

|  |  |  |  |
| --- | --- | --- | --- |
| Projekt zespołowy | | | |
| **Kierunek** | *Informatyka* | **Termin** | *Czwartek 14:15* |
| **Temat** | *Projekt elastycznej aplikacji do zarządzania urządzeniami IoT w oparciu o bibliotekę QT* | **Zgłaszający** | *InterElcom* |
| **Skład grupy** | *Adam Krizar 241276*  *Katarzyna Czajkowska 242079*  *Mateusz Gurski 242089*  *Szymon Cichy 235093*  *Arkadiusz Cichy 236011* | **Nr grupy** | *-* |
| **Prowadzący** | *Dr inż. Jan Nikodem* | **data** | *2 marca 2020* |

Wykonał: Adam Krizar  
Sprawdziła:  
Zatwierdził:

Spis treści

[1. Organizacja pracy 3](#_Toc34581129)

[ Zarządzanie projektem 3](#_Toc34581130)

[ Aplikacja desktopowa 3](#_Toc34581131)

[ Backend 3](#_Toc34581132)

[ Zbieranie informacji 3](#_Toc34581133)

[2. Założenia i wymagania 3](#_Toc34581134)

[3. Opis zadania 4](#_Toc34581135)

[4. Środowisko 4](#_Toc34581136)

[ Instalacyjne 4](#_Toc34581137)

[ Programistyczne 4](#_Toc34581138)

[5. Sprzęt 5](#_Toc34581139)

[ Microchip ESP8266 5](#_Toc34581140)

[6. Plan realizacji 6](#_Toc34581141)

[ Pierwszy punkt kontrolny [19.03] 6](#_Toc34581142)

[ Drugi punkt kontrolny [02.04] 6](#_Toc34581143)

[ Trzeci punkt kontrolny [23.04] 6](#_Toc34581144)

[ Instalacja [07.05] 6](#_Toc34581145)

[ Testy użytkownika [21.05] 6](#_Toc34581146)

[ Oddanie projektu do użytku [04.06] 6](#_Toc34581147)

[ Prezentacja naszych osiągnięć [10.06] 6](#_Toc34581148)

# Organizacja pracy

## Zarządzanie projektem

Przygotowywanie dokumentacji, reprezentowanie grupy oraz zarządzanie projektem: Adam Krizar

Kontrola dokumentacji – korekta: Katarzyna Czajkowska

Ostateczne zatwierdzenie dokumentacji: Mateusz Gurski

## Aplikacja desktopowa

Projekt interfejsu graficznego, opracowanie portu na system Windows: Adam Krizar, Katarzyna Czajkowska

Obsługa protokołu HTTP: Mateusz Gurski

Obsługa protokołu MQTT: Adam Krizar, Katarzyna Czajkowska

## Backend

Obsługa HTTP po stronie IoT: Mateusz Gurski, Arkadiusz Cichy

Obsługa MQTT: Arkadiusz Cichy, Szymon Cichy

Obsługa funkcjonalności aplikacji (zapisywanie parametrów połączenia, listy urządzeń oraz odczyt tych plików): Adam Krizar, Arkadiusz Cichy, Szymon Cichy

Oprogramowanie modułów Wi-Fi: Arkadiusz Cichy Szymon Cichy

## Zbieranie informacji

Znalezienie informacji na temat protokołu MQTT: Adam Krizar, Katarzyna Czajkowska

Znalezienie czujników/urządzeń IoT wspierających wybrane protokoły: Szymon Cichy

Wyszukanie dodatkowych protokołów komunikacji z urządzeniami IoT: Szymon Cichy, Arkadiusz Cichy

# Założenia i wymagania

Założenia zostały opracowane bazując na wymaganiach złożonych przez firmę InterElecom, dostępnych na stronie: <https://kpz.pwr.edu.pl/projekty-zespolowe/>

Opis projektu zgłoszony przez firmę:

Projekt będzie dotyczył stworzenia aplikacji działającej na kilku platformach w oparciu o bibliotekę QT i język C++. Elastyczność aplikacji będzie polegała na możliwości zmiany protokołu komunikacji z urządzeniem IoT.  
- nowoczesna, prosta w obsłudze aplikacja

Wymagania przedstawione przez firmę:

 Użycie biblioteki QT oraz języka C++

 Stworzenie aplikacji działającej minimum na dwie platformy np. (Linux, Android).

