Algorytmy i struktury danych w języku python

Projekt Zaliczeniowy

Aplikacja do pobierania i wizualizacji cen złota i walut

Imię i nazwisko: Michał Kalinowski
Numer albumu : 22418
Grupa : MZ01IP1

1. Opis Aplikacji

1.1. Opis

Aplikacja jest narzędziem do pobierania oraz wizualizacji cen złota i dolara. Działa w oparciu o dane z API Narodowego Banku Polskiego (link do dokumentacji: https://api.nbp.pl/) i zapisuje je w plikach lokalnych.

Pozwala na pobieranie, przetwarzanie, przechowywanie oraz prezentację danych w interfejsie użytkownika zbudowanym przy użyciu biblioteki Tkinter. Dzięki połączeniu biblioteki Pandas, oraz Matplotlib, aplikacja zapewnia użytkownikowi kompleksowy wgląd w pobrane ceny wybranych zasobów w postaci wykresu.

Aplikacja została napisana w sposób obiektowy, dzięki czemu kod jest modułowy i łatwy do rozbudowy. Logika aplikacji została podzielona na pakiety:

- **model** - model zbieranych danych.
- **service** - warstwa obróbki zebranych danych oraz integracaę z NBP API.
- **view** - warstwa widoku. Pokazanie graficznego interfejsu użytkownika oraz danych w formie logów lub wykresu.

1.2. Technologie

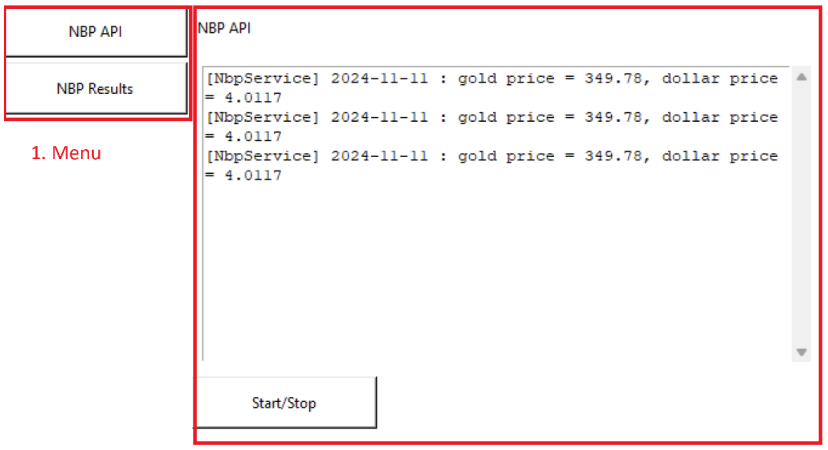
Ze względu na charakter przedmiotu, technologie użyte w projekcie, to python i powiązane z nim biblioteki.

- **Python**: Aplikacja jest napisana w języku Python, który umożliwia szybki rozwój i dużą elastyczność w tworzeniu aplikacji tego typu.
- **Pandas**: Używana do manipulacji danymi finansowymi i ich organizacji w struktury, które umożliwiają łatwe filtrowanie, sortowanie i analizę.
- **Matplotlib**: Biblioteka służąca do generowania wykresów na podstawie przetworzonych danych, zapewniając wizualną reprezentację zmian cen.
- **Tkinter**: Wykorzystywana do budowy interfejsu użytkownika, umożliwia tworzenie elementów takich jak okna, przyciski, pola tekstowe i wykresy, w sposób intuicyjny dla użytkownika.

