

**Michał Kruk, Dariusz Gorgoń, Grupa E6C1S4**

Wojskowa Akademia Techniczna, Wydział Elektroniki

e-mail: [michal.kruk@student.wat.edu.pl](mailto:michal.kruk@student.wat.edu.pl), [dariusz.gorgon@student.wat.edu.pl](mailto:dariusz.gorgon@student.wat.edu.pl)

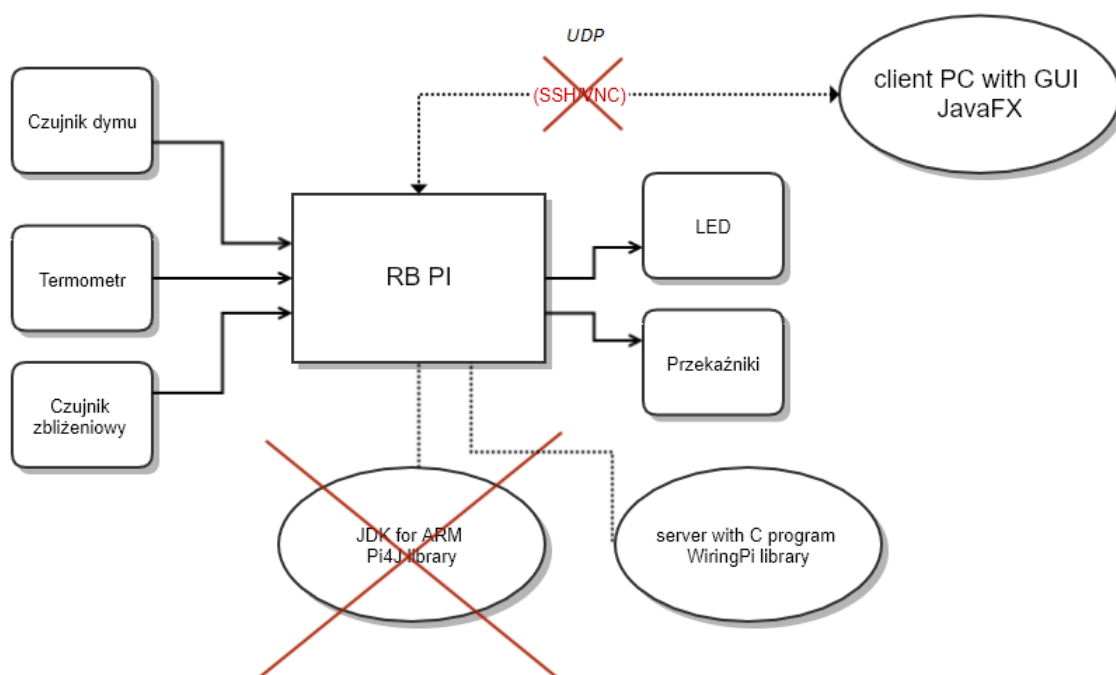
# **Zaawansowane Programowanie W języku JAVA Protokół**

## Spis treści

1. Założenia projektu .....	3
2. Warstwa sprzętowa: .....	4
3. Komunikacja między platformami: .....	4
4. Strona kliencka – GUI w JAVA: .....	5
5. Serce aplikacji: .....	8
6. Wykorzystanie zasobów: .....	10
7. Linki: .....	10

# 1. Założenia projektu

Projekt polega na zrobieniu przykładowego systemu inteligentnego domu z wykorzystaniem platformy Raspberry PI jako główna jednostka zarządzająca oraz systemu jego kontroli. Stworzona aplikacja kliencka będąca GUI użytkownika (ang. graphical user interface), pozwala osobie na monitorowanie aktualnego stanu systemu oraz sterowanie portami wejścia/wyjścia systemu. Cała aplikacja kliencka jest napisana w języku JAVA (czyt. Dżawa), z wyszczególnieniem technologii JavaFX. Główny projekt serwerowy znajduje się na jednostce centralnej (RPi) i jest napisany w języku C. Wybór tego języka był najlepszym rozwiązaniem dla szybkiego działania systemu. Poniżej przedstawiono schemat systemu:



Rys. 1 Schemat działania systemu

Początkowo rozpatrywano wykorzystanie biblioteki „Pi4J” stworzonej na rzecz układów Typu ARM. Jednak patrząc na niską wydajność w porównaniu z bibliotekami C, odrzucono te rozwiązanie.