 Stworzenie w aplikacji możliwości wyboru oraz sposobu dodawania nowych protokołu komunikacji z urządzeniem IoT.

 Obsługa w aplikacji minimum dwóch protokołów komunikacji z urządzeniem IoT np. (HTTP, MQTT).

W związku z tym zakładamy następujące cele naszego projektu:

- Wsparcie dla dwóch platform: W naszym przypadku zdecydowaliśmy się na wsparcie dwóch najpopularniejszych platform PC: Windows i Linux

- Wykorzystanie biblioteki QT w najnowszej wersji (5.14 ) oraz języka C++

- Zgodnie z wymaganiami zaimplementujemy wsparcie dla dwóch protokołów komunikacji z urządzeniami IoT: HTTP oraz MQTT.

- Zbudowanie i zaprogramowanie dwóch czujników w celu zaprezentowania funkcjonalności programu.

- Zapewnienie prostego i czytelnego interfejsu użytkownika: Aplikacja powinna być gotowa do użytku bez konieczności przeznaczania czasu na naukę jej obsługi.

- Czytelne menu pomocy, które w jasny sposób pomoże w obsłudze aplikacji.

- Parowanie urządzeń: Możliwość zapisywania parametrów połączenia z urządzeniami w celu szybkiego odbudowania sieci po na przykład: utracie zasilania.

- Możliwość edycji sparowanego połączenia: Zmiana adresu IP urządzenia IoT lub zmiana protokołu komunikacji z urządzeniem.

- Możliwość nadawania nazw podłączanym urządzeniom w celu łatwej identyfikacji.

- Możliwość zapisywania danych do plików co pewien przedział czasu regulowany przez użytkownika: (30 sekund, minuta, 5 minut, 10 minut, godzina)

# Opis zadania

Naszym celem jest zaprojektowanie i zaprogramowanie stabilnej i przejrzystej aplikacji do komunikacji z urządzeniami IoT. W tym celu musimy przygotować i zaprogramować dwa testowe urządzenia, które posłużą nam do testów naszego rozwiązania.

Szczególną uwagę musimy zwrócić uwagę na projekt interfejsu aplikacji. Wszystkie opcje muszą być dobrze opisane oraz w jasny sposób przekazywać użytkownikowi do czego służą. Ważne jest także zaimplementowanie systemu pomocy, który w sposób opisowy wytłumaczy użytkownikowi jak używać funkcji naszej aplikacji.

Musimy także znaleźć i przygotować do naszych potrzeb urządzenia oferujące odpowiednie protokoły komunikacyjne i przygotować je do naszych potrzeb. Ważne jest przestrzeganie przez nas stworzonych już protokołów w celu zapewnienia kompatybilności z istniejącymi już urządzeniami IoT.

# Środowisko

## Instalacyjne

Ze względu na brak informacji na ten temat od strony zgłaszającego musimy sami przyjąć pewne założenia. Nie wiemy na jakim obszarze będą znajdować się urządzenia, z którymi będzie komunikować się nasza aplikacja. W związku z tym zakładamy, że taka łączność będzie odbywać się przez sieć lokalną poprzez łącze przewodowe bądź z użyciem transmisji bezprzewodowej Wi-Fi.

Na ten moment w czasie trudno nam jest określić wymagania sprzętowe dla naszej aplikacji. Zakładamy jednak, że każdy sprzęt, na którym może działać nowoczesny 64 bitowy system operacyjny (Windows 10, dystrybucje Linuxa z interfejsem graficznym) będzie wystarczający dla naszych potrzeb.

