Aplikacja do generowania plików JPK_WB

Jakub Mańko

Repozytorium: https://github.com/jmanko1/JPK_WB-App

Spis treści

1. Cel ćwiczenia

Celem ćwiczenia jest napisanie aplikacji w Pythonie, która będzie generowała pliki XML w formacie JPK_WB. Jest to format pliku, który będzie przechowywał wyciąg bankowy z danego przedziału czasowego i który będzie gotowy do przesłania do Urzędu Skarbowego.

2. Analiza

Program będzie przyjmował trzy pliki źródłowe w formacie CSV. Pierwszy z nich będzie zawierał dane rachunku bankowego. Są to:

- Numer rachunku bankowego
- Kod waluty (obsługiwane: PLN, EUR, USD i CHF)

Plik ten może zawierać dane tylko jednego rachunku bankowego.

Drugi plik CSV będzie zawierał dane podmiotu będącego właścicielem rachunku bankowego. Są to:

- Pełna nazwa podmiotu
- NIP
- REGON (opcjonalny)
- Kod kraju
- Województwo
- Powiat
- Gmina
- Ulica
- Nr domu
- Nr lokalu (opcjonalny)
- Miejscowość
- Kod pocztowy
- Poczta

Plik ten może zawierać dane tylko jednego podmiotu.

Trzeci plik CSV będzie zawierał informacje o operacjach na rachunku bankowym w danym przedziale czasowym, a więc przelewy, lokaty i inne operacje. Dane, które będą przyjmowane, to:

- Data operacji
- Nazwa podmiotu
- Opis operacji
- Kwota operacji (jeżeli uznanie – liczba dodatnia, jeśli obciążenie – liczba ujemna)
- Saldo operacji

Po pomyślnym wczytaniu plików CSV, aplikacja przejdzie do walidacji otrzymanych danych. Wymagania, które muszą zostać spełnione, są następujące:

- Sprawdzenie rachunku bankowego
    - Odpowiednie kolumny
    - Dane dotyczące tylko jednego rachunku bankowego
    - Numer rachunku zawiera 26 cyfr odpowiednio oddzielone od siebie
    - Kod waluty zawiera jedną z czterech obsługiwanych walut (PLN, EUR, USD lub CHF).
- Sprawdzenie podmiotu
    - Odpowiednie kolumny
    - Dane dotyczące tylko jednego podmiotu
    - Pełna nazwa podmiotu jest tekstem i nie jest pusta.
    - NIP nie jest pusty i jest poprawny.
    - Numer REGON jest opcjonalny. Jeśli nie jest pusty, musi być poprawny.
    - Kod kraju jest PL.
    - Województwo jest prawidłowe
    - Powiat i gmina składają się z liter
    - Ulica jest tekstem i nie jest pusta.
    - Numer domu jest liczbą albo liczbą i literą (np. 2A).
    - Numer lokalu jest opcjonalny. Jeżeli nie jest pusty, musi to być liczba.
    - Miejscowość składa się z liter.
    - Kod pocztowy jest poprawny (xx-xxx, gdzie x to cyfra).
    - Poczta składa się z liter
- Sprawdzenie operacji bankowych
    - Odpowiednie kolumny
    - Data operacji jest w poprawnym formacie (rok-miesiąc-dzień).

- Nazwa podmiotu i opis operacji nie są puste.
- Kwota i saldo operacji są liczbami.

Jeśli dane przejdą pomyślnie walidację, program zapyta użytkownika o kod Urzędu Skarbowego, do którego ma zostać potem przesłany wygenerowany plik JPK_WB. Kod Urzędu Skarbowego składa się z czterech cyfr.

Na końcu program zapyta jeszcze użytkownika, gdzie zapisać plik XML i pod jaką nazwą. Następnie aplikacja wygeneruje plik XML w formacie JPK_WB, który będzie można przesłać do Urzędu Skarbowego.

3. Struktura pliku XML w formacie JPK_WB

Jednolity plik kontrolny dla wyciągu bankowego (JPK_WB) składa się z następujących elementów: „**Naglowek**”, „**Podmiot1**”, „**NumerRachunku**”, „**Salda**”, „**WyciagWiersz**”, „**WyciagCtrl**”. Poniżej znajduje się schemat takiego pliku.

https://www.gov.pl/attachment/9f318b30-efd9-446a-8d5b-6ce204381bfd – źródło poniższego schematu.

Nazwa elementu	**Opis elementu**
Naglowek	nagłówek JPK_WB
Podmiot1	podmiot JPK_WB
NumerRachunku	numer IBAN rachunku, którego dotyczy wyciąg
Salda	salda początkowe i końcowe wyciągu
WyciagWiersz	szczegółowe wiersze (zapisy) wyciągu bankowego
WyciagCtrl	sumy kontrolne dla tabeli WyciagWiersz

3.1. Opis struktury elementu Naglowek

Nazwa pola	**Opis pola**
KodFormularza	pole zawiera ustalony (wymagany) kod formularza „JPK_WB” oraz przechowuje dwa atrybuty, którymi obecnie są: kodSystemowy: JPK_WB (1) oraz wersjaSchemy: 1-0
WariantFormularza	pole przechowuje oznaczenie schematu. Obecnie jest to wartość 1
CelZlozenia	pole zawiera wariant: 1 złożenie JPK_WB po raz pierwszy
DataWytworzeniaJPK	data i czas wytworzenia JPK_WB
DataOd	data początkowa okresu, którego dotyczy JPK_WB

DataDo	data końcowa okresu, którego dotyczy JPK_WB
DomyslnyKodWaluty	trzyliterowy kod lokalnej waluty (ISO-4217), domyślny dla wytworzonego JPK_WB
KodUrzędu	czterocyfrowy kod urzędu skarbowego

3.2. Opis struktury elementu Podmiot1

Nazwa pola	**Opis pola**
IdentyfikatorPodmiotu	dane identyfikujące podmiot
AdresPodmiotu	adres podmiotu

3.2.1. Opis struktury elementu IdentyfikatorPodmiotu

Nazwa pola	**Opis pola**
NIP	Identyfikator podatkowy NIP
PelnaNazwa	Pełna nazwa
REGON	Numer REGON (pole opcjonalne)

3.2.2. Opis struktury elementu AdresPodmiotu

Nazwa pola	**Opis pola**
KodKraju	kraj
Wojewodztwo	województwo
Powiat	powiat
Gmina	gmina
Ulica	nazwa ulicy
NrDomu	numer budynku
NrLokalu	numer lokalu (pole opcjonalne)
Miejscowosc	nazwa miejscowości
KodPocztowy	kod pocztowy
Poczta	nazwa urzędu pocztowego

3.3. Opis pola NumerRachunku

Nazwa pola	**Opis pola**
NumerRachunku	numer IBAN rachunku, którego dotyczy wyciąg

3.4. Opis struktury elementu Salda

Nazwa pola	Opis pola
SaldoPoczatkowe	saldo początkowe wyciągu
SaldoKoncowe	saldo końcowe wyciągu

3.5. Opis struktury elementu WyciagWiersz

Nazwa pola	Opis pola
NumerWiersza	kolejny numer wiersza (zapisu) wyciągu
DataOperacji	data operacji
NazwaPodmiotu	nazwa podmiotu będącego stroną operacji
OpisOperacji	opis operacji
KwotaOperacji	kwota operacji
SaldoOperacji	saldo operacji

3.6. Struktura elementu WyciagCtrl

Nazwa pola	Opis pola
LiczbaWierszy	liczba wierszy wyciągu bankowego w okresie, którego dotyczy wyciąg
SumaObciazen	suma kwot obciążeń rachunku w okresie, którego dotyczy wyciąg
SumaUznan	suma kwot uznań rachunku w okresie, którego dotyczy wyciąg

4. Opis pakietu

Do napisania programu używam języka Python w wersji 3.12.2. Będę korzystał z następujących bibliotek:

- pandas - do manipulacji i analizy danych. Służy ona głównie do pracy z danymi w formie tabelarycznej.
- os – do sprawdzenia, czy podane przez użytkownika pliki CSV istnieją w systemie operacyjnym
- warnings – do zignorowania niektórych ostrzeżeń, które mogłyby zostać wypisane w oknie terminala
- biblioteka xml – do wygenerowania odpowiednio sformatowanego drzewa XML, które potem zostanie zapisane do pliku.
- numpy – do pracy ze strukturami danych
- unittest – do przeprowadzenia testów jednostkowych
- re – do wyrażeń regularnych używanych do walidacji niektórych danych

5. Opis algorytmu

Mój program zawiera następującego pliki źródłowe:

- main.py – główny plik projektu
- operacja.py – plik zawierający klasę definiującą operacje bankową
- rachunek.py – plik zawierający klasę definiującą rachunek bankowy
- podmiot.py – plik zawierający klasę definiującą podmiot
- validator.py – plik zawierający kod odpowiedzialny za walidację danych
- tests.py – plik zawierający testy jednostkowe

Na początku opiszę wszystkie zdefiniowane przeze mnie klasy.

5.1. Klasy

Plik operacja.py

```
class Operacja:
  def __init__(self, data_operacji, nazwa_podmiotu, opis_operacji,
kwota_operacji, saldo_operacji):
    self.data_operacji = data_operacji
    self.nazwa_podmiotu = nazwa_podmiotu
    self.opis_operacji = opis_operacji
    self.kwota_operacji = kwota_operacji
    self.saldo_operacji = saldo_operacji

  def __str__(self):
    text = "Data operacji: " + str(self.data_operacji) + "\n"
    text += "Nazwa podmiotu: " + str(self.nazwa_podmiotu) + "\n"
    text += "Opis operacji: " + str(self.opis_operacji) + "\n"
    text += "Kwota operacji: " + str(self.kwota_operacji) + "\n"
    text += "Saldo operacji: " + str(self.saldo_operacji) + "\n"

    return text
```

Plik ten przechowuje klasę Operacja. Każdy obiekt z tej klasy zawiera informacje o pojedynczej operacji na rachunku bankowym.