Dodatkowo by można było stworzyć zewnętrzną aplikację, zamiast SSH wykorzystano protokół bezpołączeniowy UDP. Dzięki temu mieliśmy prostą komunikację między klientem a serwerem.

## 2. Warstwa sprzętowa:

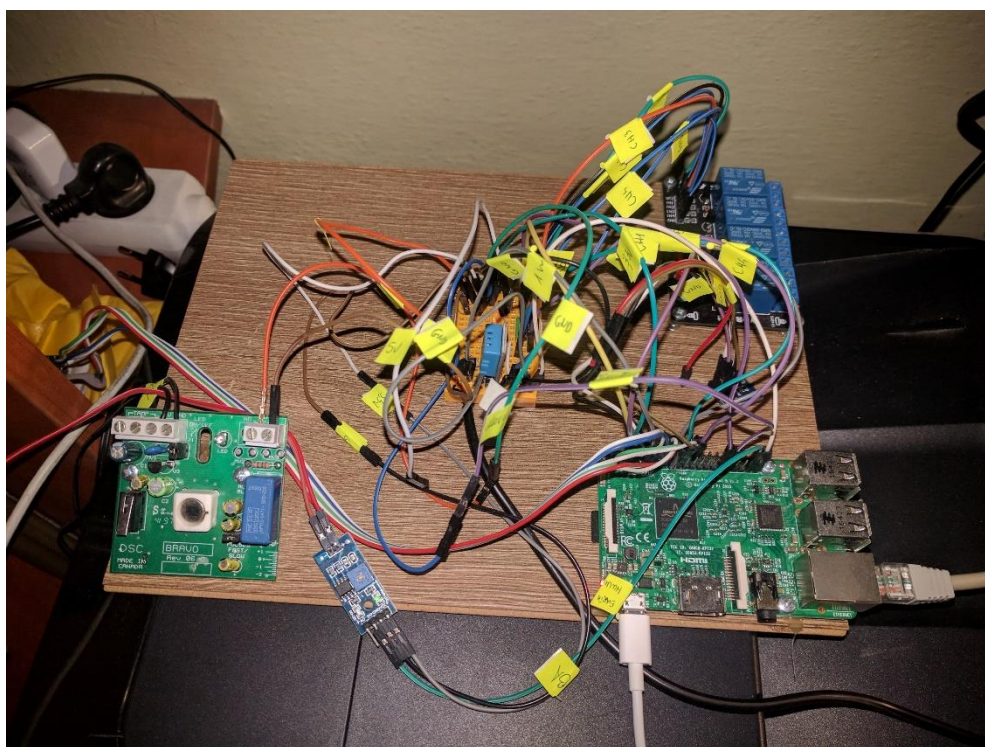
Jednostką nadrzędną, sterującą, jest Raspberry Pi 3, do którego podłączono czujniki i elementy sterujące (wykonawcze) za pomocą portów I/O. Czujniki jakie zaimplementowano w tym rozwiązaniu to:

- temperatury,
- natężenia oświetlenia,
- ruchu,
- poziomu wody.

Elementami wykonawczymi są:

- przekaźniki,
- LED.

Przy tworzeniu natknęliśmy się na moduły produkowane w Chinach, które bardzo utrudniły zadanie, gdyż były od nowości uszkodzone. Poniżej przedstawiono zdjęcie gotowego układu.