1.3. Prezentacja aplikacji

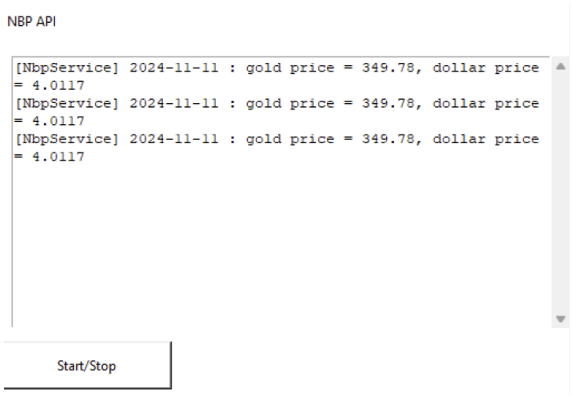
Warstwa widoku aplikacji dzieli się na kilka elementów

1. **Menu** - pozwalające na zmianę widoku.
2. **Widok** - wyświetlający aktualnie wybrany widok.

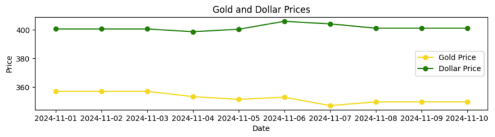


Rys 1 ogólny widok aplikacji

Użytkownik ma do wyboru jeden z dwóch dostępnych widoków:



Rys. 2 NBP API - logger



Rys. 3 widok rezultatów

- **Widok NBP API** pozwala na rozpoczęcie lub zatrzymanie wątku, odpowiedzialnego za zbieranie i prezentację logów.
- **Widok Rezultatów** pozwala na prezentację wyników.

3. Architektura

3.1. Opis

Architektura aplikacji oparta jest na zasadzie modularności i warstwowego podejścia. Struktura kodu jest podzielona na wyraźne komponenty, co ułatwia zarządzanie, rozszerzanie i utrzymanie projektu. Kluczowe elementy aplikacji to:

- warstwa modelu danych
- warstwa serwisów (logika biznesowa i repozytoria)
- warstwa widoku (interfejs użytkownika)

Dzięki zastosowaniu klas abstrakcyjnych i obiektowego podejścia aplikacja jest elastyczna, co umożliwia łatwe dodawanie nowych funkcjonalności, np. kolejnych walut, filtrów dat lub innych źródeł danych.
Aplikacja wykorzystuje współbieżność, do obsługi pobierania danych i rysowania wykresów, jednak podczas wykonywanie aplikacji natrafiłem na ograniczenia w bibliotece Matplotlib, które wywołują dziwne zachowania, przy drukowaniu wykresów w wątku innym niż main. Dla usprawnienia wydajności można rozważyć zmianę biblioteki do generowania wykresów.