Klasa Operacja zawiera następujące pola: data_operacji, nazwa_podmiotu, opis_operacji, kwota_operacji i saldo_operacji. Oprócz tego zawiera również dwie metody: __init__(), która inicjuje obiekt określonymi parametrami oraz __string__(), która zamienia obiekt na stringa.

Plik rachunek.py

```
class Rachunek:
  def __init__(self, nrrachunku, kod_waluty):
    self.nrrachunku = nrrachunku
    self.kod_waluty = kod_waluty

  def __str__(self):
    text = "Numer: " + str(self.nrrachunku) + "\n"
```

```
    text += "Kod waluty: " + str(self.kod_waluty) + "\n"

    return text
```

Plik ten przechowuje klasę Rachunek. Obiekt z tej klasy zawiera informacje rachunku bankowym.

Klasa Rachunek zawiera następujące pola: nrrachunku i kod_waluty. Oprócz tego zawiera również dwie metody: __init__(), która inicjuje obiekt określonymi parametrami oraz __string__(), która zamienia obiekt na stringa.

Plik podmiot.py

```
import math

class Podmiot:
  def __init__(self, pelna_nazwa, nip, regon, kod_kraju, wojewodztwo, powiat,
gmina, ulica, nrdomu, nrlokalu, miejscowosc, kod_pocztowy, poczta):
    self.pelna_nazwa = pelna_nazwa
    self.nip = nip
    self.regon = regon
    self.kod_kraju = kod_kraju
    self.wojewodztwo = wojewodztwo
    self.powiat = powiat
    self.gmina = gmina
    self.ulica = ulica
    self.nrdomu = nrdomu
    self.nrlokalu = nrlokalu
    self.miejscowosc = miejscowosc
    self.kod_pocztowy = kod_pocztowy
    self.poczta = poczta

  def __str__(self):
    text = "Pełna nazwa: " + str(self.pelna_nazwa) + "\n"
    text += "NIP: " + str(self.nip) + "\n"
    text += "REGON: " + str(self.regon) + "\n"
    text += "Kod kraju: " + str(self.kod_kraju) + "\n"
    text += "Wojewodztwo: " + str(self.wojewodztwo) + "\n"
    text += "Gmina: " + str(self.gmina) + "\n"
    text += "Ulica: " + str(self.ulica) + "\n"
    text += "Nr domu: " + str(self.nrdomu) + "\n"
    text += "Nr lokalu: " + str(self.nrlokalu) + "\n"
    text += "Miejscowosc: " + str(self.miejscowosc) + "\n"
    text += "Kod pocztowy: " + str(self.kod_pocztowy) + "\n"
    text += "Poczta: " + str(self.poczta)

    return text
```

Plik ten przechowuje klasę Podmiot. Obiekt z tej klasy zawiera informacje podmiocie będącym właścicielem rachunku bankowego.

Klasa Podmiot zawiera następujące pola: pelna_nazwa, nip, regon, kod_kraju, województwo, powiat, gmina, ulica, nrdomu, nrlokalu, miejscowosc, kod_pocztowy i poczta.

Oprócz tego zawiera również dwie metody: __init__(), która inicjuje obiekt określonymi parametrami oraz __string__(), która zamienia obiekt na stringa.

5.2. Plik główny

Teraz zostanie opisany plik main.py, który jest głównym plikiem całego projektu.

5.2.1. Importy

```
import datetime
import os
import warnings
import xml.dom.minidom as md
import xml.etree.ElementTree as ET

import pandas as pd

from src.jpk_classes.operacja import Operacja
from src.jpk_classes.podmiot import Podmiot
from src.jpk_classes.rachunek import Rachunek
from src.validator.validator import validate_podmiot_values,
validate_rachunek_values, validate_operacje_values, \
  validate_kod_urzedu, validate_dataframes
```

Importowane są biblioteki, zdefiniowane przeze mnie klasy oraz funkcje walidujące. Walidacja zostanie opisana później.

5.2.2. Wczytanie plików CSV i utworzenie zbiorów danych

```
#Wczytanie plików zawierających zbiory danych
print("Wczytywanie pliku CSV zawierającego dane podmiotu.")
print("PelnaNazwa,NIP,REGON,KodKraju,Wojewodztwo,Powiat,Gmina,Ulica,NrDomu,NrL
okalu,Miejscowosc,KodPocztowy,Poczta")
podmiot_file = input("Podaj ścieżkę do pliku CSV zawierającego dane podmiotu:
")
if not os.path.exists(podmiot_file):
  print("Podana ścieżka do pliku nie istnieje.")
  exit(1)

df_podmiot = pd.read_csv(podmiot_file)
warnings.filterwarnings("ignore")
df_podmiot.fillna("", inplace=True)
print()
print("Wczytano następujące dane podmiotu:")
print(df_podmiot, end="\n\n")
```

Program wczytuje plik CSV zawierający dane podmiotu. Jeżeli plik nie istnieje, program kończy działanie i wyświetla błąd. W przeciwnym razie, tworzony jest zbiór danych podmiotu. Jeżeli plik CSV zawiera puste komórki, to zostają one wpisane do zbioru danych jako nan (not a numer). Dlatego później wszelkie ewentualne wartości nan w zbiorze zostają zamienione na puste łańcuchy znaków. Ta operacja może wyświetlić ostrzeżenia, dlatego są one ignorowane,

aby nie były wyświetlane w oknie terminala. Na końcu zbiór danych podmiotu jest wyświetlany.

```
print("Wczytywanie pliku CSV zawierającego dane rachunku bankowego.")
print("NumerRachunku,KodWaluty")
rachunek_file = input("Podaj ścieżkę do pliku CSV zawierającego dane rachunku
bankowego: ")
if not os.path.exists(rachunek_file):
  print("Podana ścieżka do pliku nie istnieje.")
  exit(1)

df_rachunek = pd.read_csv(rachunek_file)
df_rachunek.fillna("", inplace=True)
print()
print("Wczytano następujące dane rachunku bankowego:")
print(df_rachunek, end="\n\n")
```

Program wczytuje plik CSV zawierający dane rachunku bankowego. Jeżeli plik nie istnieje, program kończy działanie i wyświetla błąd. W przeciwnym razie, tworzony jest zbiór danych rachunku. Jeżeli plik CSV zawiera puste komórki, to zostają one wpisane do zbioru danych jako nan (not a numer). Dlatego później wszelkie ewentualne wartości nan w zbiorze zostają zamienione na puste łańcuchy znaków. Na końcu zbiór danych rachunku jest wyświetlany.

```
print("Wczytywanie pliku CSV zawierającego dane operacji na rachunku
bankowym.")
print("DataOperacji,NazwaPodmiotu,OpisOperacji,KwotaOperacji,SaldoOperacji")
operacje_file = input("Podaj ścieżkę do pliku CSV zawierającego dane operacji
na rachunku bankowym: ")
if not os.path.exists(operacje_file):
  print("Podana ścieżka do pliku nie istnieje.")
  exit(1)

df_operacje = (pd.read_csv(operacje_file)).sort_values(by="DataOperacji")
df_operacje.fillna("", inplace=True)
print()
print("Wczytano następujące dane operacji na rachunku bankowym:")
print(df_operacje, end="\n\n")

warnings.filterwarnings("default")
```

Program wczytuje plik CSV zawierający dane operacji na rachunku bankowym. Jeżeli plik nie istnieje, program kończy działanie i wyświetla błąd. W przeciwnym razie, tworzony jest zbiór danych operacji. Jeżeli plik CSV zawiera puste komórki, to zostają one wpisane do zbioru danych jako nan (not a numer). Dlatego później wszelkie ewentualne wartości nan w zbiorze zostają zamienione na puste łańcuchy znaków. Następnie zbiór danych operacji jest wyświetlany, a na końcu wyłączane jest ignorowanie ostrzeżeń.

5.2.3. Walidacja struktur zbiorów danych

```
#Walidacja struktur zbiorów danych
if validate_dataframes(df_podmiot, df_rachunek, df_operacje) == 1:
  exit(1)
```

Przeprowadzana jest walidacja struktur zbiorów danych. Program sprawdza, czy zbiory zawierają odpowiednie kolumny, czy df_rachunek zawiera dane tylko jednego rachunku bankowego oraz czy df_podmiot zawiera dane tylko jednego podmiotu

5.2.4. Utworzenie obiektów i listy

```
#Rachunek - dane rachunku bankowego
#Podmiot - dane właściciela rachunku bankowego
#Operacje - operacje na rachunku bankowym

jpk_podmiot = Podmiot(df_podmiot.iloc[0,0], df_podmiot.iloc[0,1],
df_podmiot.iloc[0,2], df_podmiot.iloc[0,3], df_podmiot.iloc[0,4],
df_podmiot.iloc[0,5], df_podmiot.iloc[0,6], df_podmiot.iloc[0,7],
df_podmiot.iloc[0,8], df_podmiot.iloc[0,9], df_podmiot.iloc[0,10],
df_podmiot.iloc[0,11], df_podmiot.iloc[0,12])
jpk_rachunek = Rachunek(df_rachunek.iloc[0,0], df_rachunek.iloc[0,1])

jpk_operacje = list()
for indeks, wiersz in df_operacje.iterrows():
  jpk_operacja = Operacja(wiersz["DataOperacji"], wiersz["NazwaPodmiotu"],
wiersz["OpisOperacji"], wiersz["KwotaOperacji"], wiersz["SaldoOperacji"])
  jpk_operacje.append(jpk_operacja)
```

Następnie tworzone są obiekty ze zdefiniowanych przeze mnie klas. Najpierw tworzony jest obiekt jpk_podmiot klasy Podmiot, który zawiera dane podmiotu, a potem obiekt jpk_rachunek klasy Rachunek, który zawiera dane rachunku bankowego. Następnie tworzona jest lista jpk_operacje, która zawiera obiekty klasy Operacja zawierające dane poszczególnych operacji na rachunku bankowym.