*Rys. 2 Jednostka sterująca wraz z sensorami i elementami wykonawczymi*

## 3. Komunikacja między platformami:

W celach ułatwienia komunikacji stworzyliśmy ramkę danych posiadającą określone komendy sterujące zdefiniowane przez nas:

Start of frame			Command	Write Read	Type	Size of Payload	
\$	K	G	CMD	W/R	T	S	PAYLOAD

Rys. 3

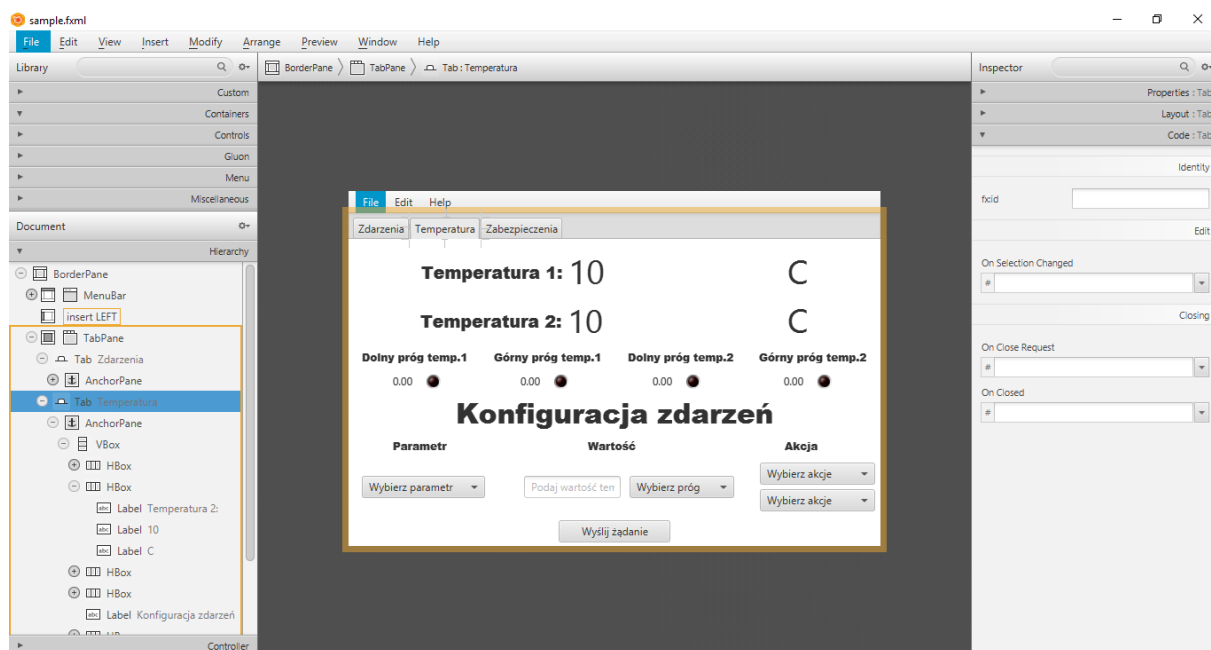
Pierwsze 3 bajty ramki to 3 charakterystyczne znaki określające początek ramki. Kolejne bajty to:

- CMD – znak komendy,
- W/R – bajt zapisu/odczytu
- T – typ zmiennej przechowywanej w payloadzie, w zależności od komendy może to być int, float lub string.
- S – rozmiar payloadu (ilość bajtów zawierających informacje)
- Payload – nasz dane.

Dzięki takiemu rozwiązaniu program „serwer” łatwiej analizował informacje. W razie błędów aplikacja serwera odsyła komunikat o typie błędu, co wyświetlane jest po stronie klienta.