3.2. Główne komponenty architektury

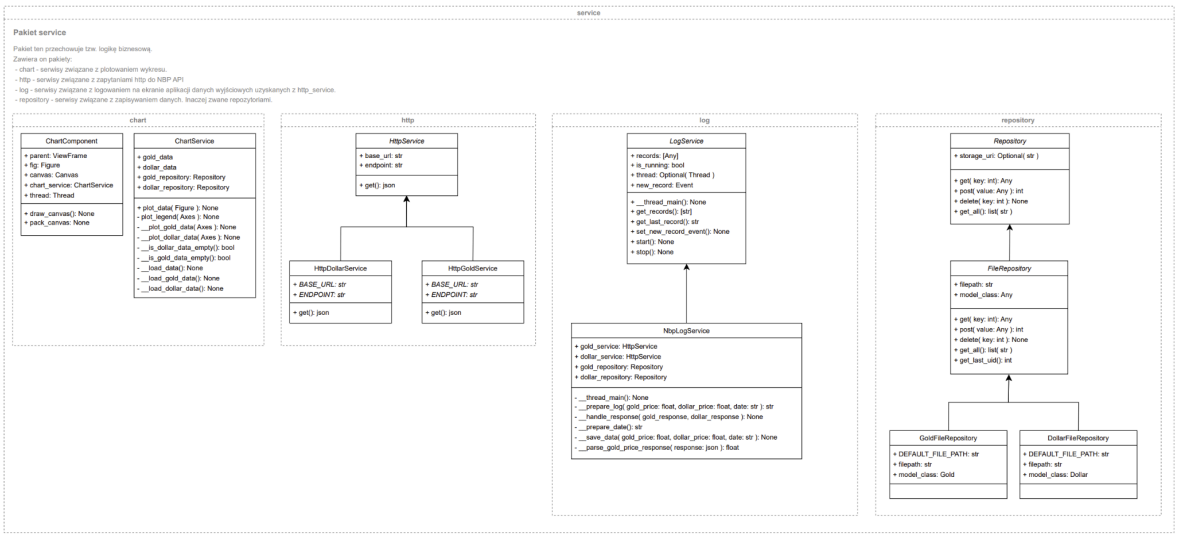
Warstwa Modelu Danych

Ta warstwa zawiera klasy reprezentujące model danych, który jest podstawą dla logiki biznesowej aplikacji. Klasy Dollar i Gold reprezentują rekordy zawierające ceny odpowiednich aktywów. Służy do przechowywania danych o cenach złota i dolara, umożliwiając łatwe mapowanie informacji na odpowiednie obiekty.

