**Aplikacja aktualizująca ceny produktów sklepu e-commerce używająca REST API do Narodowego Banku Polskiego.**

**Łukasz Ruzicki**

**Gdynia 2022**

Contents

[1. Opis potrzeby użytkownika końcowego 3](#_Toc107486604)

[2. Projekt 4](#_Toc107486605)

[2.1 Aktorzy i przypadki użycia 4](#_Toc107486606)

[2.2 Model danych 4](#_Toc107486607)

[2.3 Projekt interfejsu 4](#_Toc107486608)

[3. Implementacja 5](#_Toc107486609)

[3.1 Baza danych 5](#_Toc107486610)

[3.2 Najważniejsze paczki I moduły - Python 5](#_Toc107486611)

[3.2.1 mysql.connector, 5](#_Toc107486612)

[3.2.2 pandas, openpyxl 5](#_Toc107486613)

[3.2.3 reguest 5](#_Toc107486614)

[3.2.4 logging 5](#_Toc107486615)

[3.2.5 schedule 5](#_Toc107486616)

[3.3 Wymagania funkcjonalne 6](#_Toc107486617)

[3.3.1 Dodanie kolumn UnitPriceUSD i UnitPriceEURO 6](#_Toc107486618)

[3.3.1 Połączenie po REST API do NBP 6](#_Toc107486619)

[3.3.2 Aktualizacja cen 6](#_Toc107486620)

[3.3.1 Generowanie Excela 7](#_Toc107486621)

[3.2.3 Wymagania niefunkcjonalne 8](#_Toc107486622)

[3.2.4 OOP 8](#_Toc107486623)

[3.2.5 Dokumentacja 8](#_Toc107486624)

[3.2.6 Rozwiązanie dostępne na Github 8](#_Toc107486625)

[3.2.7 Logi 8](#_Toc107486626)

[3.2.8 Export schematu SQL 8](#_Toc107486627)

[3.2.9 Obsługa wyjątków 8](#_Toc107486628)

[4 Testy 8](#_Toc107486629)

[4.1 Utworzenie istniejącej kolumny 8](#_Toc107486630)

[4.2 Generowanie pliku Excel w pierwszej kolejności 8](#_Toc107486631)

[4.3 Niepołączenie do NBP 8](#_Toc107486632)

[5. Uruchomienie programu 9](#_Toc107486633)

# Opis potrzeby użytkownika końcowego

Sprzedawca posiada sklep e-commerce z różnymi produktami. Do tej pory handel był realizowany na terenie Polski. Sprzedawca chciałby wysyłać towary do krajów Unii Europejskiej oraz do Stanów Zjednoczonych. Pojawiała się zatem u niego potrzeba akceptacji płatności w dolarach amerykańskich (USD) oraz w Euro. Kupujący musi wiedzieć ile towar kosztuje w danej walucie. Sprzedawca potrzebuje rozwiązania, które cyklicznie raz dziennie lub na żądanie pobierze aktualny kurs walut z Narodowego Banku Polskiego i dokona aktualizacji cen dla produktów w bazie danych.

# Projekt

## Aktorzy i przypadki użycia

Diagram

Description automatically generated

Jedynym aktorem programu jest użytkownik, który będzie miał możliwość uruchomienia codziennej aktualizacji cen, aktualizacji programu jednorazowo lub generowania pliku excel z produktami sklepu. W przypadku uruchomienia aktualizacji codziennej zaleca się uruchomienie to na serwerze.

## 2.2 Model danych

Dane sklepu e-commerce przechowywane są w bazie MySQL. Można edytować je bezpośrednio przez MySQL Workbench przy użciu zapytań SQL lub dokonywać zmian poprzez skrypt Pythona. W przypadku tej aplikacji wszystkie zmiany dokonywane są przez skrypt Pythona.

## 2.3 Projekt interfejsu

W myśl Keep It Simple do uruchomienia programu użyty został terminal. Wybór dokonywany jest poprzez wprowadzenie liczby odpowiadającej interesującej opcji użytkownika. Zaoszczedza to czas przy tworzeniu GUI i w bardzo prosty sposób umozliwia dalszy rozwój interfejsu i dodanie kolejnych opcji.

# Implementacja

## Baza danych

Bza danych została zaimportowany schemat sklepu e-commerce, który jest dostępny pod adresem https://raw.githubusercontent.com/abdelatifsd/E-commerce-Database-

Project/master/3%20-%20Structure.sql przy użyciu MySQL Workbench. Następnie zostały zaimportowane testowe dane, które są dostępne pod adresem https://raw.githubusercontent.com/abdelatifsd/E-commerce-Database-

Project/master/4%20-%20Population.sql

Wszystkie zapytania wykonane zostały jedno po drugim, aby uniknąć ewentualnych problemów z wykonywania ich jednocześnie.

## Najważniejsze paczki I moduły - Python

### mysql.connector,

Umożliwia połaczenie skryptu z bazą danych MySQL. Dzięki niemu możliwe jest wykonywanie zapytań przy użyciu skryptu Python.

### pandas, openpyxl

Dane zostają zaimportowane do Data Frame, czyli struktury danych. Następnie generowany jest plik Excel na podstawie struktury.

### reguest

Biblioteka umożliwająca tworzenie zapytań HTTP przez Pythona.

### logging

Tworzenie pliku z logami informującymi o działaniach programu i jego efektach.

### schedule

Biblioteka umożliwiająca uruchamianie skryptu w określonym czasie, w tym przypadku codziennie.