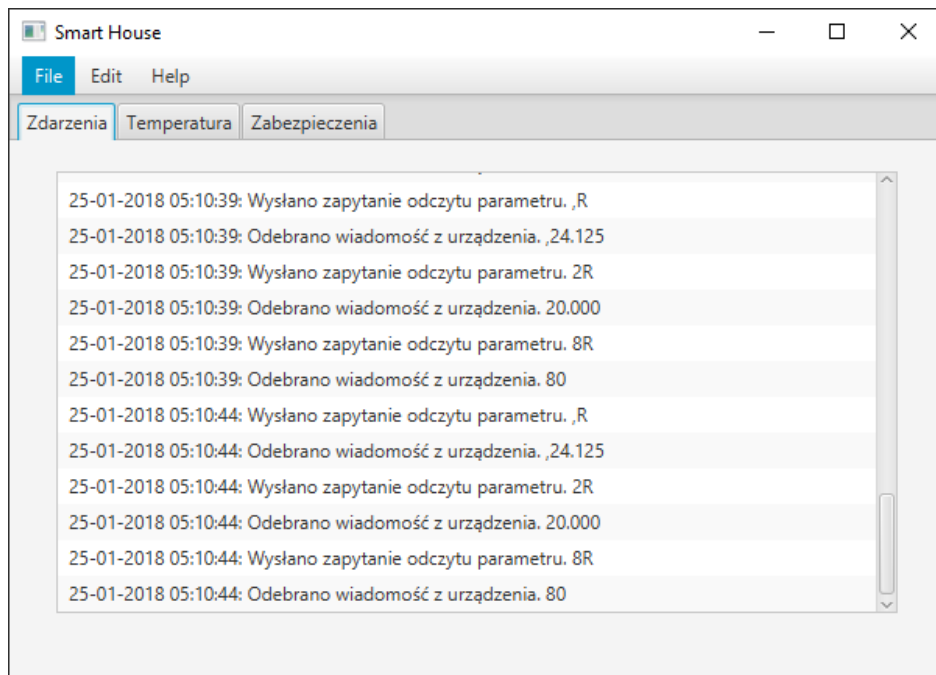
## 4. Strona kliencka – GUI w JAVA:

Tak jak wspomniano na początku, strona odbiorcza została napisana w JAVA. Użyto do tego SDK **IntelliJ IDEA** firmy **JetBrains**. Dodatkowo w celu stworzenia GUI posłużono się w większości rozszerzeniem **SceneBuilder**. Dzięki temu można było w łatwy sposób wygenerować pliki .xml, poprzez umieszczanie komponentów z listy dostępnych. Pliki XML można edytować we własnym zakresie, więc użycie tego rozszerzenia nie dyskwalifikuje metody podstawowej i można w szybki sposób wprowadzać niewielkie zmiany bez ciągłej potrzeby uruchamiania oddzielnej aplikacji. Przykład operowania rozszerzeniem przedstawiono poniżej :