## Programistyczne

Zgodnie z wymaganiami narzuconymi nam przez zgłaszającego projekt nasza aplikacja będzie wykorzystywała język c++ i framework Qt.

W związku z brakiem informacji na temat wersji z której mamy korzystać uznaliśmy, że pozostawiono ten wybór w naszych rękach. W związku z tym framework Qt zamierzamy użyć w najnowszej dostępnej wersji 5.14.1. Dzięki temu mamy najnowsze narzędzia dostępne do naszej dyspozycji.

Do stworzenia aplikacji desktopowej planujemy użyć programu Qt Creator oraz Qt Designer, które znacząco ułatwią na prace nad interfejsem użytkownika i resztą aplikacji dzięki swojej integracji z frameworkiem Qt.

Wykorzystanie aplikacji Qt Designer znacząco ułatwia nam projektowanie strony graficznej naszej aplikacji. Dzięki wykorzystaniu tej aplikacji mamy dostęp do prostego graficznego interfejsu umożliwiającego umieszczanie elementów interfejsu użytkownika poprzez wybranie ich z listy obiektów wspieranych przez framework Qt, a następnie umieszczenie ich w wybranym przez nas miejscu poprzez bez konieczności pisania linijki kodu. Dzięki wykorzystaniu opcji layoutów wokół których zbudowane jest wiele aplikacji opartych o Qt, możemy uzyskać automatyczne skalowanie elementów interfejsu do rozmiarów okna.

Ze względu na konieczność obsłużenia dwóch różnych platform przez naszą aplikacje jesteśmy ograniczeni w wykorzystaniu bibliotek natywnych dla systemów operacyjnych ze względu na możliwe braki w ich implementacji lub inne działanie na różnych platformach. W związku z tym nie możemy użyć na przykład biblioteki conio.h, która jest dostępna tylko pod systemem Windows. Dzięki wykorzystani frameworku Qt mamy jednak dostęp do wielu gotowych klas, które zadziałają na każdej platformie wspieranej przez Qt bez konieczności przepisywania kodu od początku. Przykładowo planujemy użyć następujących klas:

QNetworkAccessManager – jest to klasa, która umożliwia w prosty sposób implementacje protokołu HTTP.

QMqttClient – klasa, która umożliwia obsługę protokołu MQTT.

Kolejne klasy i wykorzystane przez nas biblioteki zostaną udokumentowane wraz z rozwojem aplikacji.

Po stronie urządzeń IoT planujemy wykorzystać narzędzia takie jak Arduino IDE, które w łatwy sposób umożliwiają nam programowanie wybranych przez nas urządzeń. Planujemy użyć takich bibliotek jak MQTT.h czy Ethernet.h, które pozwoliły by nam na zaimplementowanie wybranych przez nas protokołów.

Wybór konkretnych narzędzi i bibliotek programistycznych zostanie uzupełniony gdy uda się nam wybrać i zabezpieczyć konkretne urządzenia spełniające nasze wymagania.

W celu ułatwienia pracy w grupie będziemy wykorzystywać narzędzie kontroli wersji, dzięki któremu będziemy mogli w łatwy sposób łączyć wyniki naszej pracy i udostępniać je między sobą. W tym celu wykorzystujemy repozytorium utworzone na platformie Github.

# Sprzęt

## Mikrokontroler ESP8266

Mikrourządzenie, które dzięki swoim niewielkim rozmiarom i niskim kosztom jest idealne do zastosowania w naszym projekcie. Istnieje wiele SDK, które ułatwiają pisanie programów oraz obsługę peryferiów.

Specyfikacje:

- 80 Mhz CPU

- 32 KiB instruction RAM

- 32 KiB instruction cache RAM

- 80 KiB user-data RAM

- 16 KiB ETS system-data RAM

- IEE 802.11 b/g/n Wi-Fi

Ponadto urządzenie wspiera wybrane przez nas standardy komunikacji.

