

Uniwersytet Łódzki
Wydział Ekonomiczno-Socjologiczny
Studia Stacjonarne II Stopnia
Kierunek: Ekonometria i analityka danych
Przemysław Szymczak

***Sygnalista VAT* – aplikacja do obsługi i weryfikacji faktur VAT w języku programowania Python.**

Łódź, 2024

Spis treści

Spis treści.....	2
1. Wprowadzenie.....	4
1.1. Struktura pracy	4
1.2. Idea	4
1.3. Podatek VAT	5
1.4. Faktura VAT.....	7
1.5. Obecna oferta i rozwiązania	8
2. Technologie.....	12
2.1. Python.....	12
2.1.1. Język programowania	12
2.1.2. Base – podstawa programu.....	13
2.2. Biblioteki w <i>Pythonie</i>	14
2.2.1. Obsługa danych – <i>pandas, numpy</i>	14
2.2.2. Pobieranie danych – <i>os, requests, json, xmltodict</i>	16
2.2.3. Przetwarzanie tekstu i dat – <i>re, datetime</i>	17
2.2.4. Połączenie z bazą danych – <i>sqlalchemy</i>	17
2.3. <i>SQL</i>	18
2.4. Interfejs graficzny	20
2.4.1. Projektowanie interfejsu – Qt Designer.....	20
2.4.2. Biblioteka <i>PyQt5</i>	21
3. Instrukcja.....	22
3.1. Instalacja oprogramowania	22
3.1.1. <i>Python</i>	22
3.1.2. MS SQL Server.....	26
3.2. Instalacja programu	29
3.2.1. Folder z plikami.....	29
3.2.2. Utworzenie bazy danych	30
3.3. Obsługa programu	32
3.3.1. Uruchamianie.....	32
3.3.2. Aktualizacja bazy danych.....	33
3.3.3. Wybór faktury.....	33
3.3.4. Wyniki weryfikacji	34
4. Implementacja	37
4.1. Bazy danych - banki	37

4.2. Proces weryfikacji	40
4.3. Interfejs graficzny	47
4.4. Rozwój programu	50
Bibliografia	53
Netografia	55
Spis rysunków.....	58
Załączniki	59

1. Wprowadzenie

1.1. Struktura pracy

Treść pracy podzielona została na cztery zasadnicze rozdziały. W pierwszym z nich przedstawione zostały przesłanki stojące za powstaniem aplikacji, geneza jej nazwy, podstawy prawne oraz jej umiejscowienie pośród oprogramowania dostępnego na rynku. W drugim omówione zostały wykorzystane technologie tj. język programowania wraz z jego obiektami, funkcjami i rozszerzeniami, oprogramowanie do zarządzania bazą danych oraz do budowy interfejsu graficznego. Z kolei w trzecim rozdziale zawarta została szczegółowa instrukcja obsługi programu, obejmująca etap instalacji oprogramowania i aplikacji, budowy bazy danych oraz postępowania po uruchomieniu programu. W końcu w ostatnim rozdziale przedstawione zostały wybrane, najistotniejsze fragmenty kodu wraz z opisem ich działania oraz perspektywy na rozbudowę aplikacji w przyszłości.

1.2. Idea

Geneza powstania przedstawionej w tej pracy aplikacji sięga do projektu zaliczeniowego autora z przedmiotu Big Data. Na potrzeby tej pracy program został znacząco rozbudowany, jednak nadal przyświecała mu główna idea – zachowanie prostoty i klarowności przy procesie weryfikacji poprawności faktur, co okazało się być rozwiązaniem niewykorzystywanym powszechnie na rynku (zob. 1.4.). Początkowo program opierał się na danych zapisanych w formacie *csv*, wykorzystywał *web scraping*¹ (proces ściągania danych z interfejsu stron internetowych) oraz podatny był na błędy. Rozwinięcie opierało się przede wszystkim na wykorzystaniu możliwości języka zapytań *SQL* oraz *Microsoft SQL Server* do budowy zaawansowanych baz danych, połączeniu aplikacji poprzez *API*² z innymi programami i usługami (zob. 2.2.2.), rozbudowie procesu weryfikacji elementów faktury oraz budowie interfejsu aplikacji przyjaznego dla użytkownika (zob. 2.4.).

Nazwa programu *Sygnalista VAT* związana jest z jego działaniem. Ma on na celu wskazywanie (sygnalizowanie) błędów/naruszeń związanych z różnymi treściami umieszczonymi na fakturach VAT jak np. stawki podatku, dane sprzedawcy, nabywcy lub

¹ <https://dictionary.cambridge.org/dictionary/english/web-scraping> [Dostęp 12.03.2024]

² <https://dictionary.cambridge.org/dictionary/english/api?q=API> [Dostęp 12.03.2024]

banku. Nawiązuje w ten sposób do roli sygnalisty/informatora (ang. whistleblower) w rozumieniu *Dyrektywy Parlamentu Europejskiego i Rady (UE) 2019/1937 z dnia 23 października 2019 r. w sprawie ochrony osób zgłaszających naruszenia prawa Unii*. Według definicji sygnalista to osoba związana z pewną organizacją, która zgłasza powstawanie i zapobiega wszelkim nieprawidłowościom oraz naruszeniom, które mogłyby łamać prawo Unii Europejskiej i jednocześnie działać na szkodę społeczeństwa. Analogicznie do tego działa aplikacja przedstawiona w tej pracy – weryfikuje poprawność faktur by były one m.in. zgodne z prawem i wypełnione w sposób nie szkodzący podmiotom, których dotyczą.