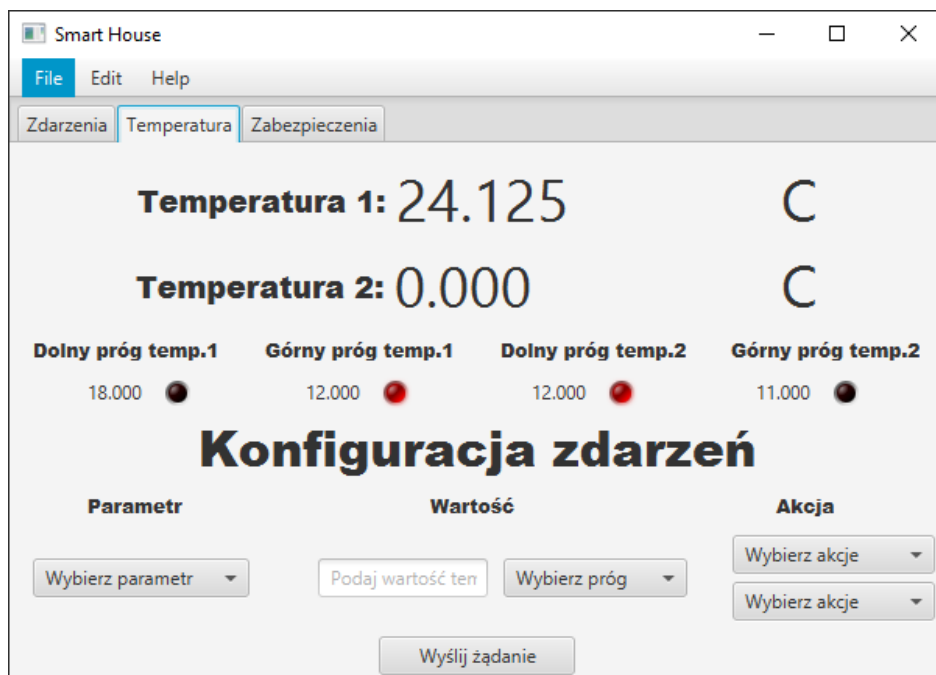


Rys. 4 Przykład operacji w SceneBuilderze

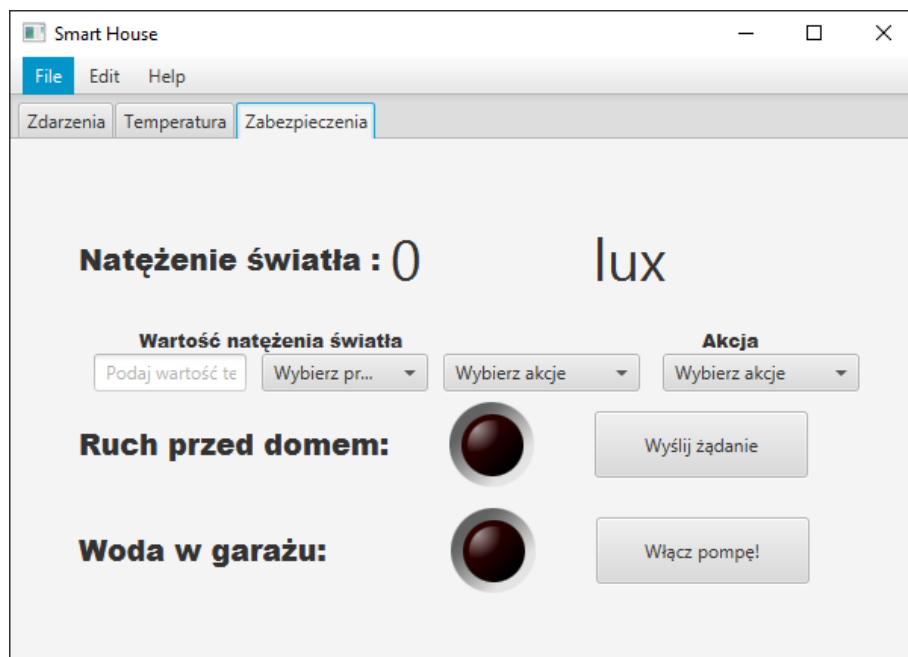
Korzystając z tego rozszerzenia udało nam się wygenerować Pełny szablon GUI naszej aplikacji.



Rys. 5 Zrzuty ekranu z aplikacji klienckiej



Rys. 6 Zrzuty ekranu z aplikacji klienckiej

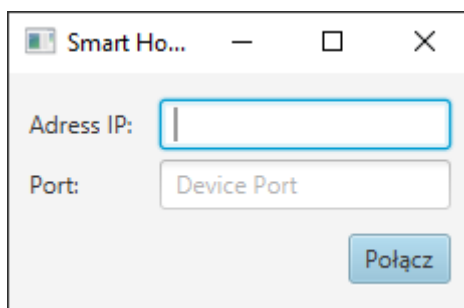


Rys. 7 Zrzuty ekranu z aplikacji klienckiej

Aplikacja posiada 3 zakładki, które odpowiadają odpowiednio:

- Za logi komunikacyjne:
- Sterowanie i wyświetlanie parametrów temperaturowych
  - Z określeniem reakcji podzespołów w zależności od naszego ustawienia
  - Monitorowaniem jego danych.
- Sterowanie i wyświetlanie innych parametrów odpowiedzialnych za
  - Wilgotność
  - Naświetlenie

Dodatkowo jest jeszcze panel komunikacyjny w którym wpisuje się port i adres IP urządzenia.

