Autor: Tymoteusz Dobrzański

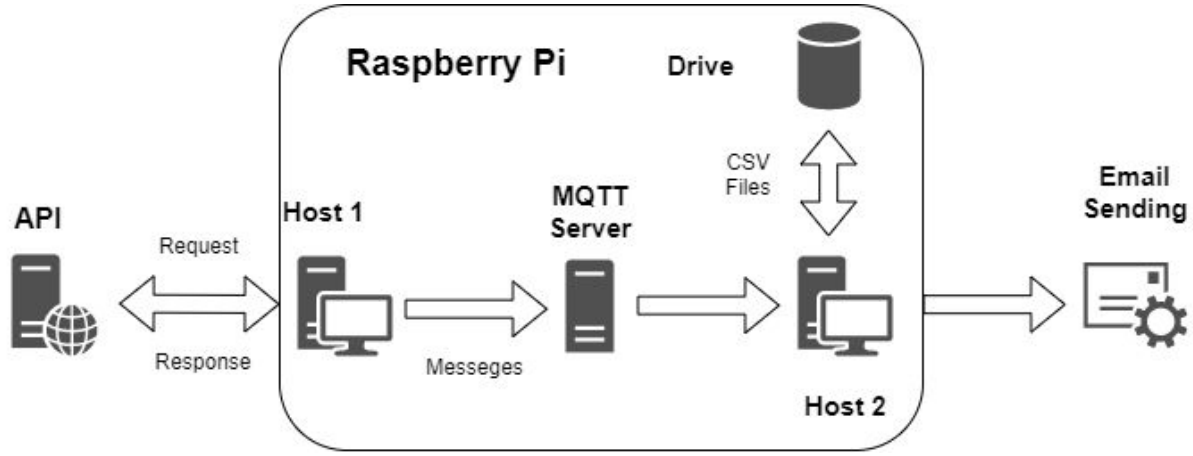

# Dokumentacja projektu: “Zbieranie i wizualizacja danych dotyczących Kryptowalut”

# 1. Ogólny funkcjonalny opis systemu:

System ma na celu zbieranie danych dotyczących wybranych kryptowalut (domyślnie będą to 4 waluty Bitcoin, Ethereum, LiteCoin, Ripple) oraz gromadzenie ich w celu dalszego przetwarzania. Pobieranie danych powinno odbywać się za pomocą API, a ich zbieranie i wstępne przetwarzanie w celu gromadzenia za pomocą protokołu MQTT.

System posiada wbudowany system powiadomień, wysyłający wiadomości email do zapisanych użytkowników.

Schemat systemu:




Zasada działania opiera się na pozyskiwaniu danych z API (Coin Market Cap), po wyciągnięciu istotnych danych, dane są przesyłane do serwera MQTT. Częstotliwość pobierania danych wynosi domyślnie 5 minut, ze względu na ograniczenia API.

Wybrane dane wstępnie przetworzone uzyskane z serwera zostają zapisane lokalnie w plikach csv. Dla każdego dnia i dla każdej waluty powstaje osobny plik.

Zbierane dane:

- symbol - symbol kryptowaluty
- timestamp - czas pobrania danych
- price_USD - cena kryptowaluty w dolarach

Dostępne dane na serwerze MQTT:

- symbol - symbol kryptowaluty
- timestamp - czas pobrania danych
- price_USD - cena kryptowaluty w dolarach
- cmc_rank - ranga przydzielona przez API

- is_fiat - informacja czy dana waluta jest kryptowalutą czy tradycyjną
- name - nazwa kryptowaluty
- percent_change_1h - zmiana ceny w ciągu godziny (w procentach)
- percent_change_24h - zmiana ceny w ciągu dnia (w procentach)
- percent_change_7d - zmiana ceny w ciągu tygodnia (w procentach)

Na koniec dnia system, dla danych zebranych na przestrzeni dnia, tworzy raport rozsyłany do zapisanych użytkowników. Raporty są personalizowane dla każdego użytkownika. Zawierają one skróconą wersję danych, najważniejsze informacje takie jak: cena otwarcia, cena zamknięcia, najwyższa cena w ciągu dnia, najniższa cena w ciągu dnia. Dla każdej kryptowaluty zostają przygotowane dwa wykresy, jeden wykres słupkowy obrazujący zmianę cen na przestrzeni dni oraz wykres liniowy pokazujące zmianę ceny ciągu dnia.

