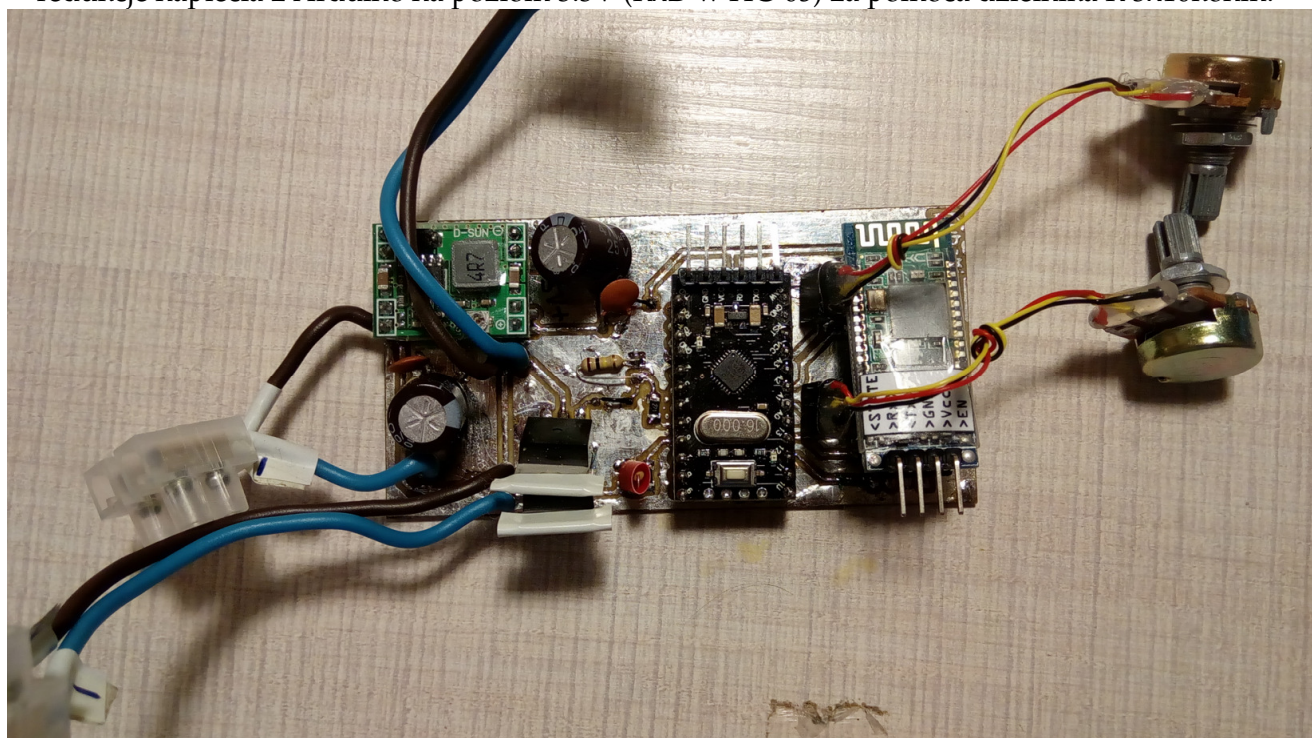
Lampa składa się z dwóch modułów LED 20W + 4W, sterowanie każdą lampą osobno, wyłączenie lampy realizowane jest przez skreślenie jasności do 0, lub ustawienie jej programowo na 0 (zatkanie tranzystora MOSFET)



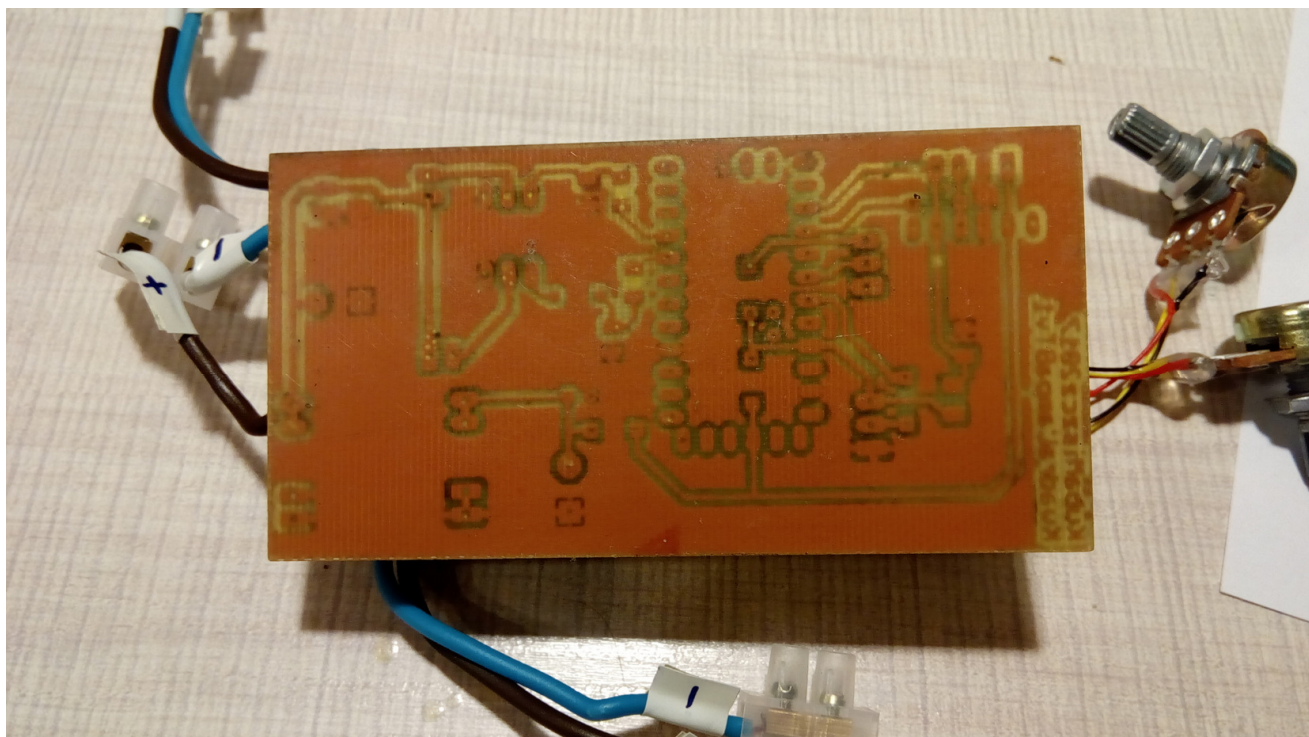
4.2. Płytki PCB

W związku z zasilaniem 20V i maksymalnym prądem ok 1A konieczne było zastosowanie tranzystora sterowanego PWM (0-255 ok. 1kHz) jako regulatora, wybór padł na MOSFET IRFZ44N gdyż jest w miarę uniwersalny, miałem go pod ręką, dla takich parametrów prawie się nie grzeje (nie trzeba radiatora), i pobiera dużo mniej mocy niż analogiczny bipolarny.

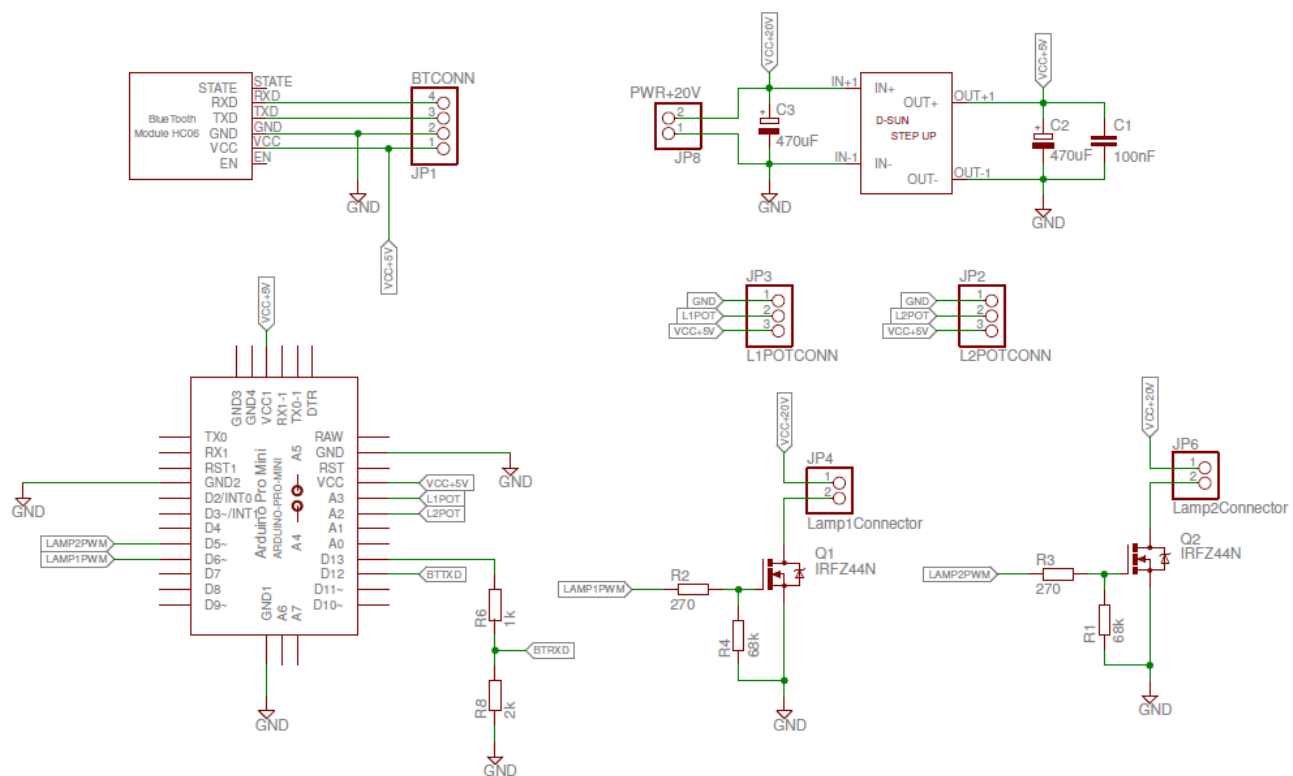
Logikę zrealizowałem za pomocą mikrokontrolera Atmega8A z bootloaderem Arduino, redukcję napięcia z Arduino na poziom 3.3V (RxD w HC-05) za pomocą dzielnika R 3x10kohm.



