Temat projektu: Pobieranie danych o obecnym i historycznym stanie jakości powietrza ze stacji pomiarowych GIOŚ za pomocą interfejsu API.

Dokumentacja projektowa

1. Użyte klasy i pliki:

    - Klasa `DataProcessor` - jest odpowiedzialna za przetwarzanie danych otrzymanych z API do formatu, jaki będzie akceptować baza danych SQLite. W razie niepowodzenia rzuca wyjątkiem ` DataProcessingError`.

      W przypadku błędu w czasie przetwarzania danych błąd ten jest logowany w pliku `data_parsing.log` bez zwracania wyjątku.

    - Klasa `DataManager` - jest odpowiedzialna za przepływ danych pomiędzy interfejsem użytkownika, a bazą danych oraz API. To przez nią zewnętrzne części programu uzyskują dostęp do danych. Klasa ta w razie niepowodzenia jakiejkolwiek operacji rzuca wyjątkiem `DataManagerError`. Zawiera metody potrzebne do generowania widoków stacji pomiarowych, stanowisk pomiarowych jak również do przygotowania danych dla rysowania wykresu.

      Wszelkie wyjątki napotkane w czasie wykonywania jednej z metod, jak i przebieg niektórych operacji jest logowany do pliku `api.log`.

    - Klasa ` DbManager` - jest odpowiedzialna za połączenie z bazą danych SQLite. Zawiera wszystkie potrzebne metody do pobierania konkretnych informacji o stacjach z bazy danych jak również metody pozwalające zapisywać wszystkie istotne dla działania programu informacje z API. Klasa ta posiada odpowiednie kwerendy do stworzenia bazy danych w przypadku pierwszego uruchomienia programu. W razie niepowodzenia operacji rzuca wyjątkiem `DbManagerError`.

    - Klasa ` GraphDrawer` - jest odpowiedzialna za rysowanie wykresów dla konkretnych danych wybranych przez użytkownika. Jest ona tzw. ‘wrapperem’ dla pewnych funkcjonalności biblioteki *matplotlib.* Ze względu na strukturę w. w. biblioteki klasa ta posiada tylko metody statyczne. Jej głównym zadaniem jest dostosowanie wyglądu wykresu do potrzeb działania aplikacji.

- Klasa ` UiMainWindow` - jest odpowiedzialna za cały interfejs użytkownika (oparty na bibliotece *PyQt5*): wygląd interfejsu oraz wszelkie interakcje. Korzysta jedynie z klas `DbManager` oraz `DataManager` i przechwytuje wszelkie wyjątki przez nie rzucane (zapisuje je w `main_program.log` oraz niektóre z nich wyświetla użytkownikowi). Część elementów tej klasy została wygenerowana automatycznie poprzez generator plików *.py* z plików typu *.ui*.
- Klasy: `WorkerSignals` oraz `ApiDataWorker` - są odpowiedzialne za asynchroniczne pobieranie kluczowych danych z API (dane wszystkich stacji pomiarowych oraz odpowiadające im stanowiska pomiarowe)
- Pliki `helpers.py` oraz `logging_setup.py` zawierają metody pomocnicze do obsługi programu (np. ustawianie relatywnych ścieżek dostępu, konfiguracja logowania błędów), ze względu na małe rozbudowanie tych elementów nie było konieczności tworzenia klas.
- Plik `main.py` odpowiada za uruchomienie programu oraz zaimportowanie wszystkich potrzebnych modułów/klas do jego uruchomienia.

2. Decyzje implementacyjne:

- Jako element przechowujący dane została wybrana baza danych *SQLite*, charakteryzująca się krótkimi czasami dostępu jak i możliwością wykonywania złożonych zapytań.
- Jako element generujący interfejs użytkownika została wybrana biblioteka *PyQt5*, charakteryzująca się rozbudowanym API i łatwością implementacji podstawowej funkcjonalności.
- Do rysowania wykresów wybrano bibliotekę *matplotlib* ze względu na łatwość obsługi i wiele możliwości dostosowań pod wymagania użytkownika.
- Do logowania przebiegu działania aplikacji wykorzystano standardową bibliotekę *logging*. Konfiguracja wszystkich `loggerów` znajduje się w pliku `logging.yaml`.

- Do testowania użyto biblioteki *pytest*. Aby testy przebiegały prawidłowo, w folderze `tests` musi znajdować się plik `data.db` dostarczony w paczce .zip.

3. Napotkane problemy:

- „Zawieszanie się UI" – problem, który nie został rozwiązany we wszystkich przypadkach. W niektórych sytuacjach (np. podczas pobierania dużego pakietu danych) interfejs użytkownika zawiesza się.
- Brak spójności danych – dane dostarczane przez GIOŚ API nie zawsze okazywały się spójne, część z nich okazywała się pusta. W takim przypadku klasa `DataProccesor` ignoruje błąd (tj. nie kończy wykonania programu, a jedynie loguje wyjątek).
- Zachowanie rozmiarów interfejsu użytkownika. Główne okno aplikacji posiada różne wymiary na różnych komputerach (zależne od rozmiaru monitora i rozdzielczości) stąd nie udało się dostosować go niezależnie. Być możne należy skorzystać z mniej oczywistych opcji generowania elementów w bibliotece *PyQt5* (np. responsywność uzyskana użyciem %wymiaru, a nie liczby pikseli)
- Skalowalność interfejsu użytkownika: niestety, przy obecnym kodzie trudno jest dodawać kolejne funkcjonalności w interfejsie. Być może należało zastosować inną strategię (interfejsy dla konkretnych typów obiektów). Niestety wiedza i czas nie pozwoliły na takie dostosowanie.