Dodatkowe informacje dostępne są pod adresem: <https://en.wikipedia.org/wiki/ESP8266>

## Mikrontroler Arduino Uno/Nano

Urządzenie pomimo gorszych specyfikacji i braku wbudowanego modułu Wi-Fi jest wspierane przez wielu pasjonatów i posiada rozbudowane biblioteki ułatwiające programowanie.

Specyfikacje:

- 16 Mhz CPU

- 32 KiB pamięci flas

- 2 KiB SRAM

- 1 KiB EEPROM

- Ilośc pinów I/O: 14 dla Arduino Nano, 22 dla Arduino Uno

- Zasilanie: 7-12V

## Moduł Wi-Fi NRF24L01

Ze względu na brak wbudowanego modułu Wi-FI w mikrokontrolerach arduino w wypadku wyboru rozwiązania opartego o nie konieczne będzie zaopatrzenie się w zewnętrzny moduł komunikacji bezprzewodowej.

Specyfikacje:

- Częstotliwość 2.4 Ghz

- Prędkości od 250 kbps do 2 Mbps

- Zasięg do 100 metrów (w przestrzeni otwartej)

## Czujnik DHT11

Tanie rozwiązane dzięki któremu będziemy mogli odczytywać dane testowe z urządzenia IoT

Specyfikacje:

- Częstotliwość próbkowania max raz na sekundę.

- Odczyt wilgotności z zakresu 20-80% z dokładnością 5%.

- Odczyt temperatury z zakresu 0-50°C z dokładnością 2°C.

# Kosztorys

Głównym kosztem w realizacji naszego projektu są urządzenia IoT koniecznie do testowania i prezentacji możliwości naszej aplikacji. Potrzebne są nam dwie platformy testowe:

Mikrokontroler ESP8266: <https://allegro.pl/oferta/esp8266-nodemcu-v3-wifi-2-4ghz-ch340-do-arduino-7241549772>, koszt 18,90 zł.

Mikrokontroler Arduino Uno: <https://allegro.pl/oferta/arduino-uno-r3-atmega328-ch340-avr-klon-fv-7645191761>, koszt 15,60 zł

Mikrokontroler Arduino Nano: <https://allegro.pl/oferta/arduino-nano-v3-0-v3-atmega328-16mhz-7941801500>, koszt 16,90 zł

Moduł Wi-Fi NRF24L01: <https://allegro.pl/oferta/nrf24l01-lacznosc-bezprzewodowa-2-4ghz-arduino-8890452977>, koszt 5,99 zł

Czujnik DHT11: <https://allegro.pl/oferta/dht11-czujnik-temperatury-i-wilgotnosci-arduino-7487941486>, koszt 4,70 zł

Całkowity koszt w zależności od wybranej podstawki wynosi odpowiednio:

ESP8266: 47,20 zł  
Arduino Uno: 52.58 zł  
Arduino Nano: 55,18 zł

# Plan realizacji

## Pierwszy punkt kontrolny [19.03]

Podstawowa aplikacja na platformę Linux oraz implementacja jednego protokołu komunikacji z IoT.

## Drugi punkt kontrolny [02.04]

Dokończenie aplikacji desktopowej na platformę Linux oraz testy łączności z urządzeniami IoT. Wstępna implementacja drugiego protokołu komunikacji.

## Trzeci punkt kontrolny [23.04]

Przeniesienie aplikacji na platformę Windows. Dokończenie implementacji drugiego protokołu IoT.

## Instalacja [07.05]

## Testy użytkownika [21.05]

Wprowadzenie ewentualnych korekt w projekcie interfejsu użytkownika zgodnie z uwagami użytkownika końcowego.

## Oddanie projektu do użytku [04.06]

## Prezentacja naszych osiągnięć [10.06]