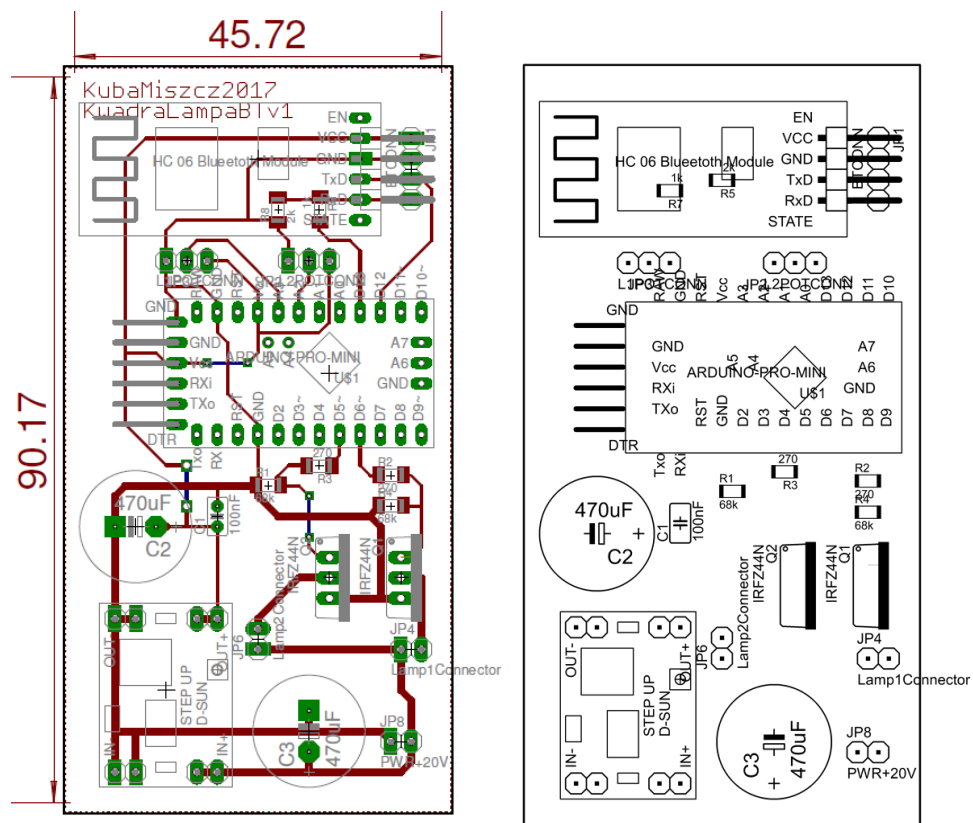
widok zmontowanej płytki



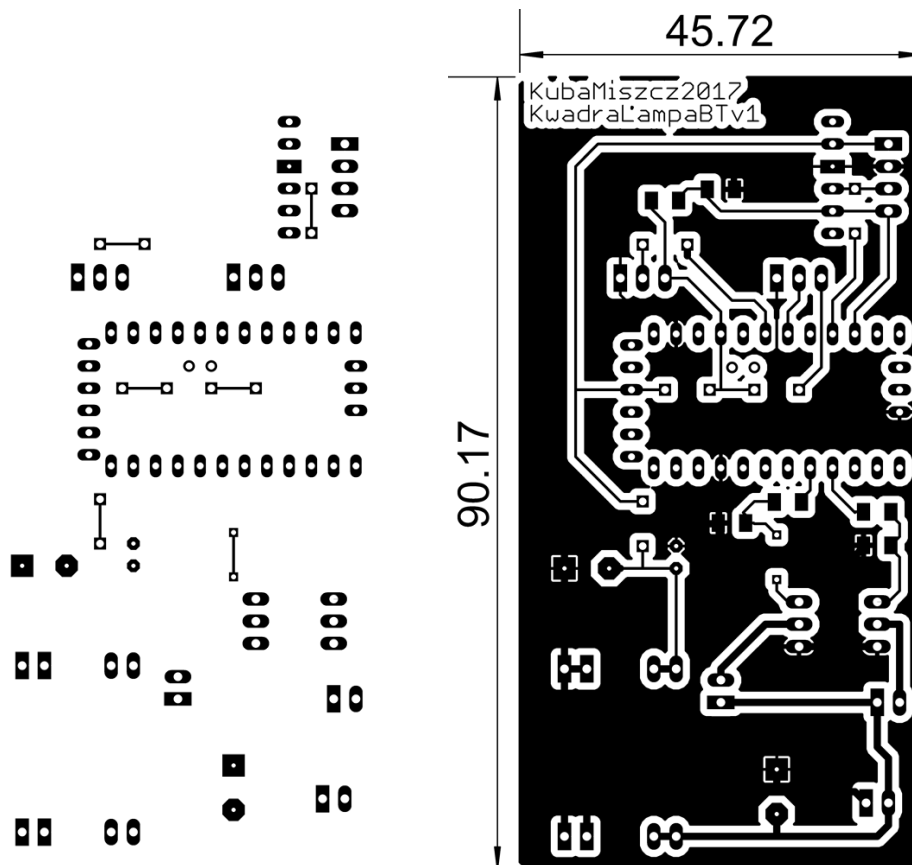
widok od spodu



schemat ideowy



widok elementów



widok warstw TOP i BOTTOM

warstwa TOP nie była drukowana, a mostki zostały dolutowane na BOTTOM

HC-05 można wypinąć w celu ewentualnej rekonfiguracji, podobnie można podpiąć się do pinów arduino w celu aktualizacji firmware

Ponieważ wiercenie jest uciążliwe elementy przewlekane również zrealizowałem jako SMD, nie pozwala to na druk etykiet od strony elementów, jednak sprawę rozwiązuje dołożenie kartki z opisem elementów na płytce (w razie poprawek za dłuższy czas)