5.2.5. Walidacja danych

```
#Walidacja danych
err1 = validate_podmiot_values(jpk_podmiot)
err2 = validate_rachunek_values(jpk_rachunek)
err3 = validate_operacje_values(jpk_operacje)

if err1 == 1 or err2 == 1 or err3 == 1:
  exit(1)
```

Następnie przeprowadzana jest walidacja danych. Jeżeli któraś z funkcji zwróci wartość 1, to znaczy, że nie wszystkie dane są poprawne. Zostanie wyświetlony komunikat o błędzie, a program zakończy działanie.

5.2.6. Dodatkowe dane

```
kod_urzedu = input("\nPodaj kod urzędu skarbowego: ")
err = validate_kod_urzedu(kod_urzedu)
if err == 1:
  print("Nieprawidłowy kod urzędu skarbowego.")
  exit(1)
```

Potem wczytywany jest od użytkownika kod urzędu skarbowego. Jeżeli nie przejdzie on pomyślnie walidacji, to program wyświetli komunikat i zakończy działanie.

```
suma_uznan = 0.0
```

```
suma_obciazen = 0.0
for operacja in jpk_operacje:
  if operacja.kwota_operacji < 0.0:
    suma_obciazen -= operacja.kwota_operacji
  else:
    suma_uznan += operacja.kwota_operacji

data_od = jpk_operacje[0].data_operacji
data_do = jpk_operacje[-1].data_operacji

data_wytworzenia = (datetime.datetime.now()).strftime("%Y-%m-%d %H:%M:%S")
```

Przed wygenerowaniem drzewa XML, obliczana jest suma uznań i suma obciążeń na rachunku bankowym. Odczytywana jest data początkowa okresu, którego będzie dotyczył plik JPK_WB oraz data końcowa tego okresu. Na końcu odczytywana jest aktualna data i czas.

5.2.7. Generowanie drzewa XML

Jeśli dane przejdą pomyślnie walidację, program generuje drzewo XML, które zostanie zapisane w pliku. Niektóre dane zostaną jeszcze odpowiednio sformatowane, np. liczby zostaną przekształcone na stringi, nazwa województwa zostanie zapisana małymi literami, słowa zawarte w nazwie gminy zostaną zapisane od wielkiej litery, a kwoty i salda operacji zapisane do dwóch miejsc po przecinku.

```
#Generowanie pliku JPK_WB
root = ET.Element("tns:JPK")
root.set("xmlns:etd",
"http://crd.gov.pl/xml/schematy/dziedzinowe/mf/2018/08/24/eD/DefinicjeTypy/")
root.set("xmlns:tns",
"http://crd.gov.pl/xml/schematy/dziedzinowe/mf/2018/08/24/eD/DefinicjeTypy/")
root.set("xmlns:xsi",
"http://crd.gov.pl/xml/schematy/dziedzinowe/mf/2018/08/24/eD/DefinicjeTypy/")
root.set("xsi:schemaLocation",
"http://crd.gov.pl/wzor/2021/12/27/11148/schemat.xsd")
```

Generowanie drzewa XML. Na początku korzeń, czyli element JPK.

```
#Nagłówek pliku JPK_WB
naglowek = ET.SubElement(root, "tns:Naglowek")
```

Element Naglowek.

```
kod_formularza = ET.SubElement(naglowek, "tns:KodFormularza")
kod_formularza.set("type", "tns:TKodFormularza")
kod_formularza.set("kodSystemowy", "JPK_WB (1)")
kod_formularza.set("wersjaSchemy", "1-0")
kod_formularza.text = "JPK_WB"
```

Element KodFormularza.

```
wariant_formularza = ET.SubElement(naglowek, "tns:WariantFormularza")
wariant_formularza.set("type", "xsd:byte")
wariant_formularza.text = "1"

cel_zlozenia = ET.SubElement(naglowek, "tns:CelZlozenia")
```

```
cel_zlozenia.set("type", "tns:TCelZlozenia")
cel_zlozenia.text = "1"

data_wytworzenia_jpk = ET.SubElement(naglowek, "tns:DataWytworzeniaJPK")
data_wytworzenia_jpk.set("type", "etd:TDataCzas")
data_wytworzenia_jpk.text = data_wytworzenia

data_od_jpk = ET.SubElement(naglowek, "tns:DataOd")
data_od_jpk.set("type", "etd:TData")
data_od_jpk.text = data_od

data_do_jpk = ET.SubElement(naglowek, "tns:DataDo")
data_do_jpk.set("type", "etd:TData")
data_do_jpk.text = data_do
```

Elementy WariantFormularza, CelZlozenia, DataWytworzeniaJPK, DataOd i DataDo.

```
domyslny_kod_waluty = ET.SubElement(naglowek, "tns:DomyslnyKodWaluty")
domyslny_kod_waluty.set("type", "kck:currCode_Type")
domyslny_kod_waluty.text = jpk_rachunek.kod_waluty.upper()

kod_urzedu_jpk = ET.SubElement(naglowek, "tns:KodUrzedu")
kod_urzedu_jpk.set("type", "etd:TKodUS")
kod_urzedu_jpk.text = kod_urzedu
```

Elementy DomyslnyKodWaluty i KodUrzedu.

```
#Dane podmiotu, którego dotyczy plik JPK_WB
podmiot1 = ET.SubElement(root, "tns:Podmiot1")
```

Element Podmiot1.

```
identyfikator_podmiotu = ET.SubElement(podmiot1, "tns:IdentyfikatorPodmiotu")
identyfikator_podmiotu.set("type", "etd:TIdentyfikatorOsobyNiefizycznej")

adres_podmiotu = ET.SubElement(podmiot1, "tns:AdresPodmiotu")
adres_podmiotu.set("type", "etd:TAdresPolski")
```

Elementy IdentyfikatorPodmiotu i AdresPodmiotu.

```
nip = ET.SubElement(identyfikator_podmiotu, "etd:NIP")
nip.set("type", "etd:TNrNIP")
nip.text = jpk_podmiot.nip

pelna_nazwa = ET.SubElement(identyfikator_podmiotu, "etd:PelnaNazwa")
pelna_nazwa.set("type", "xsd:token")
pelna_nazwa.text = jpk_podmiot.pelna_nazwa

regon = ET.SubElement(identyfikator_podmiotu, "etd:REGON")
regon.set("type", "etd:TNrREGON")
regon.text = jpk_podmiot.regon
```

Elementy NIP, PelnaNazwa I REGON.

```
kod_kraju = ET.SubElement(adres_podmiotu, "etd:KodKraju")
```

```
kod_kraju.set("type", "etd:TKodKraju")
kod_kraju.text = jpk_podmiot.kod_kraju.upper()

wojewodztwo = ET.SubElement(adres_podmiotu, "etd:Wojewodztwo")
wojewodztwo.set("type", "etd:TJednAdmin")
wojewodztwo.text = jpk_podmiot.wojewodztwo.lower()

powiat = ET.SubElement(adres_podmiotu, "etd:Powiat")
powiat.set("type", "etd:TJednAdmin")
zmienione_slowa = [slowo.capitalize() for slowo in jpk_podmiot.powiat.split()]
jpk_podmiot.powiat = ' '.join(zmienione_slowa)
powiat.text = jpk_podmiot.powiat

gmina = ET.SubElement(adres_podmiotu, "etd:Gmina")
gmina.set("type", "etd:TJednAdmin")
zmienione_slowa = [slowo.capitalize() for slowo in jpk_podmiot.gmina.split()]
jpk_podmiot.gmina = ' '.join(zmienione_slowa)
gmina.text = jpk_podmiot.gmina

ulica = ET.SubElement(adres_podmiotu, "etd:Ulica")
ulica.set("type", "etd:TUlica")
zmienione_slowa = [slowo.capitalize() for slowo in jpk_podmiot.ulica.split()]
jpk_podmiot.ulica = ' '.join(zmienione_slowa)
ulica.text = jpk_podmiot.ulica

nrdomu = ET.SubElement(adres_podmiotu, "etd:NrDomu")
nrdomu.set("type", "etd:TNrBudynku")
nrdomu.text = str(jpk_podmiot.nrdomu).upper()

nrlokalu = ET.SubElement(adres_podmiotu, "etd:NrLokalu")
nrlokalu.set("type", "etd:TNrLokalu")
nrlokalu.text = str(jpk_podmiot.nrlokalu)
```