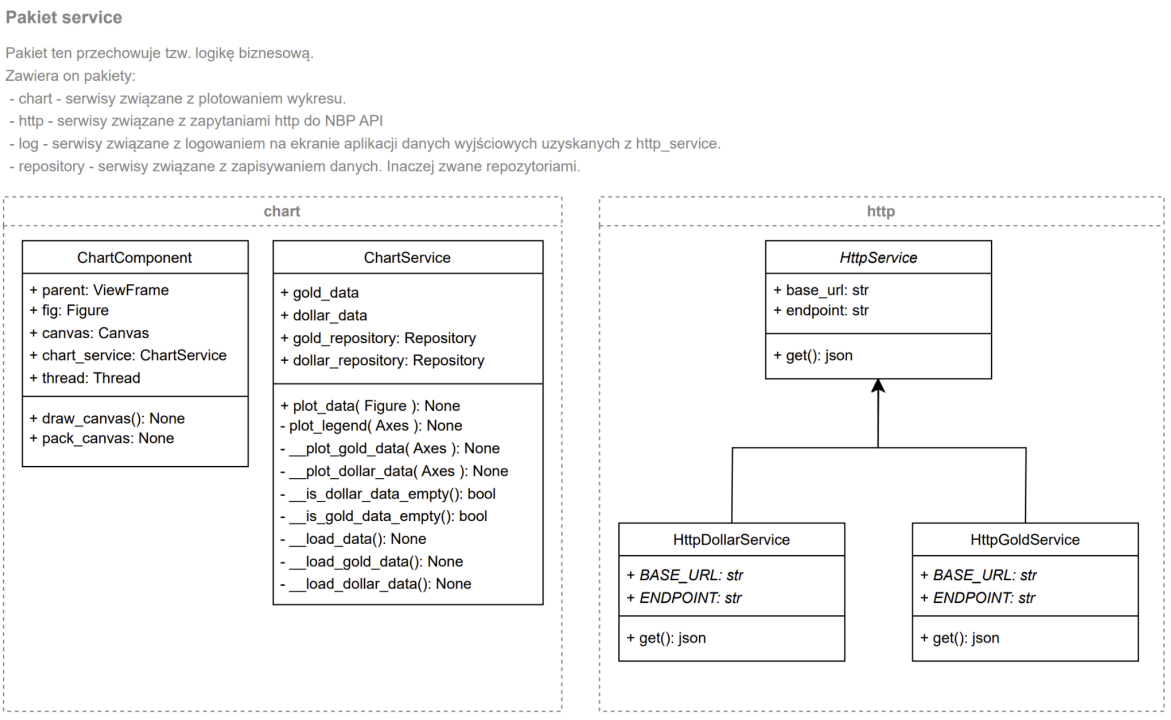
Rys. 4 UML - model

Warstwa Serwisów

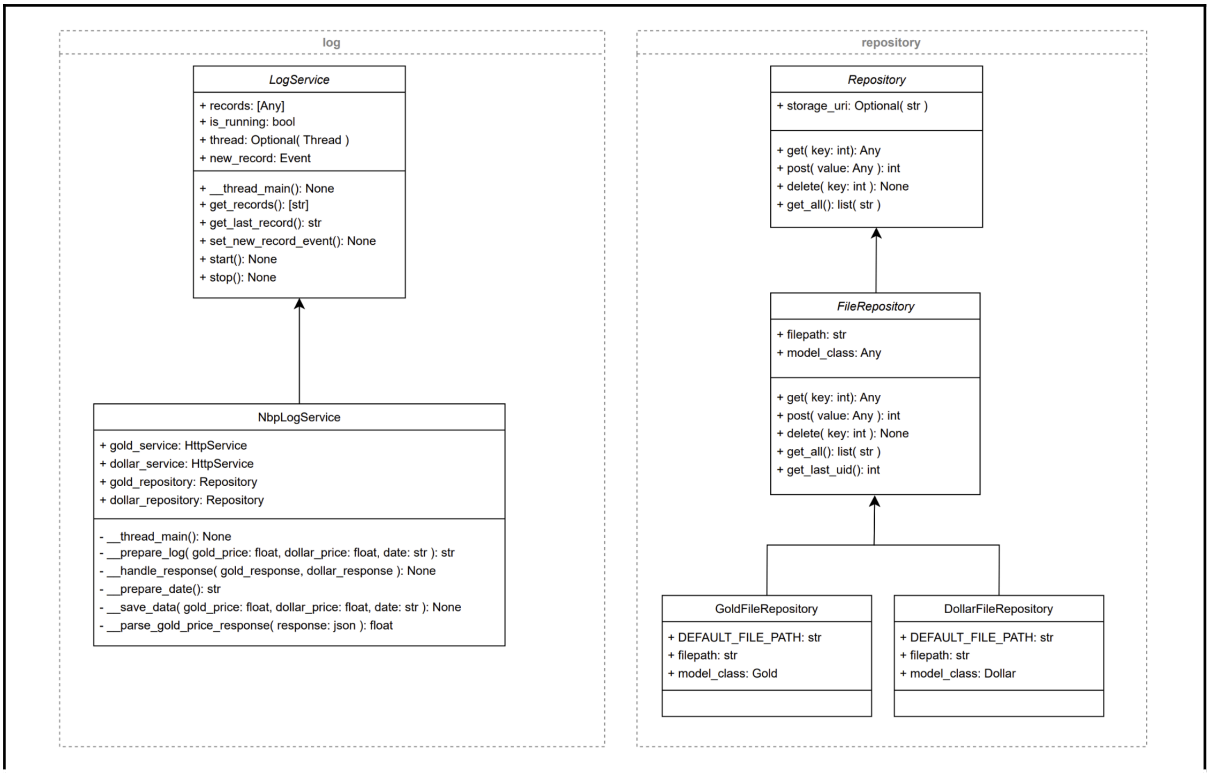
Opis: Ta warstwa odpowiada za logikę biznesową aplikacji i zapis danych, a więc za operacje, które pobierają, przetwarzają i zapisują dane.