4.3. Klienci

- Android – telefon z androidem, apka w AppInventor

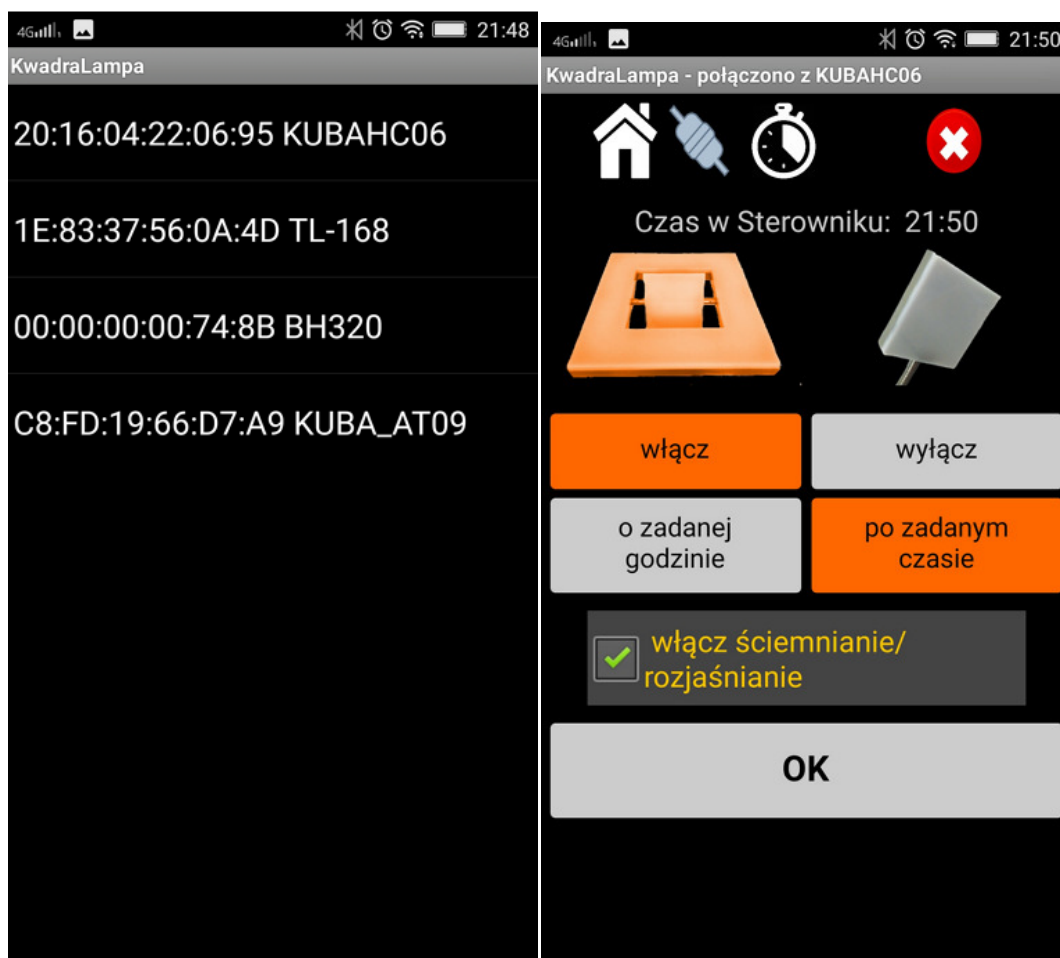


ekran główny

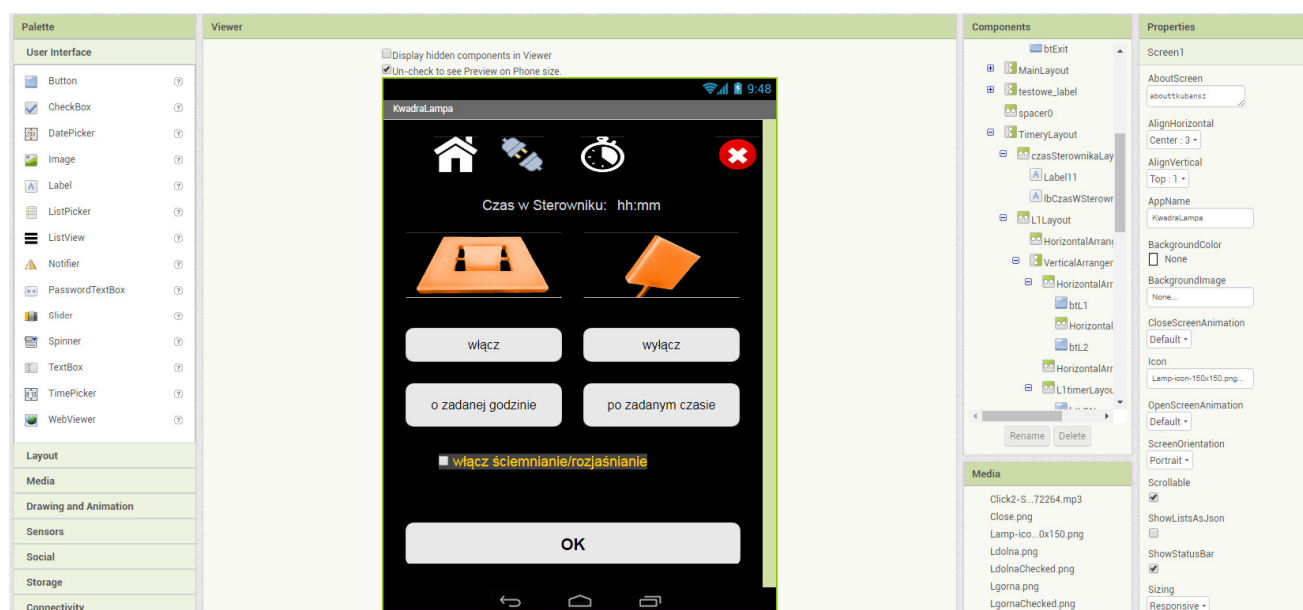
Ponieważ (najprawdopodobniej) w AppInventor jest na sztywno zapisany interwał między wysłaniem komunikatów, konieczne było zrobienie sobie niybufoora, który przy szybkim klikaniu dopisuje je do listy stringów, i wysyła co 1sek usuwając z listy, zapobiega to gubieniu komunikatów, lub wysyłaniu długiej listy, która trzeba by dzielić w sterowniku, co nie byłoby trudne do zrobienia jednak głównym powodem było to że w przypadku regulacji jasności sposób ten byłby niedobry gdyż przy zmianie o 100 jednostek trzeba byłoby czekać 100sek, gdybyśmy brali ostatnią wartość gubilibyśmy inne komunikaty przy szybkim klikaniu, aktualnie sterowanie jasnością polega na wysyłaniu ostatniego komunikatu, natomiast pozostałe komunikaty wysyłane są przez bufor.

Layoutu i jego parametrów nie będę opisywał, gdyż są to proste ustawienia kolorów, czcionek, rozmieszczenia elementów w kontenerach itp, tym bardziej że AppInventor ma bardzo ubogi i uciążliwy tryb edycji tych elementów, i w celu dalszego rozwijania klienta zdecydowałem się przepisać go od nowa w Xamarin/C#.

Podobnie obiekt `BTClient` istnieje tylko w jednym ekranie i trzeba przesyłać dane (nie ma do niego dostępu z innej activity/ekranu), rozwiązałem to zmieniając atrybut `Visible` poszczególnych layoutów (wszystkie są tak naprawdę na jednym ekranie)



lista wyboru nadajników BT i obsługa timerów



przykładowe okno edytora

4.4. Obsługa Klienta

Przed użyciem trzeba sparować urządzenie w ustawieniach BT telefonu.

Po kliknięciu na ikonkę z wtyczka pojawi się lista sparowanych klientów, po kliknięciu nastąpi próba połączenia i powrót do ekranu głównego.

Po połączeniu można operować kontrolkami, regulacja jasności aktualizuje jasność w sterowniku jednak lampa włącza się dopiero po jej włączeniu włącznikiem, podobnie efekt zmiany występuje tylko gdy lampa jest włączona.

Przy przekroczeniu galaka sterownik przechodzi w tryb ręczny i aktualizuje jasność dla ustawionej na lampie a nie telefonie.

w trybie `timer` należy po kolei wybrać lampę, czy ma się włączyć/wyłączyć, w jakim trybie (jednocześnie otwiera się kontrolka wyboru czasu), oraz ewentualny tryb płynnej jasności, i zatwierdzić OK.

Ponieważ jasność może zmieniać się co najmniej o 1 (int), więc aby zmieniała się jak najpłynniej ten interwał czasowy wyliczany jest na podstawie nastawionego czasu i aktualnej jasności, mimo to czasem zdarza się że sterownik przegapi kilka kroków (jak się okazuje 16MHz to nie tak szybko), dlatego pilnuje tego i czasem w razie potrzeby zmienia jasność o kilka kroków (szczegóły w listingu firmware). Na koniec czasu jednak osiąga zadana wartość.

Zmiany na ekranie głównym i sterowanie ręcznie anulują timery.

Za pomocą wbudowanych komunikatów można wykonać w zasadzie wszystkie operacje z lampkami dlatego w przyszłości wymagana może być tylko aktualizacja klientów bez modyfikacji i demontażu płytki.

cały program



obsługa BT i bufor wysyłający i opróżniający listę o wysłany element

```

to SendClock
do
  set global Hours to call recvTimer .Hour
  instant call recvTimer .Now
  set global Minutes to call recvTimer .Minute
  instant call recvTimer .Now
  set Label3 . Text to join " :HM: "
  get global Hours
  get global Minutes
  add items to list list
  item join " :HM: "
  get global Hours
  get global Minutes

```

```

to sendtoBT str
do
  if BluetoothClient1 . IsConnected
  then
    set Label3 . Text to get str
    call BluetoothClient1 .SendText
    text get str

```

```

to rcvfromBT2
do
  if BluetoothClient1 . IsConnected
  then
    if call BluetoothClient1 .BytesAvailableToReceive > 0
    then
      set global rcvMsg to call BluetoothClient1 .ReceiveText
      numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
      set Label12 . Text to get global rcvMsg

```

```

initialize global L1Brightness to 0
initialize global L1prevHead to 90
initialize global L1KnobLocked to false
initialize global L1timer to 0