Elementy KodKraju, Wojewodztwo, Powiat, Gmina, Ulica, NrDomu i NrLokalu. Kod kraju zapisywany jest wielkimi literami, nazwa województwa zapisywana jest małymi literami, słowa zawarte w nazwie powiatu, gminy i ulicy zapisywane są od wielkiej litery, a ewentualna litera zawarta w numerze domu zapisywana jest jako wielka.

```
miejscowosc = ET.SubElement(adres_podmiotu, "etd:Miejscowosc")
miejscowosc.set("type", "etd:TMiejscowosc")
zmienione_slowa = [slowo.capitalize() for slowo in
jpk_podmiot.miejscowosc.split()]
jpk_podmiot.miejscowosc = ' '.join(zmienione_slowa)
miejscowosc.text = jpk_podmiot.miejscowosc

kod_pocztowy = ET.SubElement(adres_podmiotu, "etd:KodPocztowy")
kod_pocztowy.set("type", "etd:TKodPocztowy")
kod_pocztowy.text = jpk_podmiot.kod_pocztowy

poczta = ET.SubElement(adres_podmiotu, "etd:Poczta")
```

```
poczta.set("type", "etd:TMiejscowosc")
zmienione_slowa = [slowo.capitalize() for slowo in jpk_podmiot.poczta.split()]
jpk_podmiot.poczta = ' '.join(zmienione_slowa)
poczta.text = jpk_podmiot.poczta
```

Elementy Miejscowosc, KodPocztowy i Poczta. Słowa zawarte w nazwie miejscowości i poczty są zapisywane od wielkiej litery.

```
#Numer rachunku bankowego
numer_rachunku = ET.SubElement(root, "tns:NumerRachunku")
numer_rachunku.set("type", "xsd:string")
numer_rachunku.text = jpk_rachunek.nrrachunku
```

Element NumerRachunku.

```
#Saldo początkowe i końcowe wyciągu
saldo1 = jpk_operacje[0].saldo_operacji - jpk_operacje[0].kwota_operacji
saldo1 = "{:.2f}".format(saldo1)
saldo2 = "{:.2f}".format(jpk_operacje[-1].saldo_operacji)

salda = ET.SubElement(root, "tns:Salda")

saldo_pocz = ET.SubElement(salda, "tns:SaldoPoczatkowe")
saldo_pocz.set("type", "tns:TKwotowy")
saldo_pocz.text = saldo1

saldo_kocz = ET.SubElement(salda, "tns:SaldoKoncowe")
saldo_kocz.set("type", "tns:TKwotowy")
saldo_kocz.text = saldo2
```

Elementy SaldoPoczatkowe i SaldoKoncowe. Najpierw wczytywane są oba salda, a potem zapisywane we właściwym formacie (część całkowita i dwie cyfry po przecinku) w drzewie XML.

```
#Operacje na rachunku bankowym
#Operacje na rachunku bankowym
n_wierszy = 0

for jpk_operacja in jpk_operacje:
  wyciag_wiersz = ET.SubElement(root, "tns:WyciagWiersz")

  numer_wiersza = ET.SubElement(wyciag_wiersz, "tns:NumerWiersza")
  numer_wiersza.set("type", "tns:TNaturalnyJPK")
  numer_wiersza.text = str(n_wierszy + 1)

  data_operacji = ET.SubElement(wyciag_wiersz, "tns:DataOperacji")
  data_operacji.set("type", "etd:TData")
  data_operacji.text = jpk_operacja.data_operacji

  nazwa_podmiotu = ET.SubElement(wyciag_wiersz, "tns:NazwaPodmiotu")
  nazwa_podmiotu.set("type", "tns:TZnakowyJPK")
  nazwa_podmiotu.text = jpk_operacja.nazwa_podmiotu
```

```
  opis_operacji = ET.SubElement(wyciag_wiersz, "tns:OpisOperacji")
  opis_operacji.set("type", "tns:TZnakowyJPK")
  opis_operacji.text = jpk_operacja.opis_operacji

  kwota_operacji = ET.SubElement(wyciag_wiersz, "tns:KwotaOperacji")
  kwota_operacji.set("type", "tns:TKwotowy")
  kwota_operacji.text = "{:.2f}".format(jpk_operacja.kwota_operacji)

  saldo_operacji = ET.SubElement(wyciag_wiersz, "tns:SaldoOperacji")
  saldo_operacji.set("type", "tns:TKwotowy")
  saldo_operacji.text = "{:.2f}".format(jpk_operacja.saldo_operacji)

  n_wierszy += 1
```

Pętla for zapisująca po kolei każdą operację bankową w każdym kolejnym elemencie WyciagWiersz w drzewie XML. Każdy element WyciagWiersz reprezentuje osobną operację bankową.

```
#Dane kontrolne pliku JPK_WB
wyciag_ctrl = ET.SubElement(root, "tns:WyciagCtrl")

liczba_wierszy = ET.SubElement(wyciag_ctrl, "tns:LiczbaWierszy")
liczba_wierszy.set("type", "tns:TNaturalnyJPK")
liczba_wierszy.text = str(n_wierszy)

sumaObciazen = ET.SubElement(wyciag_ctrl, "tns:SumaObciazen")
sumaObciazen.set("type", "tns:TKwotowy")
sumaObciazen.text = "{:.2f}".format(suma_obciazen)

sumaUznan = ET.SubElement(wyciag_ctrl, "tns:SumaUznan")
sumaUznan.set("type", "tns:TKwotowy")
sumaUznan.text = "{:.2f}".format(suma_uznan)
```

Element WyciagCtrl zawierający dane kontrolne dla pliku JPK_WB (liczba wierszy, suma obciążeń i suma uznań).

5.2.8. Zapis pliku JPK_WB

```
#Zapis pliku JPK_WB
file_name = input("\nPodaj nazwę pliku XML, do którego zostanie zapisany
wyciąg bankowy: ")
xmlstr = md.parseString(ET.tostring(root)).toprettyxml(indent="\t")
xmlstr_lines = xmlstr.splitlines(True)
xmlstr_lines = xmlstr_lines[1:]

if xmlstr_lines and xmlstr_lines[-1].endswith('\n'):
    xmlstr_lines[-1] = xmlstr_lines[-1].rstrip('\n')

# Zapisanie do pliku
with open(file_name + ".xml", "w", newline='', encoding='utf-8') as f:
  f.writelines(xmlstr_lines)
```

```
print("\nPlik JPK został pomyślnie utworzony")
```
Po utworzeniu drzewa XML, jest ono zapisywane do pliku JPK_WB. Program pyta o nazwę docelowego pliku XML, a następnie formatuje odpowiednio drzewo XML tak, aby wszystkie jego elementy nie zostały zapisane w jednej linii w pliku i żeby było ono czytelne dla użytkownika. Każdy kolejny element drzewa jest zapisywany w następnej linii pliku i jeżeli jest dzieckiem poprzedniego elementu, to dodawany jest przed nim tabulator.

Funkcja formatująca drzewo XML dodaje jeszcze do niego na początku deklarację XML. Program usuwa tę deklarację przed zapisaniem drzewa do pliku. Dodatkowo, program upewnia się, że plik XML nie będzie miał na samym końcu pustej linii po zapisaniu do niego wygenerowanego drzewa.

5.3. Plik walidujący

Teraz zostanie opisany plik validator.py, który odpowiada za walidację wczytanych danych.

5.3.1. Importy

```
import re
from datetime import datetime

import numpy as np
```

5.3.2. Sprawdzanie poprawności struktur zbiorów danych

```
def check_podmiot_columns(df_podmiot):
    columns = ['PelnaNazwa', 'NIP', 'REGON', 'KodKraju', 'Wojewodztwo',
'Powiat', 'Gmina', 'Ulica', 'NrDomu',
               'NrLokalu', 'Miejscowosc', 'KodPocztowy', 'Poczta']
    df_columns = list(df_podmiot.columns)
    return df_columns == columns
```
Funkcja check_podmiot_columns() sprawdza, czy zbiór danych podmiotu zawiera w odpowiedniej kolejności kolumny PelnaNazwa, NIP, REGON, KodKraju, Wojewodztwo, Powiat, Gmina, Ulica, NrDomu, NrLokalu, Miejscowosc, KodPocztowy i Poczta.

```
def check_rachunek_columns(df_rachunek):
    columns = ['NumerRachunku', 'KodWaluty']
    df_columns = list(df_rachunek.columns)
    return columns == df_columns
```
Funkcja check_rachunek_columns() sprawdza, czy zbiór danych rachunku bankowego zawiera w odpowiedniej kolejności kolumny NumerRachunku i KodWaluty.

```
def check_operacje_columns(df_operacje):
    columns = ['DataOperacji', 'NazwaPodmiotu', 'OpisOperacji',
'KwotaOperacji', 'SaldoOperacji']
    df_columns = list(df_operacje.columns)
    return df_columns == columns
```
Funkcja check_operacje_columns() sprawdza, czy zbiór danych operacji zawiera w odpowiedniej kolejności kolumny DataOperacji, NazwaPodmiotu, OpisOperacji, KwotaOperacji i SaldoOperacji.

```
def is_one_row(df):
```

```
    return df.shape[0] == 1
```
Funkcja is_one_row() sprawdza, czy zbiór danych zawiera tylko i wyłącznie jeden wiersz.

```
def validate_dataframes(df_podmiot, df_rachunek, df_operacje):
    if not check_podmiot_columns(df_podmiot):
        print("Plik CSV z danymi podmiotu nie zawiera poprawnych kolumn.")
        return 1

    if not check_rachunek_columns(df_rachunek):
        print("Plik CSV z danymi rachunku bankowego nie zawiera poprawnych
kolumn.")
        return 1

    if not check_operacje_columns(df_operacje):
        print("Plik CSV z danymi operacji bankowych nie zawiera poprawnych
kolumn.")
        return 1

    if not is_one_row(df_podmiot):
        print("Plik CSV z danymi podmiotu zawiera dane więcej niż jednego
podmiotu.")
        return 1

    if not is_one_row(df_rachunek):
        print("Plik CSV z danymi rachunku bankowego zawiera dane więcej niż
jednego rachunku bankowego.")
        return 1

    return 0
```
Funkcja validate_dataframes() otrzymuje zbiory danych i wywołuje funkcje walidujące struktury tych zbiorów. Jeżeli jakiś etap walidacji nie zakończy się sukcesem, to funkcja validate_dataframes() zwróci do pliku głównego wartość 1, co w konsekwencji zakończy program z błędem.