Rys. 5 UML - service



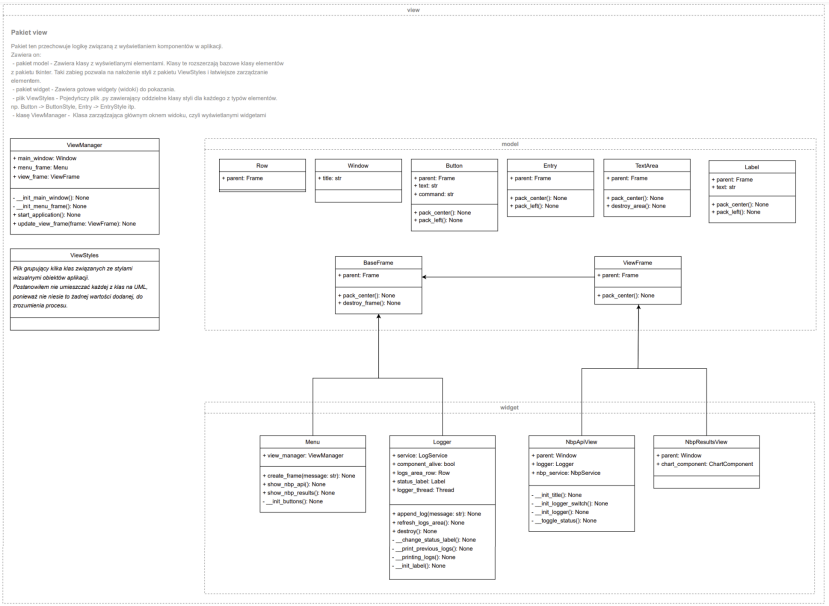
Rys. 6 UML - service - 1/2



Rys. 7 UML - service - 2/2

Warstwa Widoku

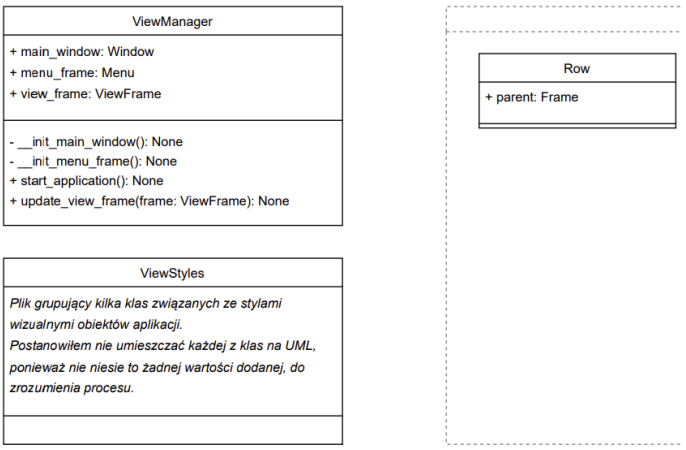
Warstwa ta jest odpowiedzialna za prezentację danych w interfejsie użytkownika. Wykorzystuje bibliotekę Tkinter do tworzenia graficznego interfejsu, w którym użytkownik może przeglądać wykresy i logi oraz nawigować między widokami.