```

```

initialize global L2Brightness to 0
initialize global L2prevHead to 90
initialize global L2KnobLocked to false
initialize global L2timer to 0

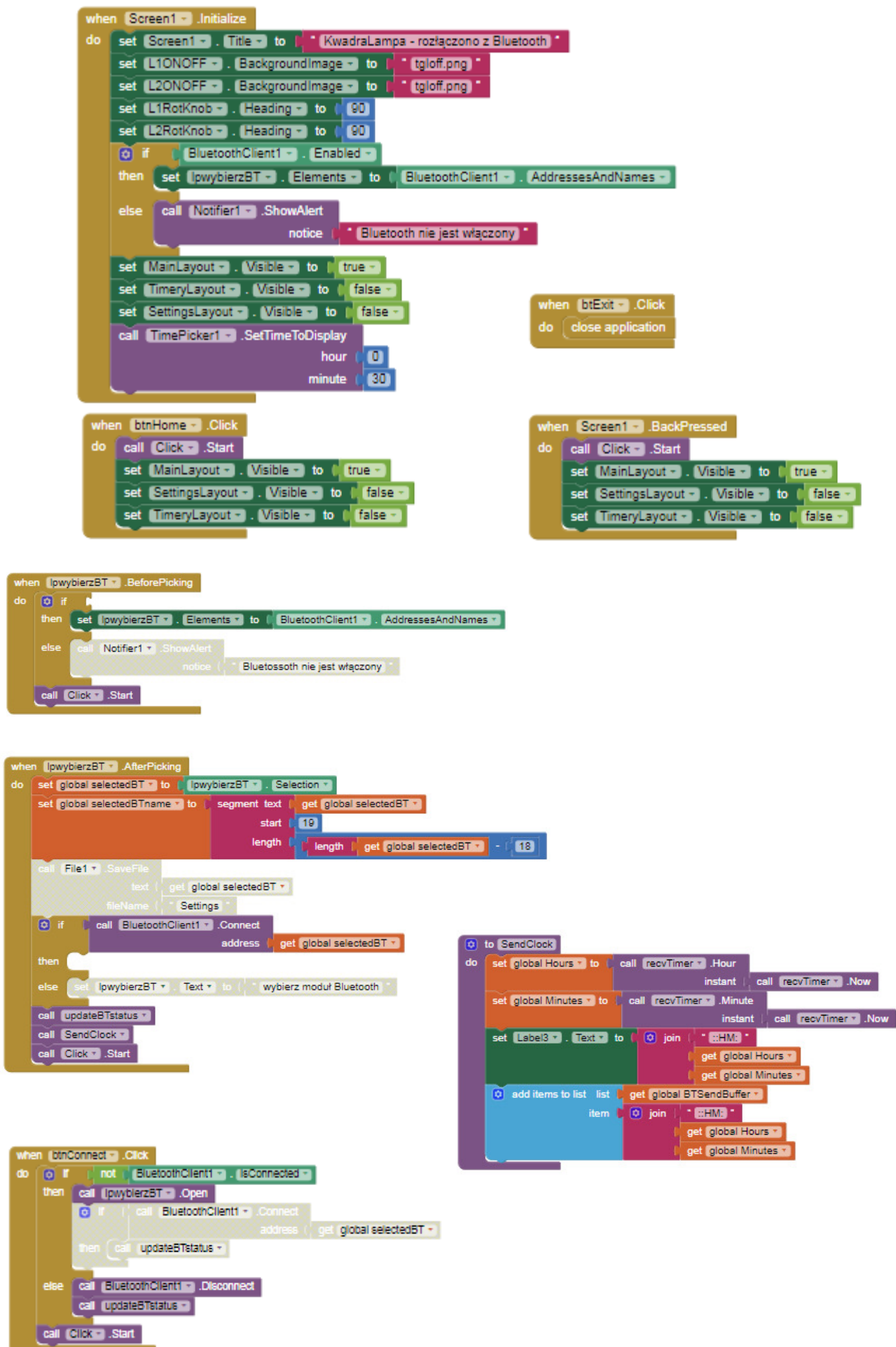
```

```

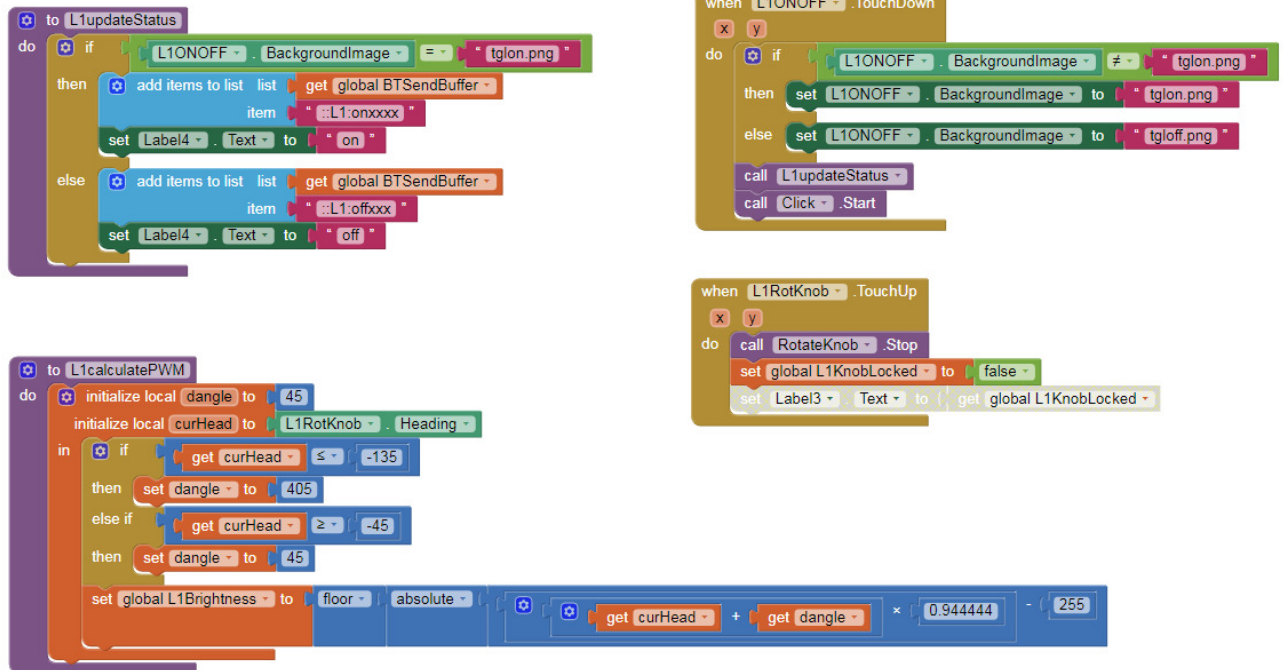
initialize global BToSendBuffer to create empty list
initialize global Hours to 0
initialize global Minutes to 0
initialize global connected to false
initialize global selectedBT to ""
initialize global selectedBTname to ""
initialize global msgToSend to ""
initialize global rcvMsg to ""
initialize global deviceList to ""