# 2. Przewodnik użytkownika (user's guide):

Zostały przewidziane dwie opcje korzystania z systemu. Jedną z nich jest uruchomienie systemu na własnym urządzeniu (szerzej opisane w rozdziale *“Przewodnik Instalacji”*, jednak dla osób, które nie chcą instalować dodatkowego oprogramowania istnieje możliwość dopisania się do listy osób subskrybujących. Aby to zrobić wystarczy wysłać maila na adres:
zombie.crypto.project@gmail.com
O treści:

Name: Your_Name

Email: example_email@host.com

Type: Add / Remove

Gdzie pole Name oznacza Imię użytkownika, Email jest to adres email na który chcemy otrzymać powiadomienia. W polu type określamy rodzaj prośby, którą wysyłamy. Do wyboru dostępne są dwa typy próśb:

- Add - prośba o dodanie do listy zawierającej adresy email na które będą wysyłane powiadomienia
- Remove - prośba o usunięcie z wyżej podanej listy

# 3. Pełny opis systemu (reference manual):

## Dostępne klasy:

### Pakiet *Data Receiver:*

#### Klasa: *DataReceiver:*

Pola:

*Statyczne:*

*URL_GET_DATA* - źródło danych dotyczących kryptowalut
*URL_CHECK_TOKENS* - źródło danych dotyczących aktualnego limitu API
*AUTHENTICATION_PATH* - ścieżka dostępu do pliku zawierającego dane autoryzujące API

Dynamiczne:

*parameters* - słownik zawierający parametry niezbędne do funkcjonowania API:

'slug' - łańcuch znaków zawierający kryptowaluty odnośnie których aplikacja pobiera dane (domyślnie 'bitcoin,litecoin,ethereum,xrp')
'convert' - waluta na którą API rzutuje cenę kryptowalut

*headers* - wczytane dane autoryzujące API

*session* - połączenie pomiędzy API i aplikacją umożliwiające pobieranie danych

Metody:

*__init__(self, cryptocurrencies="bitcoin,ethereum,litecoin,xrp", convert='USD')*
Konstruktor klasy posiadający dwa parametry:
*cryptocurrencies* - zapisywany do pola 'slug' słownika *parameters*
*convert* - zapisywany do pola '*convert*' słownika *parameters*

*prepare_session(self)*
Funkcja aktywująca połącznie pomiędzy aplikacją i API oraz autoryzująca połączenie między nimi
Zwraca wartość *None*

*get_data(self)*
Funkcja pobierające dane z API dotyczących kryptowalut
Zwraca słownik stworzony z pliku json zwróconego przez API
Schemat słownika
(https://coinmarketcap.com/api/documentation/v1/#operation/getV1CryptocurrencyQuotesLatest)

*request_counter(self, return_type='left)*
Funkcja sprawdzająca ilość pozostałych zapytań do API dla danego dnia
Parametry:
*return_type* - jedno z dwóch 'left' lub 'used' rodzaj zwracanej przez funkcję wartości
Zwraca wartość całkowitoliczbową w zależności od parametru: albo liczbę pozostałych zapytań, albo liczbę wykorzystanych wartości.

## Klasa: MessagesCreator:

Pola:

*data_receiver* - pole tworzące nowy obiekt klasy DataReciver

Metody:

*get_data(self)*
Metoda bez parametrów, pobiera dane przy pomocy klasy DataReceiver oraz wyciągająca z odpowiedzi serwera dane dotyczące wybranych kryptowalut
Metoda zwraca listę słowników zawierających informacje o kryptowalutach oraz czas uzyskania danych.