5.3.3. Sprawdzanie poprawności danych

Walidacja danych podmiotu

```
def is_not_empty_str(mystr):
    return isinstance(mystr, str) and mystr != ""
```
Funkcja is_not_empty_str() sprawdza, czy otrzymana zmienna jest stringiem oraz czy nie jest ona pusta. Służy ona do walidacji pełnej nazwy podmiotu i ulicy.

```
def is_nip_correct(nip):
    return isinstance(nip, str) and re.fullmatch(r'\d{3}-\d{3}-\d{2}-\d{2}',
nip)
```
Funkcja is_nip_correct() sprawdza, czy NIP jest stringiem oraz czy jest w formacie xxx-xxx-xx-xx, gdzie x to cyfra.

```
def is_regon_correct(regon):
```

```
    return isinstance(regon, str) and (re.fullmatch(r"^\d{2} \d{6} \d$",
regon) or re.fullmatch(r"^\d{2} \d{6} \d{1} \d{4} \d{1}$", regon) or regon ==
"")
```

Funkcja is_regon_correct() sprawdza, czy numer REGON jest stringiem oraz jeżeli nie jest on pusty, to czy jest w formacie xx xxxxxx x lub xx xxxxxx x xxxx, gdzie x to cyfra.

```
def is_cc_correct(kod_kraju):
    return kod_kraju.upper() == "PL"
```

Funkcja is_cc_correct() sprawdza, czy kod kraju jest polski.

```
def is_wojewodztwo_correct(wojewodztwo):
    wojewodztwa = [
        'dolnośląskie',
        'kujawsko-pomorskie',
        'lubelskie',
        'lubuskie',
        'łódzkie',
        'małopolskie',
        'mazowieckie',
        'opolskie',
        'podkarpackie',
        'podlaskie',
        'pomorskie',
        'śląskie',
        'świętokrzyskie',
        'warmińsko-mazurskie',
        'wielkopolskie',
        'zachodniopomorskie'
    ]
    return wojewodztwo.lower() in wojewodztwa
```

Funkcja is_wojewodztwo_correct() sprawdza, czy województwo jest prawidłowe.

```
def is_adres_element_correct(element):
    return isinstance(element, str) and re.fullmatch(r"[a-zA-
ZąćęłńóśźżĄĆĘŁŃÓŚŹŻ\s]+", element)
```

Funkcja is_adres_element_correct() sprawdza, czy zmienna jest stringiem i składa się z liter. Służy ona do walidacji powiatu, gminy, miejscowości i poczty.

```
def is_nrdomu_correct(nrdomu):
    return isinstance(nrdomu, np.int64) or isinstance(nrdomu, str) and
re.fullmatch(r"^\d+[a-zA-Z]?$", nrdomu)
```

Funkcja is_nrdomu_correct() sprawdza, czy numer domu jest liczbą całkowitą albo jest w formacie xy, gdzie x to liczba całkowita, a y to litera, np. 25A lub 5B.

```
def is_nrlokalu_correct(nrlokalu):
    return isinstance(nrlokalu, np.int64) or nrlokalu == ""
```

Funkcja is_nrlokalu_correct() sprawdza, czy numer lokalu, jeżeli został podany, jest liczbą całkowitą.

```
def is_kod_pocztowy_correct(kod_pocztowy):
```

```
    return isinstance(kod_pocztowy, str) and re.fullmatch(r"^\d{2}-\d{3}$",
kod_pocztowy)
```

Funkcja is_kod_pocztowy_correct() sprawdza, czy kod pocztowy jest stringiem oraz czy jest w formacie xx-xxx, gdzie x to cyfra.

```
def validate_podmiot_values(jpk_podmiot):
    if not is_not_empty_str(jpk_podmiot.pelna_nazwa):
        print("Niepoprawna pełna nazwa podmiotu.")
        return 1

    if not is_nip_correct(jpk_podmiot.nip):
        print("Niepoprawny NIP podmiotu.")
        return 1

    if not is_regon_correct(jpk_podmiot.regon):
        print("Niepoprawny REGON podmiotu.")
        return 1

    if not is_cc_correct(jpk_podmiot.kod_kraju):
        print("Aplikacja nie obsługuje podmiotów z innych krajów niż Polska
(PL).")
        return 1

    if not is_wojewodztwo_correct(jpk_podmiot.wojewodztwo):
        print("Niepoprawne województwo podmiotu.")
        return 1

    if not is_adres_element_correct(jpk_podmiot.powiat):
        print("Nieprawidłowa nazwa powiatu podmiotu.")
        return 1

    if not is_adres_element_correct(jpk_podmiot.gmina):
        print("Nieprawidłowa nazwa gminy podmiotu.")
        return 1

    if not is_not_empty_str(jpk_podmiot.ulica):
        print("Nieprawidłowa nazwa ulicy podmiotu.")
        return 1

    if not is_nrdomu_correct(jpk_podmiot.nrdomu):
        print("Nieprawidłowy numer domu podmiotu.")
        return 1

    if not is_nrlokalu_correct(jpk_podmiot.nrlokalu):
        print("Nieprawidłowy numer lokalu podmiotu.")
        return 1

    if not is_adres_element_correct(jpk_podmiot.miejscowosc):
        print("Nieprawidłowa nazwa miejscowości podmiotu.")
```

```
        return 1

    if not is_kod_pocztowy_correct(jpk_podmiot.kod_pocztowy):
        print("Nieprawidłowy kod pocztowy podmiotu.")
        return 1

    if not is_adres_element_correct(jpk_podmiot.poczta):
        print("Nieprawidłowa nazwa poczty podmiotu.")
        return 1

    return 0
```

Funkcja validate_podmiot_values() otrzymuje obiekt klasy Podmiot zawierający dane podmiotu i wywołuje funkcje walidujące atrybuty obiektu. Jeżeli jakiś etap walidacji nie zakończy się sukcesem, to funkcja validate_podmiot_values() zwróci do pliku głównego wartość 1, co w konsekwencji zakończy program z błędem.

Walidacja danych rachunku bankowego

```
def is_nrrachunku_correct(nrrachunku):
    return isinstance(nrrachunku, str) and re.fullmatch(r"^\d{2} \d{4} \d{4} \d{4} \d{4} \d{4} \d{4}$", nrrachunku)
```

Funkcja is_nr_rachunku_correct() sprawdza, czy numer rachunku bankowego jest stringiem w formacie xx xxxx xxxx xxxx xxxx xxxx xxxx, gdzie x to cyfra.

```
def is_waluta_correct(kod_waluty):
    waluty = ["PLN", "EUR", "CHF", "USD"]
    return kod_waluty.upper() in waluty
```

Funkcja is_waluta_correct() sprawdza, czy waluta jest jedną z czterech obsługiwanych: PLN, EUR, CHF, USD.

```
def validate_rachunek_values(jpk_rachunek):
    if not is_nrrachunku_correct(jpk_rachunek.nrrachunku):
        print("Nieprawidłowy numer rachunku.")
        return 1

    if not is_waluta_correct(jpk_rachunek.kod_waluty):
        print("Nieobsługiwana waluta. Program obsługuje PLN, EUR, USD i CHF.")
        return 1

    return 0
```

Funkcja validate_rachunek_values() otrzymuje obiekt klasy Rachunek zawierający dane rachunku bankowego i wywołuje funkcje walidujące atrybuty obiektu. Jeżeli jakiś etap walidacji nie zakończy się sukcesem, to funkcja validate_rachunek_values() zwróci do pliku głównego wartość 1, co w konsekwencji zakończy program z błędem.

Walidacja operacji na rachunku bankowym

```
def is_not_empty(a):
    return isinstance(a, np.int64) or isinstance(a, str) and a != ""
```

Funkcja is_not_empty() sprawdza, czy zmienna jest liczbą całkowitą albo stringiem, który nie jest pusty. Służy ona do walidacji nazwy podmiotu operacji i opisu operacji.

```
def is_money_amount_correct(amount):
    try:
        amount = float(amount)
    except ValueError:
        return False

    return round(amount, 2) == amount
```

Funkcja is_money_amount_correct() sprawdza, czy zmienna jest poprawną kwotą pieniężną. Służy ona do walidacji kwoty i salda operacji.

```
def validate_operacje_values(jpk_operacje):
    for jpk_operacja in jpk_operacje:
        if not is_date_correct(jpk_operacja.data_operacji):
            print("Niepoprawna data operacji w następującej operacji:")
            print(jpk_operacja)
            return 1

        if not is_not_empty(jpk_operacja.nazwa_podmiotu):
            print("Brak nazwy podmiotu w następującej operacji:")
            print(jpk_operacja)
            return 1

        if not is_not_empty(jpk_operacja.opis_operacji):
            print("Brak opisu operacji w następującej operacji:")
            print(jpk_operacja)
            return 1

        if not is_money_amount_correct(jpk_operacja.kwota_operacji):
            print("Niepoprawna kwota operacji w następującej operacji:")
            print(jpk_operacja)
            return 1

        if not is_money_amount_correct(jpk_operacja.saldo_operacji):
            print("Niepoprawna kwota operacji w następującej operacji:")
            print(jpk_operacja)
            return 1

    return 0
```

Funkcja validate_operacje _values() otrzymuje listę obiektów klasy Rachunek, które zawierają operacje na rachunku bankowym i wywołuje funkcje walidujące atrybuty obiektów w liście. Jeżeli jakiś etap walidacji nie zakończy się sukcesem, to funkcja validate_operacje_values() zwróci do pliku głównego wartość 1, co w konsekwencji zakończy program z błędem.