```

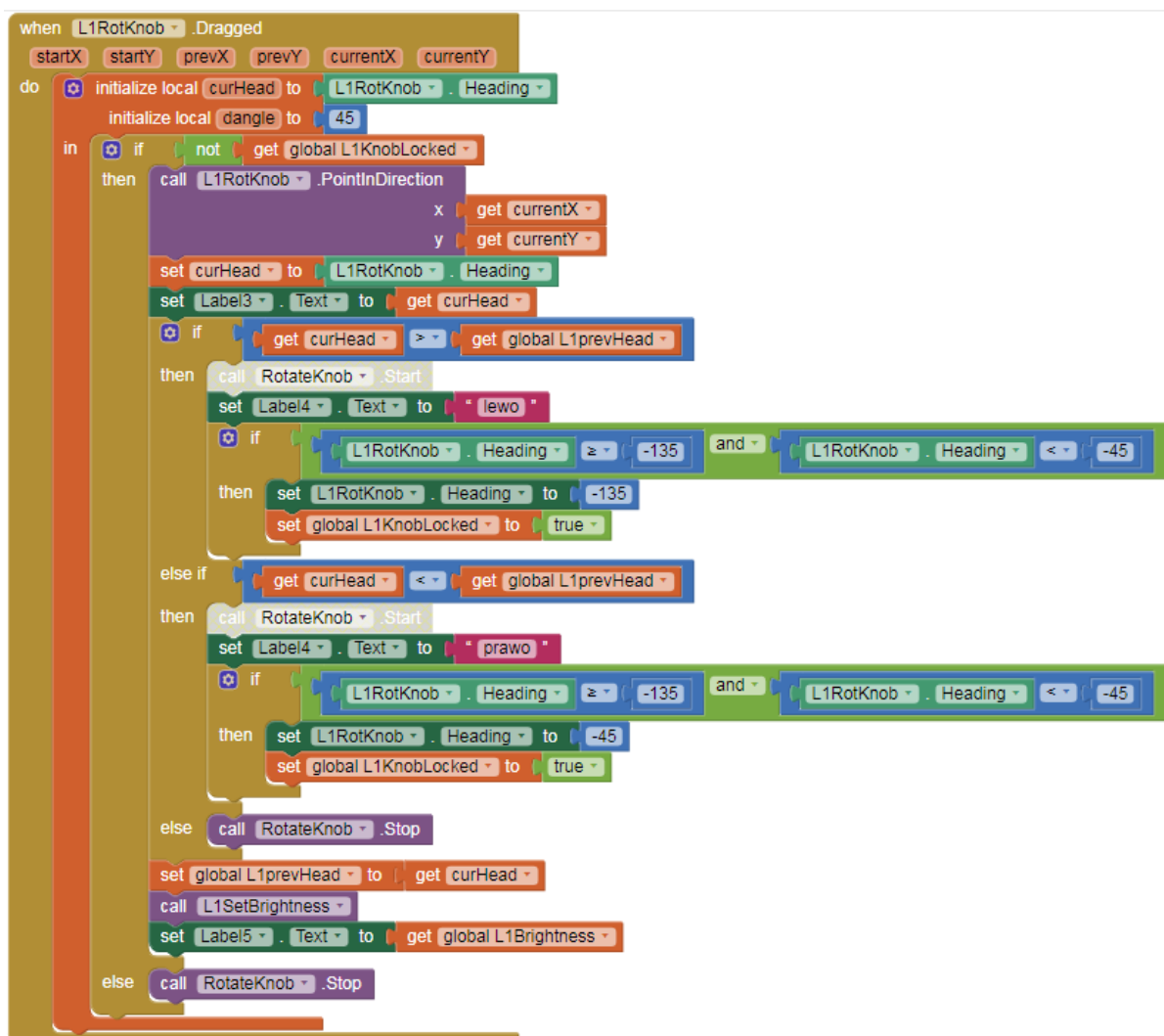
obsługa głównego okna

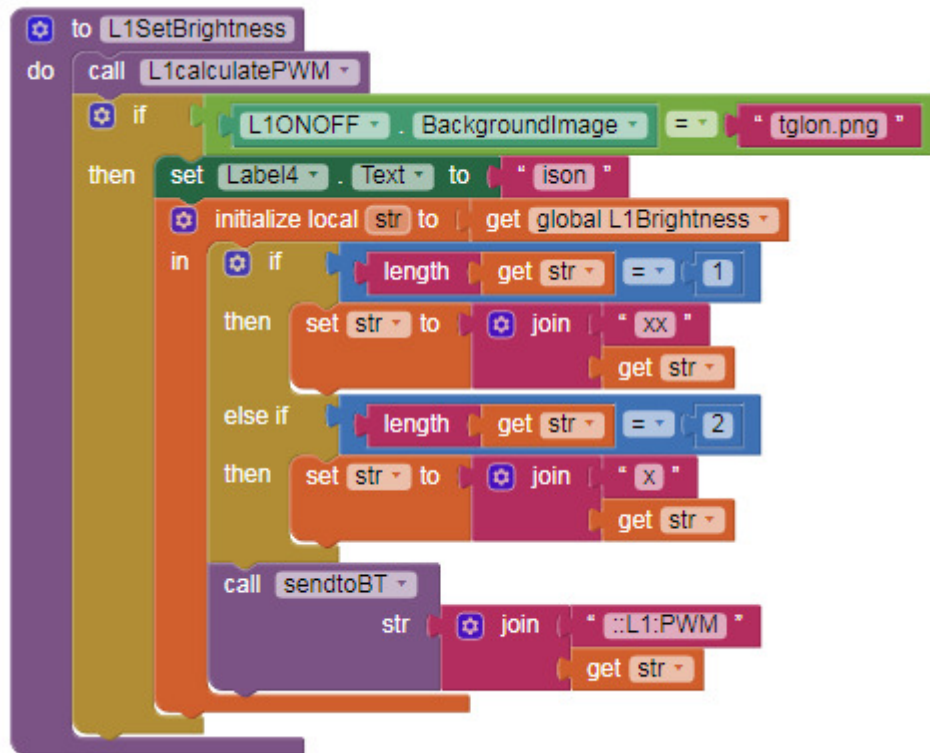


obsługa lampy

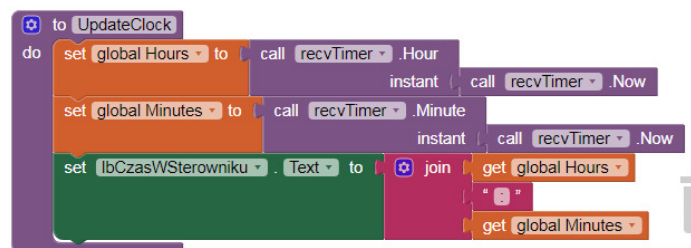
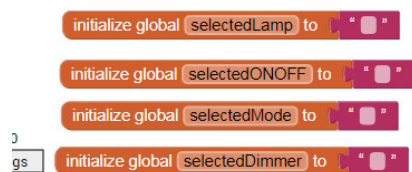
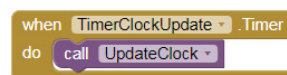
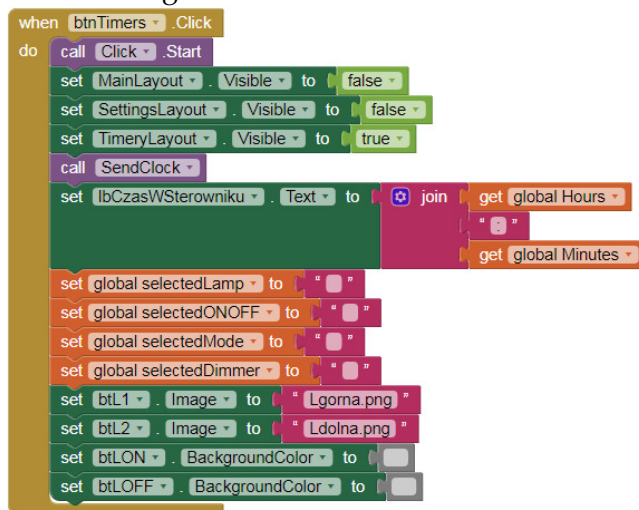


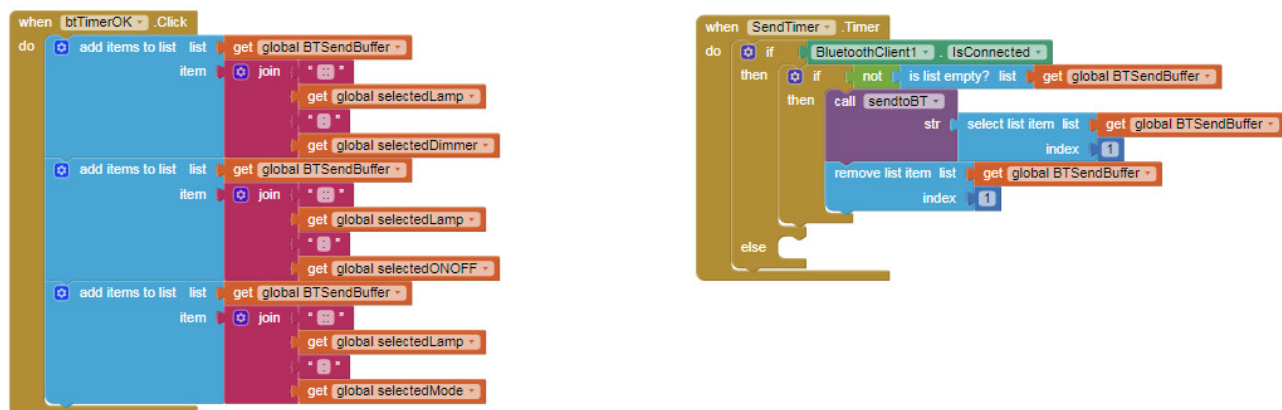
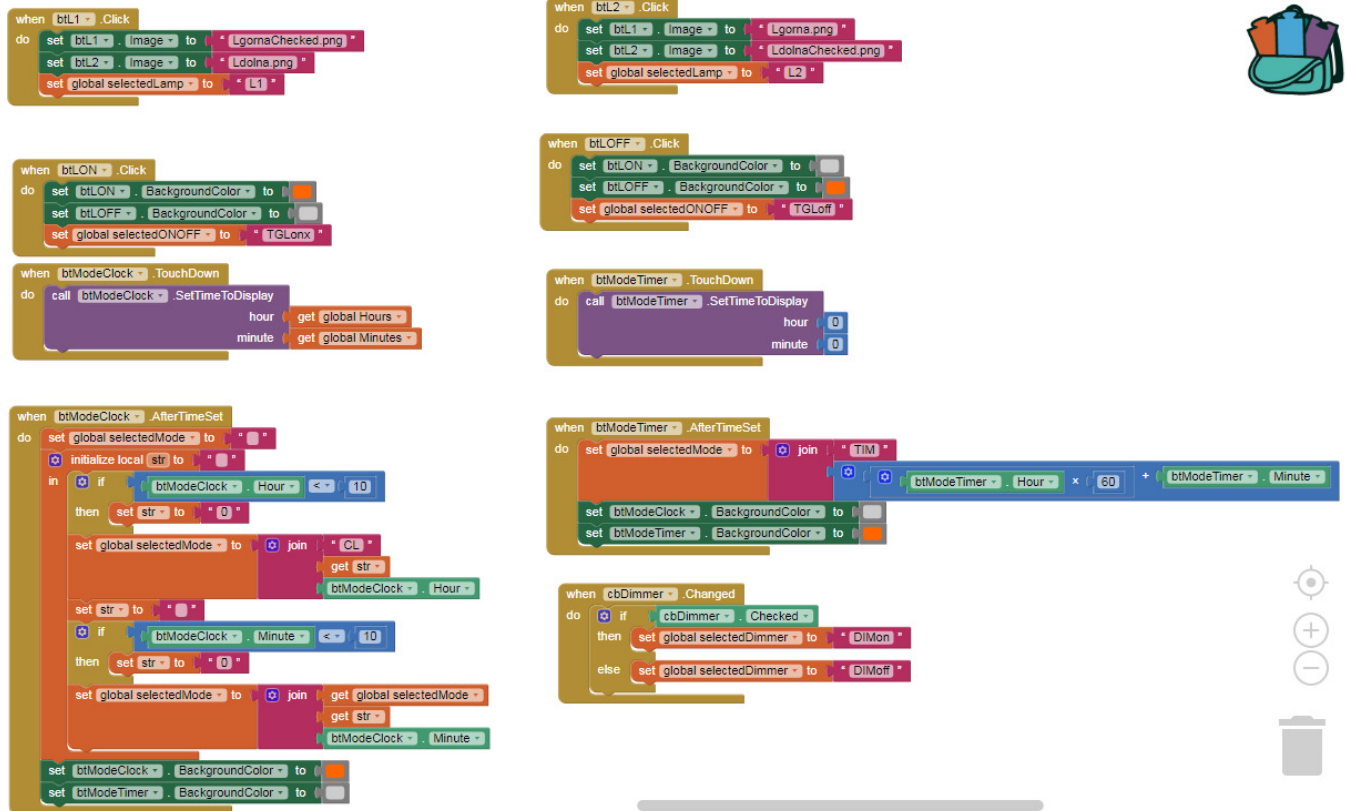
obsługa regulacji jasności



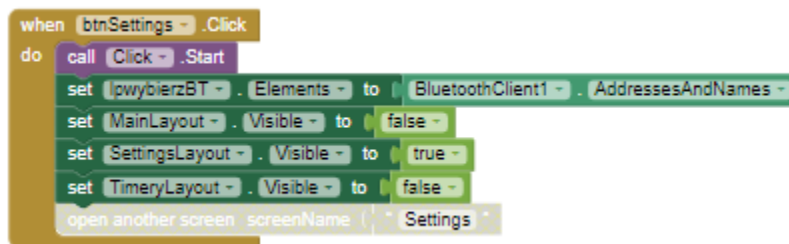


oblsuga timerow





obsługa ustawień (w trakcie)



- PC – dowolny terminal RS, lub program np w C# + kontrolka SerialPort

4.5. Firmware

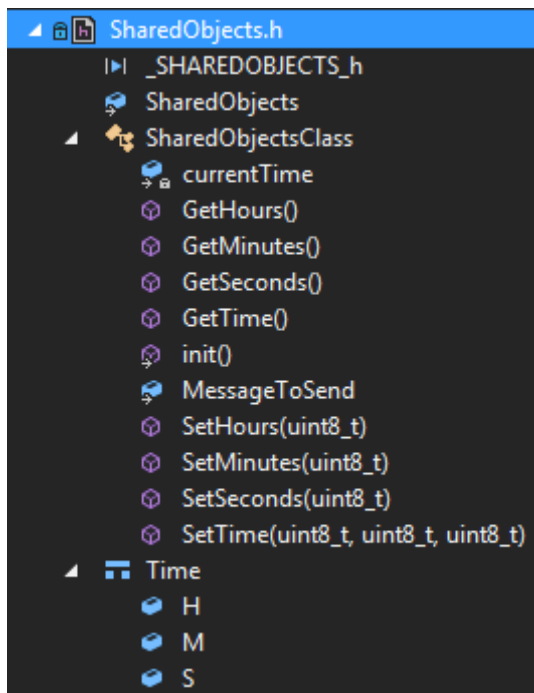
Platforma Arduino, C++, programowanie obiektowe. Większość jest wyjaśniona w komentarzach w kodzie.