*data_extractor(self, cryptocurrency)*
Metoda wyciągająca interesujące dane ze słowników otrzymanych z API
Dane:
name - nazwa kryptowaluty
symbol - symbol
cmc_rank - ranga nadan przez CoimMarket API
is_fiat - określa rodzaj waluty (0 tradycyjna, 1 kryptowaluta)
price_USD - cena w dolarach
percent_change_1h - zmiana w ceny w procentach dla okresu 1 godziny
percent_change_24h - zmiana w ceny w procentach dla okresu 24 godzin
percent_change_7d - zmiana w ceny w procentach dla okresu 7 dni
Parametry:
*cryptocurrency* - słownik z API dla którego chcemy uzyskać potrzebne dane
Metoda zwraca słownik z kluczami opisanymi w części "Dane" i wartościami uzyskanymi z API

*messages_constructor(self, dictionary)*
Metoda tworząca na podstawie słownika, wiadomości do serwera MQTT.
Ustawiając klucz jako temat wiadomości oraz wartość jako wartość wiadomości.
Domyślnie Quality of Service jest ustawione na 2
Parametry:
*dictionary* - słownik zawierający pary klucz-wartość do przetworzenia na wiadomości
Metoda zwraca listę krotek.

*publish(self)*

Metoda która po uruchomieniu pobiera dane, przetwarza je i wysyła do serwera mqtt. Korzystając z wcześniej opisanych metod.

*start_loop(self, interval=600)*
Funkcja tworząca niekończącą sie pętle, która w odstępach ustawionych jako interval uruchamia funkcje publish.
Parametr:
interval - domyślnie 600 s, określa odstępy pomiędzy kolejnymi uruchomieniami (w sekundach)

## Pakiet *Data Gathering:*

### Klasa: *MqttClient:*

Pola:

*client* - Klasa pochodząca z modułu paho-mqtt

Metody:

***__init__(self, crypto, host_name)***
Konstruktor klasy, inicjalizujący połączenie pomiędzy klientem a serwerem.
Subskrybuje tematy zbierane i zapisywane w plikach csv, takie jak price_USD, sumbol, timestamp, w tematach odnoszących się do waluty określonej w parametrze crypto.
Parametry:
crypto - waluta którą instancja subskrybuje
host_name - nazwa klienta mqtt

*set_on_connect(self, func)*
Metoda zmieniająca domyślą funkcje uruchamianą w przypadku połączenia, klienta z serwerem.
Parametr:
func - funkcja która zostanie ustawiona jako nowa funkcja

*set_on_message(self, func)*
Metoda zmieniająca domyślą funkcje uruchamianą w przypadku otrzymania wiadomości w którymś z subskrybowanych tematów
Parametr:
func - funkcja która zostanie ustawiona jako nowa funkcja

*loop_start(self)*
Metoda uruchamiająca pętlę w której klient będzie oczekiwał na wiadomości.

### Klasa: Csv*DataPasser:*

Pola:

crypto - kryptowaluta do której odnosi się instancja
row_list - lista pobranych danych które mają być zapisane do pliku
mqtt_client - instancja klasy MqttClient
host_name - nazwa instancji
temp_value_holder - słownik tymczasowo przetrzymujący uzyskane dane
current_time - aktualny czas
output_file_path - ścieżka do zapisu danych

Metody:

*__init__(self, crypto, host_name, output_file_path)*
Konstruktor klasy, inicjalizujący wartości pól, oraz tworząca instancje klasy MqttClient

*add_message(self, topic, payload)*
Metoda dekodująca otrzymane wiadomości oraz zapisująca je do słownika temp_value_holder
Parametry:
topic - temat otrzymanej wiadomości
payload - otrzymana wiadomości

*check_if_full(self)*
Metoda sprawdzająca czy słownik temp_value_holder zawiera wszystkie elementy niezbędne do zapisu. Gdy słownik jest pełen metoda uruchamia funkcje *to_csv*
Zwraca wartość *True* gdy słownik jest pełny albo *False* gdy brakuje któregokolwiek elementu.