## Wymagania funkcjonalne

### Dodanie kolumn UnitPriceUSD i UnitPriceEURO

1. def addColumndsTomydb(self):
2. # add coulumns to database
3. try:
4. self.mycursor.execute("ALTER TABLE `mydb`.`Product` ADD `UnitPriceUSD` DECIMAL")
5. except:
6. print("UnitPriceUSD already exists")
7. try:
8. self.mycursor.execute("ALTER TABLE `mydb`.`Product` ADD `UnitPriceEURO` DECIMAL")
9. except:
10. print("UnitPriceEURO already exists")

Dodane zostają kolumny do pliku przed próbą aktualizacji cen. Jeżeli isteniją zostaną wyświetlone komunikaty, nie przerwie to działania programu.

### 3.3.1 Połączenie po REST API do NBP

1. def getDataFromNBPAPI(self):
2. response = requests.get("http://api.nbp.pl/api/exchangerates/tables/a/")
3. if response.status\_code != 200:
4. self.logger.error("NBP API dosen't respond! Prices not updated!")
5. # take data from json file
6. exchange\_info = response.json()
7. for i in exchange\_info[0]['rates']:
8. if i['code'] == 'EUR':
9. current\_euro\_exchange = i['mid']
10. if i['code'] == 'USD':
11. current\_usd\_exchange = i['mid']
13. self.updatePrices(current\_euro\_exchange, current\_usd\_exchange)

Wysyłane jest zapytanie przez REST API do NBP. W przypadku uzyskania odpowiedzi 200, oznaczającej sukces zapytania, zostaje ono zwrócone w frmie pliku json skąd wyciągane są informacje o aktualnych kursach. Dla kluczowych danych wybierane są porządane kwoty.

Zwrócenie innego kodu zapytania niż 200, spowoduje zapis do logu.

### 3.3.2 Aktualizacja cen

1. def updatePrices(self, current\_usd\_exchange, current\_euro\_exchange):
2. # add foreign currency prices to mydb
3. all\_products = self.mycursor.fetchall()
4. for iterate\_product in self.result:
5. foreign\_currency\_prices = (float(iterate\_product[6]) / current\_euro\_exchange, float(iterate\_product[6]) / current\_usd\_exchange, iterate\_product[0])
6. self.mycursor.execute("UPDATE `mydb`.`Product` SET `UnitPriceEURO` = %s, `UnitPriceUSD` = %s WHERE `ProductID` = %s", foreign\_currency\_prices)
7. self.mydb.commit()
8. self.logger.error("UnitPriceEURO and UnitPriceUSD updated")

Ceny przeliczane są zgodnie z kursami. Dane są zamieniane na odpowiednie typy, aby można było wykonać operacje. Typ wyniku pasuje do UnitPrice z bazy danych. Aktualizowane są następnie dane poprzez wykonanie zapytania „UPDATE” dla odpowiednich produktów.

### Generowanie Excela

1. def importToExcel(self, filename = 'products.xlsx'):
2. self.startScheduler()
3. # prepare data to import mydb to dataframe and excel file
4. self.mycursor.execute("SHOW COLUMNS FROM `mydb`.`Product`;")
5. result = self.mycursor.fetchall()
6. column\_names = []
7. column\_index = []
8. # set column order to match the client require order
9. for i in range(11):
10. column\_index.append(i)
11. if i == 6:
12. column\_index += ([12, 13])
13. for i in column\_index:
14. column\_names.append(result[i][0])
15. # make empty dataframe with coulmn names from mydb
16. df = pd.DataFrame(columns = column\_names)
17. product\_data = []
18. self.mycursor.execute("select \* FROM `mydb`.`Product`")
19. result = self.mycursor.fetchall()
20. # make dict with pairs column name: value in client order
21. for product in result:
22. dict = {column\_names[i]: product[column\_index[i]] for i in range(len(column\_index))}
23. df = df.append(dict, ignore\_index = True)
24. # import dataframe to excel file
25. df.to\_excel(filename)

Excel generowany jest na podstawie Data Frame z nazwami kolumn w kolejności ustalonej w wymaganiach przez klienta. Wygenerowany plik może mieć zmienioną nazwę poprzez wprowadzenie argmunetów przy wywołaniu.

## Wymagania niefunkcjonalne

### OOP

Wszystko zostało napisane obiektowo.

### Dokumentacja

Czytasz ją 😊

### Rozwiązanie dostępne na Github

https://github.com/lukruz/Price-updater

### Logi

Logi zapisywane są do pliku „logfile.log”.

### Export schematu SQL

Dostępny w folderze na GitHub

### Obsługa wyjątków

Wszystkie wymagane wyjątki są obsługiwany przy pomocy „try i except”. Dodatkowo zostało to użyte przy menu i w kilku niezbędnych miejsach.

# Testy

## 4.1 Utworzenie istniejącej kolumny

Uruchomienei skryptu dwukrotnie spowoduje dodanie kolumny raz. Nie zostanie ona dodana dwukrotnie, ani nie zostanie pokazany błąd zapytania SQL.

## 4.2 Generowanie pliku Excel w pierwszej kolejności

Wybranie opcji samego generowania pliku Excel w menu poprzez wybranie opcji „3”.

## 4.3 Niepołączenie do NBP

Wpisana zostanie do logów informacja.

Mógłbym opisywać testy manualne, opisać rezultaty, jak to dokładnie testować i jak wynik byłby spodziewany, ale nie ma co się tak rozpisywać w zadaniu rekrutacyjnym.

# 5. Uruchomienie programu

Otworzyć plik dist -> main -> main.exe

Uruchomi się terminal, wyświetli się menu z 3 opcjami. Wprowadzenie innej opcji niż 1, 2, 3 spowoduje powtórzenie zapytania do skutku. Po wykonaniu programu, terminal nie czeka na żaden sygnał i zakańcza program. W zależności od opcji zostanie uruchominy skrypt harmonogramu, jednorazowa aktualizacja lub generowanie pliku excel.