Ogólnie firmware posiada klasę Lampa, i dwa obiekty tej klasy, zestaw metod do nich, plus pomocnicze statyczne metody w `SharedObjectsClass`, enumerator trybów, operacje na czasie itp.

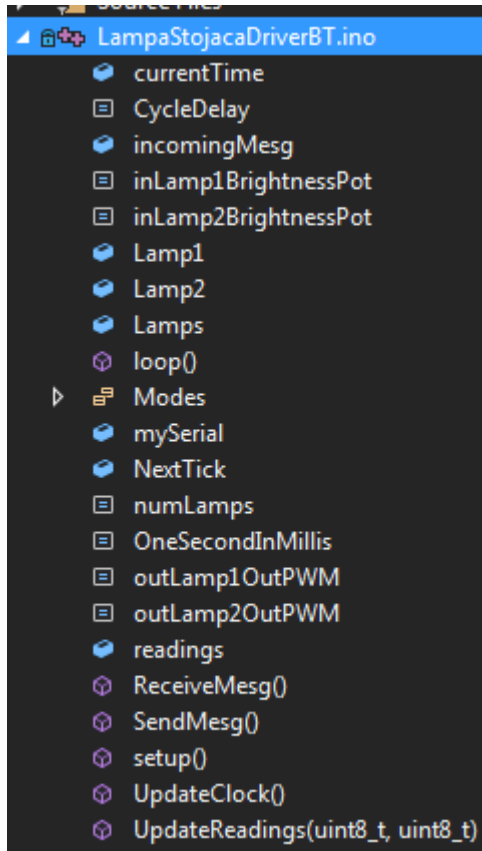
podczas działania sprawdza stan na pinach. po wykryciu prawidłowego komunikatu przechodzi w tryb BT, nie obowiązują wtedy ustawienia galek, po wykryciu zmiany na galkach przestają obowiązywać ustawienia BT (zerowane są timery)

obsługa trybu CLOCK jest zrealizowana za pomocą trybu timer – jest wyliczany czas między zadaną godziną i aktualną i wykonywana metoda dla TIMER

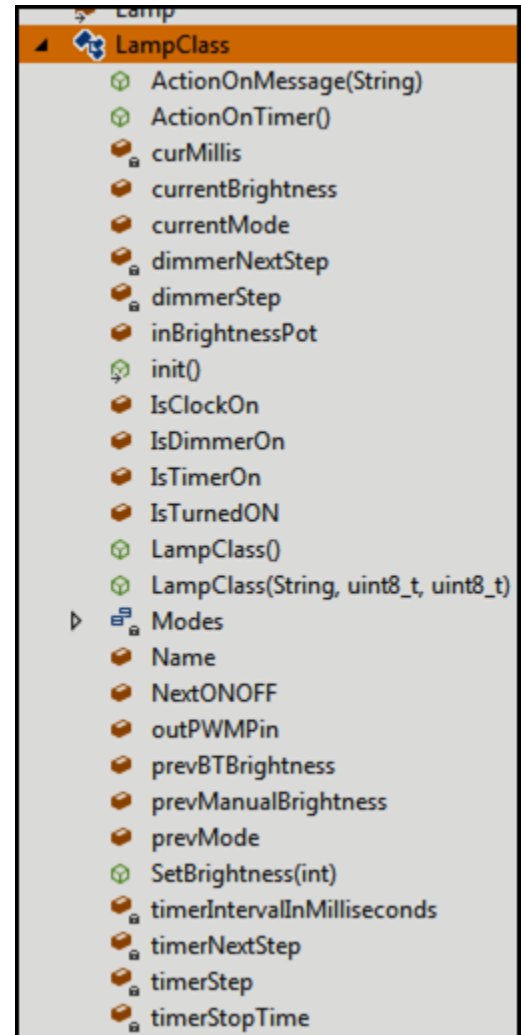
klasa SharedObjects (statyczne metody i struktury pomocnicze)



główna klasa



klasa LampClass



4.6. Listingi

```
// the setup function runs once when you press reset or power the board
#include <SoftwareSerial.h>
#include "Lamp.h"
#include "SharedObjects.h"
//#include <EEPROM.h>

//stale
const uint8_t numLamps = 2;
enum Modes { Manual = 0, Blututu };
const unsigned long int OneSecondInMillis = 1000;
const unsigned int CycleDelay = 10;

//komunikaty IN
//const String MsgSetClock = "HM:";
//const String MsgGetClock = "HM:?";

//komunikaty OUT

//pinout
//Lampa 1 - gorna 20W
const uint8_t inLamp1BrightnessPot = A2;
const uint8_t outLamp1OutPWM = 6;
LampClass Lamp1("L1", inLamp1BrightnessPot, outLamp1OutPWM);

//!!!!!!!!!!!!
//nazwy musza byc L1 L2... bo tak jest potem w komunikatach na sztywno ustawione inna nazwa i
//komunikat nie zadziala
//!!!!!!!!!!!!

//Lampa 2 - gorna 4W
const uint8_t inLamp2BrightnessPot = A1;
const uint8_t outLamp2OutPWM = 5;
LampClass Lamp2("L2", inLamp2BrightnessPot, outLamp2OutPWM);

//software serial
SoftwareSerial mySerial(13, 12); // RX, TX

LampClass Lamps[] = { Lamp1, Lamp2 }; //3 zeby nie zaczynac od indeksu 1-L1, 2-L2

//zmienne
String incomingMesg = "";
int readings[] = { 0, 0, 0 };
unsigned long int NextTick;
//unsigned int Hours = 0, Minutes = 0, Seconds = 0;
Time currentTime;
//SharedObjectsClass SharedObjects;

void setup() {
    Serial.begin(9600);
    Serial.println("hello serial");
    mySerial.begin(9600);
    mySerial.println("hello Myserial v2");
    pinMode(inLamp1BrightnessPot, INPUT);
    pinMode(outLamp1OutPWM, OUTPUT);
    pinMode(inLamp2BrightnessPot, INPUT);
    pinMode(outLamp2OutPWM, OUTPUT);
    NextTick = millis() + OneSecondInMillis; //+1 second

    for (int i = 0; i < numLamps; i++)
    {
        UpdateReadings(i, Lamps[i].inBrightnessPot);
        Serial.print("i= " + String(i));
        Serial.print(" name= " + Lamps[i].Name);
    }
}
```

```

        Serial.print(" pin= " + String(Lamps[i].inBrightnessPot));
        Serial.println(" read= " + String(readings[i]));
        Lamps[i].SetBrightness(readings[i]);
        Lamps[i].IsTurnedON = false;
        Lamps[i].prevBTBrightness = Lamps[i].currentBrightness;
        Lamps[i].prevManualBrightness = Lamps[i].currentBrightness;
    }
}

//void loop2() {
//    int read = analogRead(Lamp1.inBrightnessPot);//0-1023
//    Serial.print(read); Serial.print(" ");
//    int curManBrightness = map(read, 0, 1023, 0, 255);//0-255
//    Serial.print(curManBrightness); Serial.print(" ");
//    Lamp1.SetBrightness(curManBrightness);
//    Serial.print(Lamp1.currentBrightness); Serial.print(" \n");
//}

// the loop function runs over and over again forever
void loop() {
    delay(CycleDelay);
    //##### dodaj pobieranie satusu zmiennych zeby poustwaiac kontrolki w apce przy
    //polaczeniu
    //przeyslanie timera stanow alamek jasnosci itp
    String str = ReceiveMesg();//odebranie wiadomosci z nadajnika
    SendMesg();
    UpdateClock();
    //ReceiveMesg222();
    //UpdateClockv2(&Seconds);

    for (uint8_t i = 0; i < numLamps; i++)
    {
        //#####
        //zzaczynamy w trybie manual
        //#####
#pragma region MANUAL currentMode
        if (Lamps[i].currentMode == Modes(Manual)) //MANUAL
        {
            //przejsice w blututu
            if (str.startsWith(Lamps[i].Name))
            {
                Lamps[i].currentMode = Modes(Blututu);
                Lamps[i].prevMode = Modes(Manual);

                Serial.println("curmode " + Lamps[i].Name + " =man>>BT"
                    + String(Lamps[i].currentMode) + " "
                    + String(Lamps[i].currentBrightness));

                //zapamietanie ostatniego brightness w MAN
                UpdateReadings(i, Lamps[i].inBrightnessPot);
                Lamps[i].prevManualBrightness = readings[i];
            }
            else { //zostajemy w manual
                UpdateReadings(i, Lamps[i].inBrightnessPot);
                //Serial.print("man analog read " + String(read));
                //Serial.println(" man mapped analog readings[i] " +
String(readings[i]));
                Lamps[i].SetBrightness(readings[i]);
                //if (Lamps[i].currentBrightness < 5)
                //{
                //    //mySerial.println("::L1:PWM" +
String(Lamp1.currentBrightness) + "\n");
                //}
            }
        }
    }
#pragma endregion
}