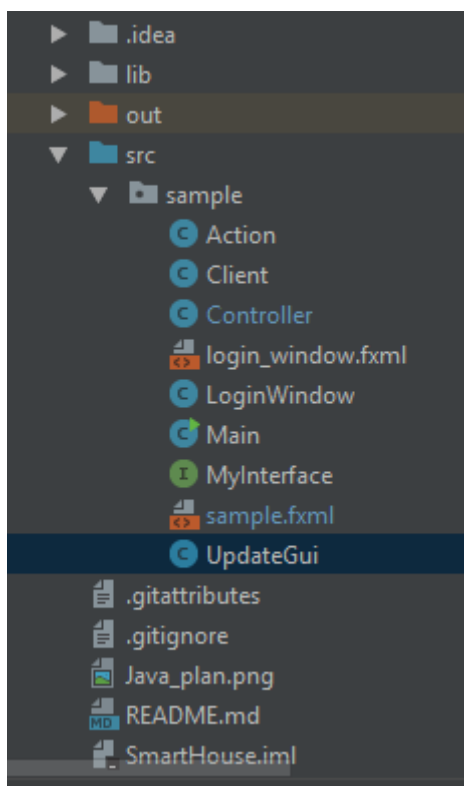


Rys. 8 Panel autoryzacji.

W celu ulepszenia GUI i ułatwienia wyświetlania, dodano zewnętrzną bibliotekę „ENZO”, która posiada kontrolki w JavaFX. Dzięki niej można było umieścić ora sterować diody na GUI.

## 5. Serce aplikacji:

Drzewo aplikacji posiadające wszystkie klasy przedstawiono poniżej:



Rys. 9

Klasa „Controller” jest standardową klasą tworzoną przez główny panel GUI. W niej mamy wszystkie metody odpowiedzialne za wyświetlanie danych w głównym oknie. Jednak w innych klasach również potrzebowaliśmy mieć informacji o nich, oraz mieć możliwość ustawiania parametrów. Dlatego Klasa Controller tworzy interfejs o nazwie „MyInterface” w którym są wszystkie elementy GUI potrzebne w innych miejscach.



```

4
5
6
7 void setLed2up(boolean check);
8 void setTempTres2up(String checkThres);
9 void setLed2down(boolean check);
10 void setTempTres2down(String checkThres);
11 void setLedlup(boolean check);
12 void setTempTreslup(String checkThres);
13 void setLedlDown(boolean check);
14 void setTempTreslDown(String checkThres);
15 //gettery
16 String getTempTres2up();
17 String getTempTres2down();
18 String getTempTreslup();
19 String getTempTreslDown();
20

```

Rys. 10

Dla klienta stworzyliśmy klasę, „Client” która posiadała w sobie jedną metodę odpowiadającą za wysłanie i odebranie komunikatu po UDP. Przyjmuje jako argumenty adres IP oraz nasz Interfejs, by można było odświeżać GUI w odpowiednim czasie.

By GUI Ciągłe miała aktualizowane pola należało użyć klasy platform i przez wyrażenie lambda „podpiąć” odświeżanie pod dany element GUI

```
Platform.runLater(() -> ...);
```

Klasa „Login Window” jest do wyświetlania panelu połączeniowego.

Klasy „Action” oraz „UpdateGUI” służą do ułatwienia wybierania komend oraz wyświetlenia informacji.

Metoda odpowiadająca za wysyłanie wiadomości :

```

public void sendMes (String msg) throws IOException{
    try {
        updateGui = new UpdateGui(myInterface);
        message = msg.getBytes();
        System.out.println(message);
        outputStream = new DatagramPacket(message, message.length,
        InetAddress, port);
        newSocket.send(outputStream);
        if (openWindow) {
            updateGui.visibleLog(msg, 1);
        }
        inputStream = new DatagramPacket(messageIn, messageIn.length);
        newSocket.receive(inputStream);

        String received = new String(
            inputStream.getData(), 0, inputStream.getLength());

        String test = received.startsWith("$KG") ? received.substring(7) :
        "Problem z Połączeniem";
        reciveMessage =test;
    }
}