*run(self, func, max_time)*
Metoda uruchamiająca klienta mqtt po uprzedniej zmianie funkcji uruchamianej gdy klient otrzyma wiadomość, działa przez określony czas nasłuchując tematów do których klient został zapisany.
Parametry:
func - funkcja, która podmieni domyślną funkcję klienta on_message
max_time - maksymalny czas działania funkcji

*timestamp_processing(self, timestamp, to_print=False)*
Metoda przetwarzania czasu uzyskanego z API.
Parametry:
timestamp - czas uzyskany z API
to_print -wartość boolowska określający czy metoda będzie wypisywać aktualny czas po jego przetworzeniu
Zwraca datę oraz aktualny czas w postaci krotki (godzina, minuta, sekunda)

*to_csv(self, date, output_file_path)*
Metoda zapisująca wartości do pliku csv
Parametry:
date - aktualna data
output_file_path - ścieżka gdzie będą zapisywane pliki csv

## Pakiet *Utilities:*

### Plik functions.py:

Dostępne funkcje:

*mqtt_receiving(crypto, output_file_path)*
Metoda tworząca instancje klasy CsvDataParser i uruchamiają ją, używana w celu tworzenia wielu wątków.
Parametry:
crypto - kryptowaluta dla której będą zbierane dane
output_file_path - ścieżka do zapisu plików wyjściowych

*get_dates(crypto)*
Funkcja zbierające daty dla których powstały pliki csv
Parametry:
crypto - kryptowaluta której pliki funkcja będzie wyszukiwać
Zwraca listę dostępnych dat

*get_output_path(path_type)*
Funkcja zwracająca ścieżkę dostępu do katalogu zawierającego pliki wyjściowe o podanym typie
Parametry:
path_type - typ ścieżki określający rodzaj plików wyjściowych, są to pliki csv lub jpeg (pliki wykresów)

*get_file_path(crypto, date)*
Funkcja generująca ścieżkę zapisu pliku, tworząc jednocześnie nazwę tego pliku.
Parametry:
crypto - kryptowaluta dla której zbierane były dane
date - aktualna data

*get_prices_files(crypto)*
Funkcja zwracająca listę plików zawierających dane dotyczące wybranej kryptowaluty
Parametry:
crypto - wybrana kryptowaluta

*get_last_date()*
Funkcja określająca ostatnią datę dla której zbierane były dane i zwracająca ją

*get_plot_path(date, plot_type, crypto)*

Funkcja generująca ścieżkę zapisu pliku wykresu, tworząc jednocześnie nazwę tego pliku.
Parametry:
crypto - kryptowaluta dla której zbierane były dane
date - aktualna data
plot_type - rodzaj wykresu jaki ma zostać zapisany

*list_to_html_table(list_of_data)*
Funkcja generująca łańcuch znaków tworząc kod html dla tabeli uwzględniając dane podane jako argument funkcji
Parametr:
list_of_data - lista składająca się z list zawierających dane, każda z podlist będzie zapisana do osobnego wiersza w tabeli

*add_subscriber(email, name)*
Funkcja dodająca adres email do listy osób subskrybujących
Parametry:
email - adres email, który zostanie dodany do  listy subskrybujących
name - imię przypisane do tego adres również zapisane w celu personalizacji wiadomości

*remove_subscriber(email, name)*
Funkcja usuwająca adres email z listy osób subskrybujących
Parametry:
email - adres email, który zostanie usunięty z  listy subskrybujących
name - imię przypisane do tego adres również zostanie usunięte

Plik functions_dataframe.py:

Dostępne funkcje:

*get_one_date_dataframe(crypto, date)*
Funkcja generująca Dataframe z danymi zebranymi z jednego dnia dla danej kryptowaluty
Parametry:
crypto - wybrana kryptowaluta
date - data dla której funkcja wygeneruje Dataframe

*get_full_crypto_dataframe(crypto)*
Funkcja przygotowująca Dataframe ze wszystkich dostępnych danych dla danej kryptowaluty.
Parametry:

crypto - wybrana kryptowaluta

*get_daily_dataframe(crypto)*
Funkcja przygotowująca Dataframe zawierający najważniejsze dane dla każdego dnia, dla danej kryptowaluty.
Dane:
‘High’ - najwyższa cena danego dnia
‘Low’ - najniższa cena z dnia
‘Open’ - cena otwarcia
‘Close’ - cena zamknięcia
Parametry:
crypto - wybrana kryptowaluta

Dodatkowe skrypty:

*data_getter.py*
Skrypt pobierający dane, przetwarzający je oraz przesyłający wynik do serwera mqtt. Korzysta w tym celu z klas pakietu Data Receiver

*mqtt_receivers.py*
Skrypt tworzący instancje klientów. Używając wielowątkowości, program generuje odrębny proces dla każdej kryptowaluty. Każdy z tych procesów nasłuchuje odpowiednio dobranych tematów, po zebraniu danych zostaną one dopisane do plików csv. Skrypt korzysta z klas pakietu Data Gathering.