Walidacja kodu urzędu skarbowego

```
def validate_kod_urzedu(kod_urzedu):
    try:
```

```
        kod_urzedu = int(kod_urzedu)
    except ValueError:
        return 1

    if not re.fullmatch(r"^\d{4}$", str(kod_urzedu)):
        return 1

    return 0
```

Funkcja validate_kod_urzedu() sprawdza, czy kod urzędu jest czterocyfrową liczbą całkowitą.

6. Testy poprawności

6.1. Testy jednostkowe funkcji walidujących

Testy jednostkowe funkcji walidujących zawarte są w pliku tests.py. Funkcje walidujące otrzymują różnego rodzaju zmienne i sprawdzane są wyniki, które one zwrócą. Każdy zwrócony wynik musi być zgodny z oczekiwanym, aby testy zakończyły się sukcesem.

Importy

```
import unittest

import numpy as np
import pandas as pd

from src.validator.validator import is_date_correct, check_podmiot_columns,
check_rachunek_columns, check_operacje_columns, \
    is_one_row, is_not_empty_str, is_nip_correct, is_regon_correct,
is_cc_correct, is_wojewodztwo_correct, \
    is_adres_element_correct, is_nrdomu_correct, is_nrlokalu_correct,
is_kod_pocztowy_correct, is_nrrachunku_correct, \
    is_waluta_correct, is_not_empty, is_money_amount_correct,
validate_kod_urzedu
```

Test funkcji is_date_correct()

```
class TestIsDateCorrect(unittest.TestCase):
    #Prawidłowa data w formacie rok-miesiąc-dzień - True
    def test_valid_date(self):
        self.assertTrue(is_date_correct("2024-05-11"))

    #Data w formacie rok-dzień-miesiąc - False
    def test_invalid_date(self):
        self.assertFalse(is_date_correct("2024-30-02"))

    #Dzień, miesiąc i rok nieoddzielone od siebie - False
    def test_custom_format(self):
        self.assertFalse(is_date_correct(11052024))
```

```
    #Data w formacie rok/miesiąc/dzień - False
    def test_invalid_format(self):
        self.assertFalse(is_date_correct("2024/05/11"))
```

Test funkcji check_podmiot_columns()

```
class TestCheckPodmiotColumns(unittest.TestCase):
    #Pasujące kolumny - True
    def test_matching_columns(self):
        my_columns = ['PelnaNazwa', 'NIP', 'REGON', 'KodKraju', 'Wojewodztwo',
'Powiat', 'Gmina', 'Ulica',
                      'NrDomu', 'NrLokalu', 'Miejscowosc',
'KodPocztowy', 'Poczta']
        df = pd.DataFrame(columns=my_columns)
        self.assertTrue(check_podmiot_columns(df))

    #Brak jednej kolumny - False
    def test_non_matching_columns(self):
        my_columns = ['PelnaNazwa', 'NIP', 'REGON', 'KodKraju', 'Wojewodztwo',
'Powiat', 'Gmina', 'Ulica',
                      'NrDomu', 'NrLokalu', 'Miejscowosc',
'KodPocztowy']
        df = pd.DataFrame(columns=my_columns)
        self.assertFalse(check_podmiot_columns(df))

    #Dodatkowa kolumna - False
    def test_extra_columns(self):
        my_columns = ['PelnaNazwa', 'NIP', 'REGON', 'KodKraju', 'Wojewodztwo',
'Powiat', 'Gmina', 'Ulica',
                      'NrDomu', 'NrLokalu', 'Miejscowosc',
'KodPocztowy', 'Poczta', 'DodatkowaKolumna']
        df = pd.DataFrame(columns=my_columns)
        self.assertFalse(check_podmiot_columns(df))
```

Test funkcji check_rachunek_columns()

```
class TestCheckRachunekColumns(unittest.TestCase):
    #Odpowiednie kolumny - True
    def test_matching_columns(self):
        my_columns = ['NumerRachunku', 'KodWaluty']
        df = pd.DataFrame(columns=my_columns)
        self.assertTrue(check_rachunek_columns(df))

    #Brak jednej kolumny - False
    def test_non_matching_columns(self):
        my_columns = ['NumerRachunku']
        df = pd.DataFrame(columns=my_columns)
```

```
        self.assertFalse(check_rachunek_columns(df))

    #Dodatkowa kolumna - False
    def test_extra_columns(self):
        my_columns = ['NumerRachunku', 'KodWaluty', 'DodatkowaKolumna']
        df = pd.DataFrame(columns=my_columns)
        self.assertFalse(check_rachunek_columns(df))
```

Test funkcji is_one_row()

```
class TestIsOneRow(unittest.TestCase):
    #Jeden wiersz - True
    def test_single_row(self):
        df = pd.DataFrame({'A': [1], 'B': [2]})
        self.assertTrue(is_one_row(df))

    #Wiele wierszy - False
    def test_multiple_rows(self):
        df = pd.DataFrame({'A': [1, 2], 'B': [2, 3]})
        self.assertFalse(is_one_row(df))

    #Brak wierszy - False
    def test_empty_dataframe(self):
        df = pd.DataFrame(columns=['A', 'B'])
        self.assertFalse(is_one_row(df))
```

Test funkcji check_operacje_columns()

```
class TestCheckOperacjeColumns(unittest.TestCase):
    #Odpowiednie kolumny - True
    def test_matching_columns(self):
        my_columns = ['DataOperacji', 'NazwaPodmiotu', 'OpisOperacji',
'KwotaOperacji', 'SaldoOperacji']
        df = pd.DataFrame(columns=my_columns)
        self.assertTrue(check_operacje_columns(df))

    #Brak jednej kolumny - False
    def test_non_matching_columns(self):
        my_columns = ['DataOperacji', 'NazwaPodmiotu', 'OpisOperacji',
'KwotaOperacji']
        df = pd.DataFrame(columns=my_columns)
        self.assertFalse(check_operacje_columns(df))

    #Dodatkowa kolumna - False
    def test_extra_columns(self):
        my_columns = ['DataOperacji', 'NazwaPodmiotu', 'OpisOperacji',
'KwotaOperacji', 'SaldoOperacji', 'DodatkowaKolumna']
        df = pd.DataFrame(columns=my_columns)
```

```
        self.assertFalse(check_operacje_columns(df))
```

Test funkcji is_not_empty_str()

```
class TestIsNotEmptyStr(unittest.TestCase):
    #Niepusty string - True
    def test_non_empty_string(self):
        self.assertTrue(is_not_empty_str("Hello"))

    #Pusty string - False
    def test_empty_string(self):
        self.assertFalse(is_not_empty_str(""))

    #Zmienna, która nie jest stringiem - False
    def test_non_string_input(self):
        self.assertFalse(is_not_empty_str(123))
```

Test funkcji is_nip_correct()

```
class TestIsNIPCorrect(unittest.TestCase):
    #Prawidłowy NIP - True
    def test_valid_nip(self):
        self.assertTrue(is_nip_correct("123-456-32-18"))

    #Nieprawidłowy NIP (za dużo cyfr) - False
    def test_invalid_nip(self):
        self.assertFalse(is_nip_correct("123-456-32-189"))

    #Zmienna, która nie jest stringiem - False
    def test_non_string_input(self):
        self.assertFalse(is_nip_correct(123))

    #NIP prawidłowy ale bez myślników - False
    def test_non_dash_format(self):
        self.assertFalse(is_nip_correct("1234563218"))

    #NIP z odpowiednią liczbą cyfr, ale są one źle oddzielone myślnikami -
False
    def test_incorrect_length(self):
        self.assertFalse(is_nip_correct("12-3456-321-8"))
```

Test funkcji is_regon_correct()

```
class TestIsREGONCorrect(unittest.TestCase):
    #Prawidłowy REGON - True
    def test_valid_regon1(self):
        self.assertTrue(is_regon_correct("12 345678 9"))
```

```
    #Prawidłowy REGON - True
    def test_valid_regon2(self):
        self.assertTrue(is_regon_correct("12 345678 9 1234 5"))

    #Prawidłowy REGON ale bez spacji - False
    def test_valid_regon_without_spaces(self):
        self.assertFalse(is_regon_correct("123456789"))

    #Nieprawidłowy REGON (nieodpowiednia liczba cyfr) - False
    def test_invalid_regon(self):
        self.assertFalse(is_regon_correct("1234567890"))

    #Zmienna, która nie jest stringiem - False
    def test_non_string_input(self):
        self.assertFalse(is_regon_correct(123))

    #Brak REGON - True
    def test_empty_string(self):
        self.assertTrue(is_regon_correct(""))
```