```

```

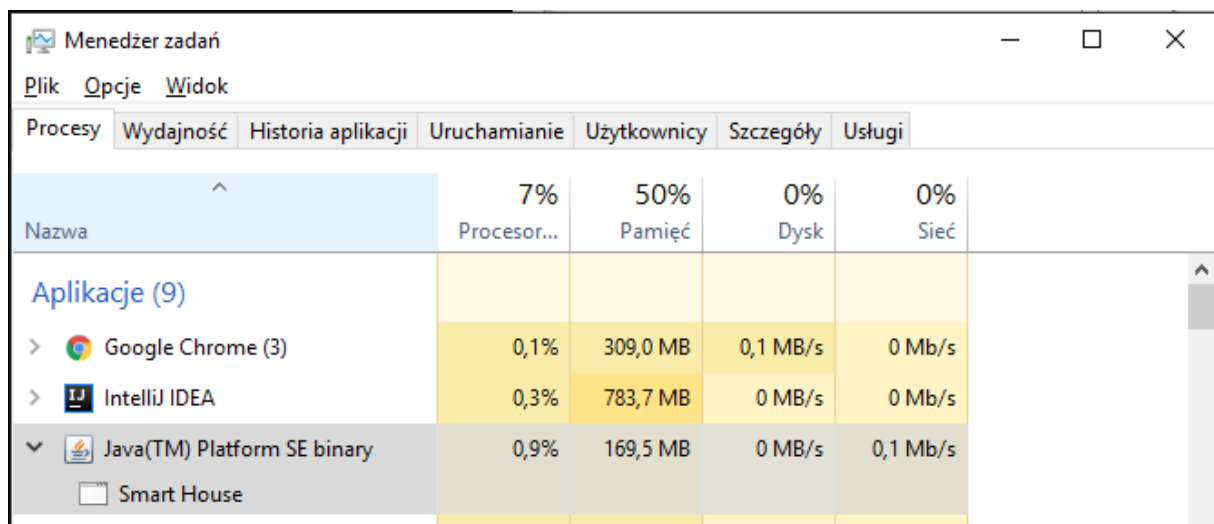
        if (openWindow) {
            updateGui.visibleLog(received, 2);
            updateGui.checkParam(msg, test);
        }
    } catch (IOException e) {
        {
            System.out.println(e);
            if (openWindow) {
                updateGui.visibleLog(e.toString(), 3);
            }
        }
    }
}

```

Adres oraz port do komunikacji są pobierane podczas pierwszego panelu, w którym wpisujemy te wartości. Potem dzięki przekazywaniu instancji (`getInstance()`) możemy wysyłać datagram UDP na ten sam adres z innej klasy, czy metody.

## 6. Wykorzystanie zasobów:

Aplikacja kliencka podczas uruchamiania rezerwuje sobie określoną ilość pamięci RAM. Poniżej przedstawiono zużycie pamięci RAM na stacji klienckiej:



The screenshot shows the Windows Task Manager window titled 'Menedżer zadań'. The 'Procesy' tab is selected. The table below represents the data shown in the task manager:

Nazwa	Procesor...	Pamięć	Dysk	Sieć
<b>Aplikacje (9)</b>				
> Google Chrome (3)	0,1%	309,0 MB	0,1 MB/s	0 Mb/s
> IntelliJ IDEA	0,3%	783,7 MB	0 MB/s	0 Mb/s
▼ Java(TM) Platform SE binary	0,9%	169,5 MB	0 MB/s	0,1 Mb/s
Smart House				

Rys. 11 Wykorzystanie zasobów stacji klienckiej

## 7. Linki:

Projekt aplikacji klienta:

- <https://github.com/DariuszGorgon/SmartHouse>

Projekt aplikacji sterującej w C:

- [https://github.com/krukm94/SmartHouse\\_Rpi](https://github.com/krukm94/SmartHouse_Rpi)