*candle_stick.py*
Skrypt tworzący wykresy odnoszące się do zmian ceny kryptowaluty na przestrzeni dni, dla których zostały zapisane dane. Wykresy po utworzeniu zostają zapisane.
Sposób zapisu:
[symbol_kryptowaluty]_[rodzaj_wykresu]_[data].png
Przykład:
BTC_candle_stick_2021_01_24.png
Wygląd wykresu:


Zielony kolor oznacza wzrost ceny kryptowaluty na przestrzeni dnia, czerowny - spadek. Gruba linia określa różnicę pomiędzy ceną otwarcia, a ceną zamknięcia, natomiast cienką linią oznaczone są ceny minimalna i maksymalna w ciągu dnia.

*line_plot.py*
Skrypt przygotowujący wykresy liniowe dla zmiany ceny kryptowaluty w ciągu jednego dnia.
Schemat zapisu jest taki sam jak w przypadku wykresów słupkowych.
Wygląd wykresu:

*reporting.py*
Skrypt przygotowujący spersonalizowane raporty i wysyłający je do użytkowników zapisanych w liście subskrybujących.
Wygląd raportu:

**zombie.crypto.project@gmail.com**
do mnie

# Hello Tymoteusz!

Here is Your daily Cryptocurrency report.
Results form last day: 2021-01-24

| Cryptocurrency | Low | High | Open | Close |
|---|---|---|---|---|
| BTC | 31106.69 | 32938.77 | 32067.64 | 32347.11 |
| LTC | 134.54 | 142.6 | 137.74 | 141.81 |
| ETH | 1225.87 | 1389.75 | 1230.99 | 1389.75 |
| XRP | 0.27 | 0.28 | 0.27 | 0.27 |

Dodatkowo w załącznikach znajdują się wykresy, po dwa każdej waluty, opisane przy skryptach *candle_stick.py* i *line_plot.py*.

*setup.py*
Skrypt przygotowujący środowisko do uruchomienia systemu.
Wykonywane operacje:

- stworzenie wymaganych katalogów
- przygotowanie plików autentykacyjnych na podstawie danych od użytkownika
- instalacja niezbędnych bibliotek

*email_organizer.py*
Skrypt do zarządzania listą subkeybujących, pobiera widomości email ze skrzynki i przeszukuje ich treść w celu znalezienia podanego schematu. Odpowiednio dodaje lub usuwa użytkowników z listy.
Uwaga! Maile które zostaną przeszukane zostaną przeniesione do kosza.

Wykorzystywane pliki:

Dostępne w katalogu /Data/Input

*subscribers.csv*
Plik zawierający listę zapisanych do newslettera urzytkowników.

*email_form.txt*
Schemat emaila wysyłanego do użytkowników. W przypadku wysyłania email zostaje wcześniej spersonalizowany, dla zebrancyh danych.

*authentication_email.json*
Plik zawierający dane dotyczące maila który obsługuje system wysyłający powiadomienia mailowe. Plik jest niezbędny do uwierzytelnienia użytkownika.

*authentication.json*
Plik zawierający dane uwierzytelnienia dla API.

# 4. Przewodnik instalacji:

Wymagania:

- dowolny serwer MQTT (w tym przypadku Mosqitto)
- konto email (najlepiej gmail)
- konto API (Coin Market Cap)

Proces instalacji:

- Pobranie repozytorium:

git clone https://github.com/ZombiePy/JiBADProject.git

- Skrypt przygotowujący środowisko:

python3 setup.py

Wykorzystane biblioteki:

- `requests`
- `paho-mqtt`
- `pandas`
- `matplotlib`
- `yagmail`
- `mpl_finance`
- `time`
- `os`
- `re`
- `json`
- `sys`
- `getpass`
- `threading`
- `imaplib`

## 5. Kod źródłowy:

Załączony do raportu oraz dostępny na portalu github:
**https://github.com/ZombiePy/JiBADProject**