Test funkcji is_cc_correct()

```
class TestIsCCCorrect(unittest.TestCase):
    #Prawidłowy kod kraju - True
    def test_valid_cc(self):
        self.assertTrue(is_cc_correct("PL"))

    #Kod kraju inny niż polski - False
    def test_invalid_cc(self):
        self.assertFalse(is_cc_correct("US"))

    #Prawidłowy kod kraju zapisany małymi literami - True
    def test_lower_case_cc(self):
        self.assertTrue(is_cc_correct("pl"))

    #Prawidłowy kod kraju zapisany małymi i wielkimi literami - True
    def test_mixed_case_cc(self):
        self.assertTrue(is_cc_correct("Pl"))
```

Test funkcji is_wojewodztwo_correct()

```
class TestIsWojewodztwoCorrect(unittest.TestCase):
    #Prawidłowe województwo - True
    def test_valid_wojewodztwo(self):
        self.assertTrue(is_wojewodztwo_correct("małopolskie"))

    #Nieprawidłowe województwo - False
    def test_invalid_wojewodztwo(self):
```

```
        self.assertFalse(is_wojewodztwo_correct("lubusk"))

    #Prawidłowe województwo zapisane wielkimi literami - True
    def test_upper_case_wojewodztwo(self):
        self.assertTrue(is_wojewodztwo_correct("MAŁOPOLSKIE"))

    #Prawidłowe województwo zapisane wielkimi i małymi literami - True
    def test_mixed_case_wojewodztwo(self):
        self.assertTrue(is_wojewodztwo_correct("MałoPoLsKiE"))
```

Test funkcji is_adres_element_correct()

```
class TestIsAdresElementCorrect(unittest.TestCase):
    #Poprawny element adresu - True
    def test_valid_element(self):
        self.assertTrue(is_adres_element_correct("Kazimierz Dolny"))

    #Nieprawidłowy element adresu - False
    def test_invalid_element(self):
        self.assertFalse(is_adres_element_correct("Kazimierz123"))

    #Zmienna, która nie jest stringiem - False
    def test_non_string_input(self):
        self.assertFalse(is_adres_element_correct(123))

    #Pusty string - False
    def test_empty_string(self):
        self.assertFalse(is_adres_element_correct(""))

    #Poprawny lement adresu z polskimi literami - True
    def test_special_characters(self):
        self.assertTrue(is_adres_element_correct("Pruszków"))
```

Test funkcji is_nrdomu_correct()

```
class TestIsNrdomuCorrect(unittest.TestCase):
    #Liczba całkowita - True
    def test_valid_nrdomu_int(self):
        self.assertTrue(is_nrdomu_correct(np.int64(10)))

    #Numer domu w postaci xy, gdzie x to liczba, a y to wielka litera - True
    def test_valid_nrdomu_str(self):
        self.assertTrue(is_nrdomu_correct("10A"))

    #Numer domu w postaci xy, gdzie x to liczba, a y to mała litera - True
    def test_valid_nrdomu_str2(self):
        self.assertTrue(is_nrdomu_correct("10a"))
```

```
    #Nieprawidłowy numer domu - False
    def test_invalid_nrdomu(self):
        self.assertFalse(is_nrdomu_correct("10Aa"))

    #Liczba zmiennoprzecinkowa - False
    def test_non_string_or_int_input(self):
        self.assertFalse(is_nrdomu_correct(10.5))

    #Pusty string - False
    def test_empty_string(self):
        self.assertFalse(is_nrdomu_correct(""))
```

Test funkcji is_nrlokalu_correct()

```
class TestIsNrLokaluCorrect(unittest.TestCase):
    #Liczba całkowita - True
    def test_valid_nrlokalu_int(self):
        self.assertTrue(is_nrlokalu_correct(np.int64(10)))

    #Pusty numer lokalu - True
    def test_empty_nrlokalu(self):
        self.assertTrue(is_nrlokalu_correct(""))

    #String - False
    def test_invalid_nrlokalu(self):
        self.assertFalse(is_nrlokalu_correct("A"))

    #Liczba zmiennoprzecinkowa - False
    def test_non_string_or_int_input(self):
        self.assertFalse(is_nrlokalu_correct(10.5))
```

Test funkcji is_kod_pocztowy_correct()

```
class TestIsKodPocztowyCorrect(unittest.TestCase):
    #Prawidłowy kod pocztowy - True
    def test_valid_kod_pocztowy(self):
        self.assertTrue(is_kod_pocztowy_correct("12-345"))

    #Nieprawidłowy kod pocztowy - False
    def test_invalid_kod_pocztowy(self):
        self.assertFalse(is_kod_pocztowy_correct("123-456"))

    #Zmienna, która nie jest stringiem - False
    def test_non_string_input(self):
        self.assertFalse(is_kod_pocztowy_correct(123))

    #Pusty string - False
    def test_empty_string(self):
```

```
        self.assertFalse(is_kod_pocztowy_correct(""))

    #Kod pocztowy bez myślnika - False
    def test_invalid_format(self):
        self.assertFalse(is_kod_pocztowy_correct("12345"))
```

Test funkcji is_nrrachunku_correct()

```
class TestIsNrRachunkuCorrect(unittest.TestCase):
    #Prawidłowy numer rachunku - True
    def test_valid_nr_rachunku(self):
        self.assertTrue(is_nrrachunku_correct("12 3456 7890 1234 5678 9012
3456"))

    #Nieprawidłowy numer - False
    def test_invalid_nr_rachunku(self):
        self.assertFalse(is_nrrachunku_correct("123 456 7890 1234 5678 9012
3456"))

    #Zmienna, która nie jest stringiem - False
    def test_non_string_input(self):
        self.assertFalse(is_nrrachunku_correct(123))

    #Pusty string - False
    def test_empty_string(self):
        self.assertFalse(is_nrrachunku_correct(""))
```

Test funkcji is_waluta_correct()

```
class TestIsWalutaCorrect(unittest.TestCase):
    #Prawidłowa waluta - True
    def test_valid_waluta(self):
        self.assertTrue(is_waluta_correct("PLN"))

    #Nieprawidłowa waluta - False
    def test_invalid_waluta(self):
        self.assertFalse(is_waluta_correct("GBP"))

    #Prawidłowa waluta zapisana małymi literami - True
    def test_lower_case_waluta(self):
        self.assertTrue(is_waluta_correct("eur"))

    #Prawidłowa waluta zapisana małymi i wielkimi literami - True
    def test_mixed_case_waluta(self):
        self.assertTrue(is_waluta_correct("ChF"))
```

Test funkcji is_not_empty()

```
class TestIsNotEmpty(unittest.TestCase):
    #Niepusty string - True
    def test_not_empty_string(self):
        self.assertTrue(is_not_empty("Hello"))

    #Pusty string - False
    def test_empty_string(self):
        self.assertFalse(is_not_empty(""))

    #Liczba całkowita - True
    def test_non_string_input(self):
        self.assertTrue(is_not_empty(np.int64(123)))

    #Tablica - False
    def test_non_empty_list(self):
        self.assertFalse(is_not_empty([1, 2, 3]))

    #Pusta tablica - False
    def test_empty_list(self):
        self.assertFalse(is_not_empty([]))
```

Test funkcji is_money_amount_correct()

```
class TestIsMoneyAmountCorrect(unittest.TestCase):
    #Prawidłowa kwota pieniężna - True
    def test_valid_amount(self):
        self.assertTrue(is_money_amount_correct("10.50"))

    #Nieprawidłowa kwota pieniężna - False
    def test_invalid_amount(self):
        self.assertFalse(is_money_amount_correct(10.555))

    #Nienumeryczny string - False
    def test_non_numeric_input(self):
        self.assertFalse(is_money_amount_correct("abc"))

    #Liczba całkowita - True
    def test_integer_amount(self):
        self.assertTrue(is_money_amount_correct(10))

    #Prawidłowa ujemna kwota - True
    def test_negative_amount(self):
        self.assertTrue(is_money_amount_correct(-10.50))

    #Zero w stringu - True
    def test_zero_amount(self):
        self.assertTrue(is_money_amount_correct("0"))
```

Test funkcji validate_kod_urzedu()

```
class TestValidateKodUrzedu(unittest.TestCase):
    #Prawidłowy kod urzędu - 0
    def test_valid_kod_urzedu(self):
        self.assertEqual(validate_kod_urzedu("1234"), 0)

    #Nieprawidłowy kod urzędu - 1
    def test_invalid_format_kod_urzedu(self):
        self.assertEqual(validate_kod_urzedu(12345), 1)

    #Nienumeryczna wartość - 1
    def test_non_numeric_input(self):
        self.assertEqual(validate_kod_urzedu("abc"), 1)

    #Pusty string - 1
    def test_empty_input(self):
        self.assertEqual(validate_kod_urzedu(""), 1)
```

Wyniki testów jednostkowych

```
Ran 81 tests in 0.078s


OK


Process finished with exit code 0
```

Wszystkie testy jednostkowe zakończyły się sukcesem.

6.2. Poprawność wygenerowanego pliku XML

Sprawdźmy teraz poprawność wygenerowanego przez program pliku XML w formacie JPK_WB. Użyjemy następujących plików CSV:

Zawartość pliku CSV zawierającego dane podmiotu

```
PelnaNazwa,NIP,REGON,KodKraju,Wojewodztwo,Powiat,Gmina,Ulica,NrDomu,NrLokalu,Miejscowosc,KodPocztowy,Poczta

Jan Kowalski,123-456-32-18,12 345678 9,PL,mazowieckie,Pruszkowski,Nadarzyn,Warszawska,1,12,Nadarzyn,11-111,Nadarzyn
```

Zawartość pliku CSV zawierającego dane rachunku bankowego

```
NumerRachunku,KodWaluty

11 1111 1111 1111 1111 1111 1111,PLN
```

Zawartość pliku CSV zawierającego dane operacji na rachunku bankowym

```
DataOperacji,NazwaPodmiotu,OpisOperacji,KwotaOperacji,SaldoOperacji
2024-05-01,Praca,Wynagrodzenie,2500.00,2500.00
2024-05-05,Sklep spożywczy,Zakupy spożywcze,-150.00,2350.00
2024-05-10,Restauracja,Obiad z rodziną,-200.00,2150.00
```

Kodem urzędu skarbowego będzie kod 1234, a nazwą docelowego pliku będzie jpkwb.xml.