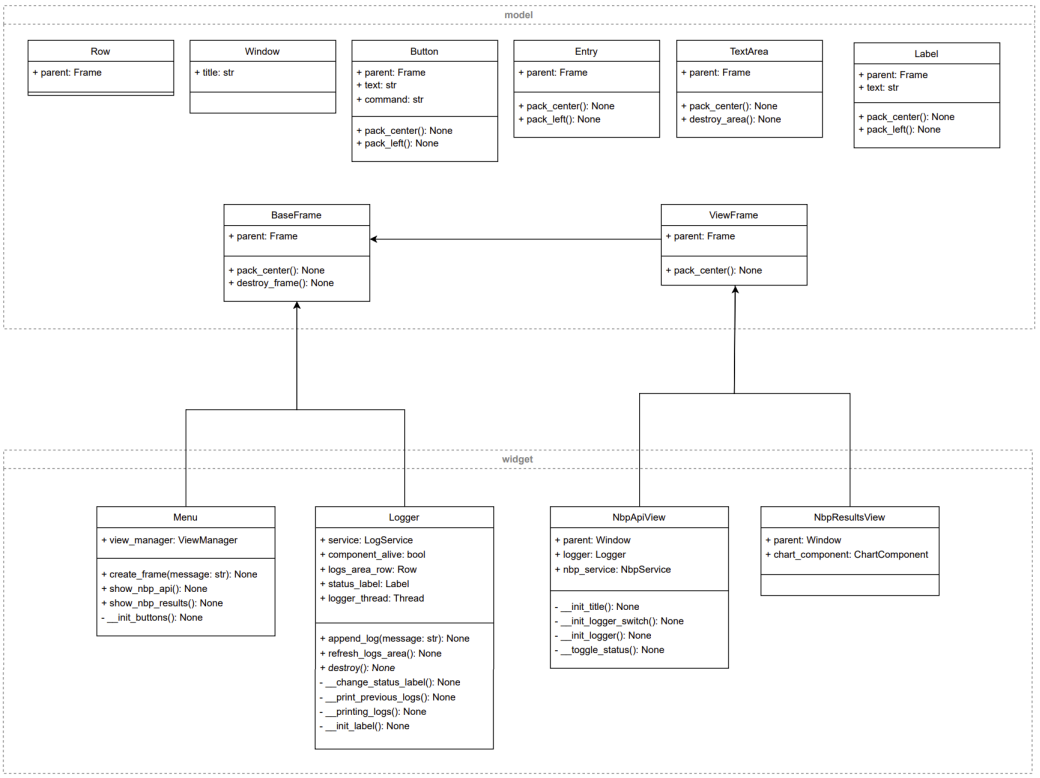
Rys. 8 UML - view

Pakiet view

Pakiet ten przechowuje logikę związaną z wyświetlaniem komponentów w aplikacji.
Zawiera on:
- pakiet model - Zawiera klasy z wyświetlanymi elementami. Klasy te rozszerzają bazowe klasy elementów z pakietu tkinter. Taki zabieg pozwala na nałożenie styli z pakietu ViewStyles i łatwiejsze zarządzanie elementem.
- pakiet widget - Zawiera gotowe widgety (widoki) do pokazania.
- plik ViewStyles - Pojedyńczy plik .py zawierający oddzielne klasy styli dla każdego z typów elementów. np. Button -> ButtonStyle, Entry -> EntryStyle itp.
- klasę ViewManager - Klasa zarządzająca głównym oknem widoku, czyli wyświetlanymi widgetami



Rys. 9 UML - view - 1/2



Rys. 10 UML - view - 2/2

4. Wnioski

Aplikacja do analizy cen złota i dolara stanowi przykład rozwiązania, które dzięki podejściu obiektowemu i wykorzystaniu klas abstrakcyjnych jest wyjątkowo elastyczne i łatwe do rozbudowy.

Sztywne typowanie sprawia, że kod jest łatwiejszy w utrzymaniu oraz wymusza na IDE sprawdzanie typów podczas pisania kodu, co minimalizuje ryzyko błędów i pozwala na skuteczne zarządzanie złożonością aplikacji. Z drugiej strony pythonowy kod sztywno typowany wydaje mi się mniej przejrzysty i brzydszy.

GUI oraz funkcja rysowania wykresów służy głównie celom edukacyjnym, tj. praktyce w zakresie obsługi bibliotek Tkinter, Pandas i Matplotlib.

Ze względu na charakter pracy API NBP, gdzie dane są aktualizowane raz dziennie, zasadnym podejściem byłoby rozdzielenie stworzonej aplikacji na aplikację konsolową, zbierającą dane i oddzielnej aplikacji do ich wyświetlania. Inny możliwy dalszy rozwój aplikacji obejmuje:

- **Zmianę biblioteki do rysowania wykresów** na bardziej zoptymalizowaną pod kątem współbieżności.
- **Dokładną analizę procesu rysowania wykresu** w celu zidentyfikowania wąskich gardeł.
- **Dodanie bazy danych** np. MySql. W celu sprawniejszego i mniej awaryjnego przechowywania.
- **Konteneryzacja aplikacji** np. w dockerze, aby można było ją bez problemów uruchomić na dowolnym serwerz obsługujacym dockera.
- **Dodanie filtrów dat** do widoku nbp api
- **Dodanie innych walut**

Był to mój pierwszy “większy” projekt w pythonie i przyznaję, że python jest językiem, w którym bardzo szybko można napisać działający kod, tzw.POC’a - proof of concept. Przez luźne podejście do określania zwracanych typów i brak kompilatora jest językiem bardzo elastycznym. Uważam to jednak za miecz obusieczny, ponieważ brak powyższych wymagań może skutkować produkcją kodu trudnego w utrzymaniu i zrozumieniu.