```

```

//#####
//a teraz jesli tryb BT //blutut
//#####
#pragma region Blutut Mode
if (Lamps[i].currentMode == Modes(Blututu)) //BLUTUT
{
    Lamps[i].ActionOnTimer(); //tylko w btmode, w manual nie ma timera
    //czy przejsc w man mode
    UpdateReadings(i, Lamps[i].inBrightnessPot);
    if (
        (readings[i] > (Lamps[i].prevManualBrightness + 10))
//filtre bo ADC plywa moze dodoac jakies 100nF?
        ||
        (readings[i] < (Lamps[i].prevManualBrightness - 10)))
    {
        Lamps[i].currentMode = Modes(Manual);
        Lamps[i].prevMode = Modes(Blututu);
        Lamps[i].SetBrightness(readings[i]);
        Serial.println("curmode= " + Lamps[i].Name + " BT>>MAN"
            + String(Lamps[i].currentMode) + " "
            + String(Lamps[i].currentBrightness));
    }
    else if (str.startsWith(Lamps[i].Name)) {
        Serial.println(str);
        Lamps[i].ActionOnMessage(str); //a co z tym stringiem zrobic robi
metoda w klasie
    }
}
#pragma endregion
}
//#####KONIEC PRZELATYWANIA PO TABLICY LAMP

//#####ZAPYTANIA DO STEROWNIKA
//Przy przesyłaniu pamiętaj żeby długość komunikatu była 11 znaków
//dodaj te :: na początku, ale już do rozpoznawania jest bez tych ::
//##### ZAPYTANIA LUB USTAWIANIE PARAMETRÓW STEROWNIKA
if (str.startsWith("HM:")) { //HM:1234xx
    if (str.startsWith("HM:?")) { //która godzina
        mySerial.println("Clock: " + String(currentTime.H) + ":" +
String(currentTime.M) + ":" + String(currentTime.S));
        Serial.println("Clock send: " + String(currentTime.H) + ":" +
String(currentTime.M) + ":" + String(currentTime.S));
    }
    else //ustawianie zegara
    {
        currentTime.H = str.substring(3, 5).toInt();
        currentTime.M = str.substring(5, 7).toInt(); //
        SharedObjects.SetTime(currentTime.H, currentTime.M, currentTime.S);
        Serial.println("Clock Updated: " + String(currentTime.H) + ":" +
String(currentTime.M) + ":" + String(currentTime.S));
    }
}
}

String ReceiveMesg() {
    String str = "";
    if (mySerial.available() > 0) {
        str = mySerial.readString();
        str.trim();
        Serial.println("Received>> " + str); //echo na sterownik
        if (str.startsWith("::")) {
            if (str.length() > 11) {
                str = str.substring(str.length() - 9); //9 ostatnich znaków, już
bez "::"
            }
        }
    }
}

```



```

        else {
            str =str+"xxxxxxxxxx";           //dopelnienie do 11 znakow
            str = str.substring(2, 11);
        }
    }
    else str = "";
}
else str = "";
return str;
}

void SendMesg() {
    if (SharedObjects.MessageToSend != "") {
        mySerial.println(SharedObjects.MessageToSend);
        SharedObjects.MessageToSend = "";
    }
}

//void ReceiveMesg222() {
//    String str = "";
//    if (Serial.available() > 0) {
//        str = Serial.readString();
//        Serial.println("Received222>> " + str);
//    }
//}

void UpdateReadings(uint8_t i, uint8_t BrightnessPotentiometer) {
    readings[i] = analogRead(BrightnessPotentiometer);//0-1023
    readings[i] = map(readings[i], 0, 1023, 0, 255);//0-255
}

void UpdateClock() {
    if (millis() > NextTick)
    {
        currentTime.S += 1;
        if (currentTime.S >= 60)
        {
            currentTime.S = 0;
            currentTime.M += 1;
            Serial.println("CLK: " + String(currentTime.H) + ":" +
String(currentTime.M) + ":" + String(currentTime.S));
            if (currentTime.M >= 60)
            {
                currentTime.M = 0;
                currentTime.H += 1;
                if (currentTime.H >= 24) {
                    currentTime.H = 0;
                }
            }
        }
        SharedObjects.SetTime(currentTime.H, currentTime.M, currentTime.S);
        //Serial.println("CLK: " + String(currentTime.H) + ":" + String(currentTime.M) +
":" + String(currentTime.S));
        NextTick += OneSecondInMillis;
    }
}
}

```

4.7. Lamp.h

```

// Lamp.h

#ifndef _LAMP_h
#define _LAMP_h

#if defined(ARDUINO) && ARDUINO >= 100

```

```

#include "arduino.h"
#include "SharedObjects.h"

#else
#include "WProgram.h"
#endif

class LampClass
{
public:
    String Name;
    uint8_t inBrightnessPot;
    uint8_t outPWMPin;
    uint8_t currentBrightness;//0-255
    uint8_t currentMode;//enum "BT" or "Man"
    uint8_t prevMode;
    bool IsTurnedON;
    uint8_t prevBTBrightness;//do sprawdzania czy zmienic mode
    uint8_t prevManualBrightness;//do sprawdzania czy zmienic mode
    bool IsTimerOn;
    bool IsClockOn;
    bool IsDimmerOn;
    bool NextONOFF;    //0 off, 1 on
private:
    enum Modes { Manual = 0, Blututu };
    unsigned long int timerIntervalInMilliseconds;
    unsigned long int timerStopTime;
    unsigned long int timerStep = 1000;//3sek
    unsigned long int timerNextStep;
    unsigned long int dimmerStep;
    unsigned long int dimmerNextStep;
    unsigned long int curMillis;

public:
    void init();
    LampClass() {};
    LampClass(String name, uint8_t potPin, uint8_t PWMPin) {
        Name = name;
        inBrightnessPot = potPin;
        outPWMPin = PWMPin;
        prevManualBrightness = map(analogRead(potPin), 0, 1023, 0, 255);//0-255
        currentBrightness = prevManualBrightness;
        prevBTBrightness = currentBrightness;
        currentMode = Modes(Manual);
        prevMode = currentMode;
        IsTurnedON = false;
        //prevBTBrightness;//do sprawdzania czy zmienic mode
        //prevManBrightness;//do sprawdzania czy zmienic mode
        timerIntervalInMilliseconds = 0;
        IsTimerOn = false;
        IsDimmerOn = false;
        NextONOFF = 0;
    }

    //0-255

    void SetBrightness(int val) {
        if (val < 0) {
            val = 0;
        }
        if (val > 255) {
            val = 255;
        }
        currentBrightness = val;
        float factor = val / 255.;
        int brightnessToSet = (uint8_t)(val*factor*factor*factor); //doswiadczalnie zeby
jasnosc rosła w miare liniowo
    }
};

```

```

        analogWrite(outPWMPin, brightnessToSet);
        //Serial.println("curbright "+Name+" = "+String(currentBrightness));
    }

    void ActionOnMessage(String str) { //reakcja na blutut
        curMillis = millis();
        str = str.substring(3); //usuniecie "L1:"
        //Serial.println("str-3= " + str);

        //power on //::L1:onxxxx
        if (str.startsWith("on")) {
            IsTurnedON = true;
            IsTimerOn = false;
            SetBrightness(prevBTBrightness); //ostatnia brightness a nie 255
            Serial.println("BT " + Name + " on prevBright=" +
String(prevBTBrightness) + " on curBright=" + String(currentBrightness));
        }

        //power off //::L1:offxxx
        if (str.startsWith("off")) {
            IsTurnedON = false;
            IsTimerOn = false;
            prevBTBrightness = currentBrightness;
            SetBrightness(0);
            Serial.println("BT " + Name + " on prevBright=" +
String(prevBTBrightness) + " on curBright=" + String(currentBrightness));
        }

        //set brightness //::L1:PWM12x
        if (str.startsWith("PWM")) {
            IsTurnedON = true;
            str = str.substring(3); //usuniecie "PWM"
            str.replace("x", ""); //usuniecie xx
            currentBrightness = str.toInt(); //przeyslane jako 0-255 a nie 0-100
            SetBrightness(currentBrightness);
            Serial.println("BT " + Name + " setBright= " +
String(currentBrightness));
        }

        //##### TIMER //::L1:TIM12x
        if (str.startsWith("TIM")) { //set timer
            IsTimerOn = true;
            prevBTBrightness = currentBrightness;
            str = str.substring(3); //usuniecie "TIM"
            str.replace("x", ""); //usuniecie xx
            unsigned long int timerValueInMinutes = str.toInt(); //0-999 minut
            Serial.println("BT " + Name + " tim= " + String(timerValueInMinutes) +
"min");

            //#####timer w minutach tak jak trzeba do release
            timerIntervalInMilliseconds = timerValueInMinutes * 60 * 1000;
            //#####

            //minuty jako skenudy - to potem usun tylko do testow
            //timerIntervalInMilliseconds = timerIntervalInMilliseconds / 60;

            timerStopTime = curMillis + timerIntervalInMilliseconds;
            Serial.println("BT " + Name + " curmillis= " + String(curMillis));
            //Serial.println("BT " + Name + " curInterval= " +
String(timerIntervalInMilliseconds));
            Serial.println("BT " + Name + " timStop= " + String(timerStopTime));

            if (IsDimmerOn)
            {
                if (NextONOFF == 0) //ma wylaczyc lampe po czasie
                {