Działanie programu:

```
Wczytywanie pliku CSV zawierającego dane podmiotu.
PelnaNazwa,NIP,REGON,KodKraju,Wojewodztwo,Powiat,Gmina,Ulica,NrDomu,NrLokalu,Miejscowosc,KodPocztowy,Poczta
Podaj ścieżkę do pliku CSV zawierającego dane podmiotu: podmiot.csv

Wczytano następujące dane podmiotu:
    PelnaNazwa          NIP       REGON  ... Miejscowosc KodPocztowy    Poczta
0  Jan Kowalski  123-456-32-18  12 345678 9  ...    Nadarzyn     11-111  Nadarzyn

[1 rows x 13 columns]

Wczytywanie pliku CSV zawierającego dane rachunku bankowego.
NumerRachunku,KodWaluty
Podaj ścieżkę do pliku CSV zawierającego dane rachunku bankowego: rachunek.csv

Wczytano następujące dane rachunku bankowego:
                NumerRachunku KodWaluty
0  11 1111 1111 1111 1111 1111 1111      PLN

Wczytywanie pliku CSV zawierającego dane operacji na rachunku bankowym.
DataOperacji,NazwaPodmiotu,OpisOperacji,KwotaOperacji,SaldoOperacji
Podaj ścieżkę do pliku CSV zawierającego dane operacji na rachunku bankowym: operacje.csv

Wczytano następujące dane operacji na rachunku bankowym:
  DataOperacji    NazwaPodmiotu     OpisOperacji  KwotaOperacji  SaldoOperacji
0   2024-05-01          Praca     Wynagrodzenie         2500.0         2500.0
```

```
1   2024-05-05  Sklep spożywczy  Zakupy spożywcze        -150.0        2350.0
2   2024-05-10      Restauracja   Obiad z rodziną         -200.0        2150.0



Podaj kod urzędu skarbowego: 1234


Podaj nazwę pliku XML, do którego zostanie zapisany wyciąg bankowy: jpkwb


Plik JPK został pomyślnie utworzony


Process finished with exit code 0
```

Plik jpkwb.xml

```
<tns:JPK
xmlns:etd="http://crd.gov.pl/xml/schematy/dziedzinowe/mf/2018/08/24/eD/DefinicjeTypy/"
xmlns:tns="http://crd.gov.pl/xml/schematy/dziedzinowe/mf/2018/08/24/eD/DefinicjeTypy/"
xmlns:xsi="http://crd.gov.pl/xml/schematy/dziedzinowe/mf/2018/08/24/eD/DefinicjeTypy/"
xsi:schemaLocation="http://crd.gov.pl/wzor/2021/12/27/11148/schemat.xsd">
    <tns:Naglowek>
        <tns:KodFormularza type="tns:TKodFormularza" kodSystemowy="JPK_WB (1)"
wersjaSchemy="1-0">JPK_WB</tns:KodFormularza>
        <tns:WariantFormularza type="xsd:byte">1</tns:WariantFormularza>
        <tns:CelZlozenia type="tns:TCelZlozenia">1</tns:CelZlozenia>
        <tns:DataWytworzeniaJPK type="etd:TDataCzas">2024-05-11
23:02:17</tns:DataWytworzeniaJPK>
        <tns:DataOd type="etd:TData">2024-05-01</tns:DataOd>
        <tns:DataDo type="etd:TData">2024-05-10</tns:DataDo>
        <tns:DomyslnyKodWaluty
type="kck:currCode_Type">PLN</tns:DomyslnyKodWaluty>
        <tns:KodUrzedu type="etd:TKodUS">1234</tns:KodUrzedu>
    </tns:Naglowek>
    <tns:Podmiot1>
        <tns:IdentyfikatorPodmiotu type="etd:TIdentyfikatorOsobyNiefizycznej">
            <etd:NIP type="etd:TNrNIP">123-456-32-18</etd:NIP>
            <etd:PelnaNazwa type="xsd:token">Jan Kowalski</etd:PelnaNazwa>
            <etd:REGON type="etd:TNrREGON">12 345678 9</etd:REGON>
        </tns:IdentyfikatorPodmiotu>
        <tns:AdresPodmiotu type="etd:TAdresPolski">
            <etd:KodKraju type="etd:TKodKraju">PL</etd:KodKraju>
            <etd:Wojewodztwo
type="etd:TJednAdmin">mazowieckie</etd:Wojewodztwo>
```

```
            <etd:Powiat type="etd:TJednAdmin">Pruszkowski</etd:Powiat>
            <etd:Gmina type="etd:TJednAdmin">Nadarzyn</etd:Gmina>
            <etd:Ulica type="etd:TUlica">Warszawska</etd:Ulica>
            <etd:NrDomu type="etd:TNrBudynku">1</etd:NrDomu>
            <etd:NrLokalu type="etd:TNrLokalu">12</etd:NrLokalu>
            <etd:Miejscowosc
type="etd:TMiejscowosc">Nadarzyn</etd:Miejscowosc>
            <etd:KodPocztowy type="etd:TKodPocztowy">11-111</etd:KodPocztowy>
            <etd:Poczta type="etd:TMiejscowosc">Nadarzyn</etd:Poczta>
        </tns:AdresPodmiotu>
    </tns:Podmiot1>
    <tns:NumerRachunku type="xsd:string">11 1111 1111 1111 1111 1111
1111</tns:NumerRachunku>
    <tns:Salda>
        <tns:SaldoPoczatkowe type="tns:TKwotowy">0.00</tns:SaldoPoczatkowe>
        <tns:SaldoKoncowe type="tns:TKwotowy">2150.00</tns:SaldoKoncowe>
    </tns:Salda>
    <tns:WyciagWiersz>
        <tns:NumerWiersza type="tns:TNaturalnyJPK">1</tns:NumerWiersza>
        <tns:DataOperacji type="etd:TData">2024-05-01</tns:DataOperacji>
        <tns:NazwaPodmiotu type="tns:TZnakowyJPK">Praca</tns:NazwaPodmiotu>
        <tns:OpisOperacji
type="tns:TZnakowyJPK">Wynagrodzenie</tns:OpisOperacji>
        <tns:KwotaOperacji type="tns:TKwotowy">2500.00</tns:KwotaOperacji>
        <tns:SaldoOperacji type="tns:TKwotowy">2500.00</tns:SaldoOperacji>
    </tns:WyciagWiersz>
    <tns:WyciagWiersz>
        <tns:NumerWiersza type="tns:TNaturalnyJPK">2</tns:NumerWiersza>
        <tns:DataOperacji type="etd:TData">2024-05-05</tns:DataOperacji>
        <tns:NazwaPodmiotu type="tns:TZnakowyJPK">Sklep
spożywczy</tns:NazwaPodmiotu>
        <tns:OpisOperacji type="tns:TZnakowyJPK">Zakupy
spożywcze</tns:OpisOperacji>
        <tns:KwotaOperacji type="tns:TKwotowy">-150.00</tns:KwotaOperacji>
        <tns:SaldoOperacji type="tns:TKwotowy">2350.00</tns:SaldoOperacji>
    </tns:WyciagWiersz>
    <tns:WyciagWiersz>
        <tns:NumerWiersza type="tns:TNaturalnyJPK">3</tns:NumerWiersza>
        <tns:DataOperacji type="etd:TData">2024-05-10</tns:DataOperacji>
        <tns:NazwaPodmiotu
type="tns:TZnakowyJPK">Restauracja</tns:NazwaPodmiotu>
        <tns:OpisOperacji type="tns:TZnakowyJPK">Obiad z
rodziną</tns:OpisOperacji>
        <tns:KwotaOperacji type="tns:TKwotowy">-200.00</tns:KwotaOperacji>
        <tns:SaldoOperacji type="tns:TKwotowy">2150.00</tns:SaldoOperacji>
    </tns:WyciagWiersz>
    <tns:WyciagCtrl>
        <tns:LiczbaWierszy type="tns:TNaturalnyJPK">3</tns:LiczbaWierszy>
        <tns:SumaObciazen type="tns:TKwotowy">350.00</tns:SumaObciazen>
```

```
        <tns:SumaUznan type="tns:TKwotowy">2500.00</tns:SumaUznan>
    </tns:WyciagCtrl>
</tns:JPK>
```

Zawartość wygenerowanego pliku XML jest zgodna ze schematem.

7. Wnioski

- Implementacja skutecznej walidacji dużych zbiorów danych wprowadzanych przez użytkownika bywa naprawdę trudna i czasochłonna.
- Program byłby o wiele lepszy, gdyby miał dostęp do bazy danych zawierającej wszystkie powiaty, gminy i miejscowości (i ich kody pocztowe) w Polsce oraz wszystkie adresy w poszczególnych miejscowościach.
- Przydałby się też dostęp do bazy danych zawierającej zarejestrowane numery NIP i REGON.

8. Dalsze możliwości rozwoju

- Generowanie innych plików JPK, np. JPK_VAT albo JPK_FA.
- Aplikacja mobilna