Wstępna analiza funkcjonalna aplikacji „Bardzo mini SCADA”

# 1. Koncepcja i cele projektu

Stworzenie prostego systemu monitorowania i wizualizacji danych.

Projekt składa się z dwóch podstawowych elementów:

1. **Serwer** – generuje i udostępnia w sposób ciągły dane pomiarowe przez sieć (np. poprzez TCP/IP).
2. **Klient** – odbiera dane z serwera i zapewnia ich wizualizację w postaci wykresów.

Aplikacja powinna być:

* **Łatwa w uruchomieniu i obsłudze**
* **Rozszerzalna** – możliwość dodawania własnych metod przetwarzania danych (w formie pluginów lub parsera wyrażeń matematycznych).
* **Elastyczna** w zakresie prezentacji danych – skalowanie osi, zmiana kolorów, rodzajów linii, siatki itp.

# 2. Opis działania

1. **Serwer**
   * Uruchamiany jako osobna aplikacja
   * W określonych odstępach czasu wysyła losowe dane pomiarowe
   * Może przyjmować prostą konfigurację
2. **Klient**
   * Po uruchomieniu umożliwia wybranie adresu i portu serwera.
   * Klient łączy się z serwerem, a następnie cyklicznie odbiera dane pomiarowe.
   * Dane są gromadzone w wewnętrznym buforze – tak, by można było:
     + wyświetlić wykres w czasie rzeczywistym,
     + przeglądać dane historyczne (ewentualnie w ograniczonym zakresie, np. ostatnie 1000 próbek).
   * Przed wyświetleniem użytkownik może zastosować wybrane przetwarzanie danych
   * Dane są wyświetlane na wykresie w czasie rzeczywistym. Użytkownik może dostosować parametry widoku:
     + zakres osi X i osi Y,
     + siatkę (włączenie/wyłączenie, gęstość),
     + kolory linii, rodzaj linii,
3. **Buforowanie i opcje historyczne**
   * Klient przechowuje w pamięci pewną liczbę ostatnio odebranych próbek (np. 1000 czy 5000) w celu umożliwienia przewinięcia wykresu.
   * Po osiągnięciu limitu starsze dane są usuwane w trybie FIFO (first in, first out).
4. **Struktura klas (szkic)**
   * **DataServer**
     + odpowiada za generowanie i wysyłanie danych (odpowiedzialny za socket, wysyłanie w pętli).
   * **DataClient**
     + odpowiada za nawiązywanie połączenia z serwerem i odbiór danych.
     + udostępnia metody do buforowania danych i podstawowe API dla modułu wizualizacji.
   * **DataProcessor** (interfejs lub klasa bazowa)
     + w przypadku wtyczek: klasa bazowa, z której dziedziczą konkretne filtry/przetwarzania (np. FilterSmooth).
     + w przypadku parsera: klasa zawierająca mechanizm interpretacji wyrażeń matematycznych.
   * **PlotWidget** (moduł wyświetlania)
     + zarządza renderowaniem wykresów.
     + umożliwia konfigurację wyglądu (kolory, zakresy, siatka).
   * **MainWindow** (w przypadku interfejsu GUI)
     + obsługuje interakcje użytkownika (menu, przyciski, panele konfiguracji).

# Szkic interfejsu użytkownika

-----------------------------------------------------------

| [Menu] [Połącz z serwerem] [Konfiguracja wtyczek] ... |

-----------------------------------------------------------

| | |

| Lista | Wykres danych (PlotWidget) |

| dostępnych |------------------------------------------- |

| filtrów | [Opcje wyświetlania] |

| | - Skala osi X, Y |

| | - Kolor linii, rodzaj linii |

| | - Włącz/wyłącz siatkę |

---------------------------------------------------------

| Status: Połączono / Brak połączenia |

---------------------------------------------------------

# 4. Technologia i środowisko

* **Kompilator / Środowisko**:
  + **MinGW**.
  + IDE: **Qt Creator**
* **Biblioteki**:
  + Do obsługi sieci: standardowe gniazda (sockets) w C++ lub rozwiązania z Qt (QTcpServer).
  + Biblioteka do rysowania wykresów:
    - **QChart** lub **QCustomPlot**
  + W przypadku parsera – exprtk
* **System operacyjny**: Windows

Diagram klas

# 1. Szkic diagramu (UML) Obraz zawierający tekst, zrzut ekranu, wizytówka, Czcionka Opis wygenerowany automatycznie

# 2. Opis klas i odpowiedzialności

### 2.1. **MainWindow** (interfejs użytkownika)

**Odpowiedzialność:** Główne okno aplikacji, zarządzanie interakcją użytkownika

**Główne zadania**:

* Inicjalizuje obiekty logiki (np. DataClient) i interfejsu (np. PlotWidget).
* Odbiera akcje użytkownika (np. kliknięcie przycisku „Połącz z serwerem”) i wywołuje odpowiednie metody logiki.
* Umożliwia konfigurację dodatkowych wtyczek / filtrów i ich zastosowanie do danych.

**Kluczowe pola/metody**:

* dataClient : DataClient
* plotWidget : PlotWidget
* onConnectButtonClicked()
* onApplyFilterButtonClicked()

### **2.2 DataServer** (warstwa serwera)

**Odpowiedzialność**: Generowanie danych pomiarowych w pętli oraz udostępnianie ich przez sieć.

**Główne zadania**:

* Tworzy gniazdo i słucha na ustalonym porcie.
* Okresowo generuje losowe dane pomiarowe.
* Wysyła dane do podłączonych klientów.

**Kluczowe pola/metody**:

* port : int - port nasłuchu.
* startServer() - uruchamia serwer i rozpoczyna wysyłanie danych.
* stopServer() - zatrzymuje serwer.

### 2.3 **DataClient** (warstwa klienta)

**Odpowiedzialność**: Nawiązywanie połączenia z serwerem, odbieranie danych oraz ich buforowanie.

**Główne zadania**:

* Łączy się z serwerem po TCP.
* Cyklicznie odbiera dane pomiarowe.
* Przechowuje dane w wewnętrznym buforze.
* Zapewnia metodę dostępu do odebranych danych innym modułom.

**Kluczowe pola/metody**:

* serverAddress : std::string
* port : int
* dataBuffer : std::vector<double>
* connectToServer()
* receiveData()
* getDataBuffer() : std::vector<double>

### 2.4 **DataProcessor** (interfejs lub klasa bazowa – logika przetwarzania)

**Odpowiedzialność**: Definiowanie wspólnego kontraktu dla wszelkich filtrów / metod przetwarzania danych.

**Główne zadania**:

* Deklaruje metodę processData(), która przyjmuje surowe dane i zwraca dane przetworzone.
* Może zostać rozszerzona o różne filtry

**Kluczowe pola/metody**:

* virtual std::vector<double> processData(const std::vector<double>& inputData) = 0;

**Przykład konkretnej klasy dziedziczącej:** FilterSmooth

* Wykonuje wygładzanie danych w oknie czasowym.
* Metoda processData() implementuje algorytm wygładzenia.

### 2.5 **PlotWidget** (wizualizacja danych)

**Odpowiedzialność**: Prezentacja otrzymanych danych na wykresie w czasie rzeczywistym.

**Główne zadania**:

* Rysuje wykres na podstawie dostarczonych próbek.
* Umożliwia konfigurację wyglądu.
* Oferuje metody do odświeżania wykresu, gdy nadejdą nowe dane.

**Kluczowe pola/metody**:

* updatePlot(const std::vector<double>& data)
* setPlotParameters(const PlotParams& params) (gdzie PlotParams może zawierać np. zakres osi, kolory itp.)