```

```

        dimmerStep = timerIntervalInMilliseconds /
currentBrightness;
        dimmerNextStep = curMillis + dimmerStep;
    }
    else { //ma wlaczyc lampe po czasie
        dimmerStep = timerIntervalInMilliseconds / (255 -
currentBrightness);
        dimmerNextStep = curMillis + dimmerStep;
    }
}
}
//##### cancel TIMER
if (str.startsWith("Ticnc1")) { //set dimmer
    IsTimerOn = false;
    Serial.println("BT " + Name + " TIMCancel");
}

//##### CLOCK //::L1:CLhmm
if (str.startsWith("CL")) { //set Clock on/off
    //CLOCK to to samo co timer tylko tutaj zamiast recznej wyliczac czas

timer
    //to tutaj licze podst godziny obecnej i nastawionej
    IsTimerOn = true;
    prevBTBrightness = currentBrightness;
    uint8_t CLKHours = str.substring(2, 4).toInt(); //
    uint8_t CLKMinutes = (str.substring(4, 6)).toInt(); //
    Serial.println("BT " + Name + " CLKtim= " + String(CLKHours) + ":" +
String(CLKMinutes));

    //przeliczanie na timer
    //dorob ifa gdy przepelnienie czyli godzina ustawiona wczesniej niz
obecan godzina

    unsigned long int curTimeInMinutes =
        SharedObjects.GetTime().H * 60
        + SharedObjects.GetTime().M;

    unsigned long int setTimeInMinutes =
        CLKHours * 60
        + CLKMinutes;

    //#####timer tak jak trzeba do release
    if (setTimeInMinutes > curTimeInMinutes)
    { //godizna nastawiona pozniej niz obecna
        timerIntervalInMilliseconds = (setTimeInMinutes -
curTimeInMinutes) * 60 * 1000;
    }
    else
    { //godizna nastawiona wczensije niz obecna
        timerIntervalInMilliseconds = ((24 * 60) + (setTimeInMinutes -
curTimeInMinutes)) * 60 * 1000;
    }

    //minuty jako skenudy - to potem usun tylko do testow
    //timerIntervalInMilliseconds = timerIntervalInMilliseconds / 60;

    timerStopTime = curMillis + timerIntervalInMilliseconds;
    timerNextStep = curMillis + timerStep;
    Serial.println("BT " + Name + " curmillis= " + String(curMillis));
    //Serial.println("BT " + Name + " CLKInterv= " +
String(timerIntervalInMilliseconds));
    Serial.println("BT " + Name + " CLKStop= " + String(timerStopTime));

    if (IsDimmerOn)
    {
        dimmerStep = timerIntervalInMilliseconds / currentBrightness;
        dimmerNextStep = curMillis + dimmerStep;
    }
}
}

```

```

    }
}

//##### cancel CLOCK
if (str.startsWith("Clnc1")) { //set dimmer
    IsTimerOn = false;
    Serial.println("BT " + Name + " CLKCancel");
}

//##### DIMMER
if (str.startsWith("DIMon")) { //set dimmer
    IsDimmerOn = true;
    Serial.println("BT " + Name + " DIMon");
}
if (str.startsWith("DIMoff")) { //set dimmer
    IsDimmerOn = false;
    Serial.println("BT " + Name + " DIMoff");
}

//##### czy wlaczyc/wylaczyc po dimmer
if (str.startsWith("TGLon")) { //set dimmer
    NextONOFF = 1;
    Serial.println("BT " + Name + " TGLon");
}
if (str.startsWith("TGLoff")) { //set dimmer
    NextONOFF = 0;
    Serial.println("BT " + Name + " TGLoff");
}

//#####zapytania ::L1:DIMoff
if (str.startsWith("?")) { //stan lampy
    if (currentBrightness > 0)
        str = ("::" + Name + ":" + "onxxx");
    else
    {
        str = ("::" + Name + ":" + "offxxx");
    }
    SharedObjects.MessageToSend = str;
    Serial.println("Status send: " + str);
}

if (str.startsWith("DIM?")) { //stan sciemniacza
    if (IsDimmerOn)
        str = ("::" + Name + ":" + "DIMonxxx");
    else
    {
        str = ("::" + Name + ":" + "DIMoffxxx");
    }
    SharedObjects.MessageToSend = str;
    Serial.println("Status send: " + str);
}

if (str.startsWith("BRGHT?")) { //stan sciemniacza
    str = ("::" + Name + ":" + "PWM=" + String(currentBrightness));
    SharedObjects.MessageToSend = str;
    Serial.println("Status send: " + str);
}
}

void ActionOnTimer() {
    curMillis = millis();
    if (IsTimerOn)
    {
        //do testow timera
        //if (curMillis > timerNextStep) {
        //    timerNextStep = curMillis + timerStep;

```



```

        //}
        if (IsDimmerOn)
        {
            if (curMillis > dimmerNextStep)
            {
                //Serial.println("dimstep" + String(dimmerStep) + " cur-
nextstep " + String(curMillis - dimmerNextStep));
                uint8_t brightnessStep = ((curMillis - dimmerNextStep) /
dimmerStep) + 1;
                if (NextONOFF == 0) {
                    SetBrightness(currentBrightness - brightnessStep);
                }
                else {
                    SetBrightness(currentBrightness + brightnessStep);
                }
                //dimmerNextStep = curMillis + dimmerStep;
                dimmerNextStep += brightnessStep * dimmerStep;
                Serial.println("BT " + Name + " DimFires " +
String(dimmerNextStep) + " Bright " + String(currentBrightness));
            }
        }

        if (curMillis > timerStopTime)
        {
            IsTimerOn = false;
            if (NextONOFF == 1)
            {
                IsTurnedON = true;
                SetBrightness(255);
            }
            else {
                IsTurnedON = false;
                SetBrightness(0);
            }
            Serial.println("BT " + Name + " TIMFires");
        }
    }
}

};

extern LampClass Lamp;

#endif

```

4.8. SharedObjects.h

```

// SharedObjects.h

#ifndef _SHAREDOBJECTS_h
#define _SHAREDOBJECTS_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "arduino.h"
#else
    #include "WProgram.h"
#endif
struct Time {
    uint8_t H; uint8_t M; uint8_t S;
};

class SharedObjectsClass
{
private:
    static Time currentTime;

public:

```

```

void init();
static String MessageToSend;

int GetHours() {
    return currentTime.H;
}
void SetHours(uint8_t val) {
    if (val<0 || val>23)currentTime.H=0;
    else currentTime.H = val;
}

int GetMinutes() {
    return currentTime.M;
}
void SetMinutes(uint8_t val) {
    if (val<0 || val>59)currentTime.M = 0;
    else currentTime.M = val;
}
int GetSeconds() {
    return currentTime.S;
}
void SetSeconds(uint8_t val) {
    if (val<0 || val>59)currentTime.S = 0;
    else currentTime.S = val;
}

void SetTime(uint8_t h, uint8_t m, uint8_t s) {
    currentTime.H = h;
    currentTime.M = m;
    currentTime.S = s;
}
Time GetTime() {
    return currentTime;
}
};

extern SharedObjectsClass SharedObjects;

#endif

```

5. Podsumowanie i Wnioski

Podczas pisania programu spotkałem się ze zjawiskiem że zabrakło pamięci, stała zajętość wyniosła ok 60%, jednak doszło do przepelnienia stosu, po usunięciu stałych stringów z definicji klasy, zamianie zmiennych int na uint_8t itp, zajetos spadła do 42% i juz problem juz się nie pojawiał. Pomogły by pewno tez dyrektywy preprocesora `#define`

Problem z czytelnością komunikatów czyli zapis ich gdzieś na początku mogłby rozwiązać jeszcze jeden enumerator (jak w trybach), wtedy zamiast całego stringu mamy tylko jeden bajt. Ale w firmware w zasadzie nie ma co rozbudowywać więc nie poprawiałem już tego.

Na przyszłość lepiej zainteresować się mikrokontrolerem STM32F103C9T6, podobny rozmiar nieco tańszy a oferujący dużo większe możliwości, dzięki bibliotece CubeMX przypomina Arduino z jego prostotą i szybkością łatwego tworzenia działających aplikacji

AppInventor nie spełnił oczekiwań, mimo że na początku wszystko ładnie działało to dość szybko dał się we znaki toporny interfejs gdzie każda kontrolka trzeba było do nowa ustawiać, nie ma kopiuj/wklej, layout nie odpowiada rzeczywistości, kompilator często zrywa połączenie.

Edytor z kolei ma kopiuj/wklej, ale nie można zaznaczać kilku obiektów co utrudnia organizację programu, komentarze są niewygodne.

AppInventor nadaje się do naprawdę prowizorycznych programów.

Po doswiadczeniach z Xamarinem zalowalem ze od razu nie napisalem klienta w C#, podejrzewam ze na walke z AppInventroem i bezmyslne i wtórne klikanie stracilem o wiele czasu niż zaoszczedzony na poczatku.

Ale zawsze to wiedza na przyszlosc, podobnie jak to ze kolejny projekt bede robil na STM32

6. Planowane Upgrady

6.1. ogólnie

- liniowe rozjaśnianie
- przyspieszenie wysylania – plynne rozjasnianie w czasei rzeczywistym
- zmiana nazwy modulu z HC-05 na KwadraLampa

6.2. Klient Android

- dzwieki on/off
- krotkei nazwy nadajnikow na liscie
- domsylly nadajnik zapisz , zapis aktualny jako domyslly,
- lacz przy starcie z domysllym,
- ikonka help i about
- ustwiaenia BT i inne do ekranu ustwaienia
- komunikacja w obie strony