1.3. Podatek VAT

Poniższe dwa podrozdziały nawiązują w znacznym stopniu do *Ustawy z dnia 11 marca 2004 r. o podatku od towarów i usług (ze zmianami)* zwanej dalej ustawą. Zawiera ona wszystkie najważniejsze informacje o m.in. obowiązku podatkowym, podatku oraz fakturze VAT. Zagadnienia te zostaną po krótkce omówione, aby lepiej można było zrozumieć istotę i podstawę prawną programu *Sygnalista VAT*. W związku z tym, że akty prawne ulegają ciągłym nowelizacjom, należy zaznaczyć, iż to studium opiera się o wersję ustawy obowiązującą od 1 lipca 2023 r., według *Obwieszczenia Marszałka Sejmu RP w sprawie ogłoszenia jednolitego tekstu ustawy o podatku od towarów i usług*. Co więcej, za każdym razem, kiedy w pracy pojawiają się określenia „podatek” lub „faktura” dotyczą one podatku lub faktury VAT.

Na wstępie należy przytoczyć najważniejsze definicje. Według artykułu 15. ust. 1. i 2. ustawy: podatnikami VAT są m.in. osoby prawne oraz fizyczne wykonujące działalność gospodarczą, czyli proceder polegający na ciągłym wykorzystaniu towarów oraz wartości niematerialnych i prawnych w celach zarobkowych. Jednostki te rejestrowane są w urzędzie skarbowym jako podatnicy VAT (Art. 96.) oraz widoczne są w wykazie elektronicznym zwanym białą listą podatników VAT (Art. 96b. ust. 1.).

Podmioty te poprzez obrót przedmiotem opodatkowania (Art. 5.) np. przez sprzedaż towarów (przeniesienie prawa do rozporządzania dobrem – Art. 7. ust. 1.) lub usług (świadczenie ich na rzecz osób prawnych, fizycznych itp., niebędące dostawą towaru – Art. 8. ust. 1.), obarczone zostają obowiązkiem podatkowym (Art. 19a. ust. 1.). Jest to świadczenie pieniężne na rzecz urzędu skarbowego (Art. 103. ust. 1.) związane ze sprzedażą towaru lub usługi. Ustawa obejmuje znacznie więcej rodzajów podatników, przedmiotów opodatkowania oraz sytuacji generujących obowiązek podatkowy, jednak przedstawione zostały najistotniejsze, najogólniejsze i najczytelniejsze z nich, w celu zrozumienia podstawowego działania prawa.

Zgodnie z powyższym VAT (ang. value added tax) to podatek od towarów i usług lub od wartości dodanej, płacony przez sprzedawcę lub usługodawcę w momencie sprzedaży lub wykonywania usługi. Dokładniej mówiąc jest to podatek od wartości towaru lub usługi stanowiącej podstawę opodatkowania tj. zapłaty z tytułu sprzedaży wraz z dopłatami i subwencjami (Art. 29a. ust. 1.). Wartość netto towaru lub usługi wykorzystywana jest do obliczenia kwoty podatku i wartości brutto do czego potrzebne są stawki podatku.

Wyróżnić można cztery podstawowe stawki – 23%, 8%, 5% oraz 0% (Art. 41. ust. 1-4. oraz Art. 146ea.).

- Standardowa stawka 23% (obowiązująca od 1 stycznia 2011 r. w miejscu stawki 22%) dotyczy towarów oraz usług, których nie obejmują inne stawki, wylistowane w załącznikach do ustawy. Są to przykładowo: gaz ziemny, paliwo, energia elektryczna, alkohol, kosmetyki, zabawki czy urządzenia RTV i AGD.
- Obniżona stawka 8% (przed 1 stycznia 2011 r. – 7%) obowiązuje dla towarów i usług zebranych w załączniku nr 3 do ustawy, czyli m.in. kawa i herbata, cukry i wyroby cukiernicze, zwierzęta i rośliny żywe, usługi nieweterynaryjne związane z rolnictwem, chodem i hodowlą zwierząt, wywozem odpadów oraz odprowadzaniem ścieków, transport lądowy, wodny i lotniczy, usługi weterynaryjne oraz fryzjerskie.
- Z kolei stawka 5% związana jest z towarami i usługami wskazanymi w załączniku nr 10 do ustawy, takimi jak: produkty spożywcze (mięsa i ryby, warzywa, owoce, zboża, produkty mleczarskie, tłuszcze, oleje, przetwory, lody, wody), książki, gazety, środki higieny oraz usługa dostarczania drogą elektroniczną gazet, książek itp.
- W końcu stawką 0% (załącznik nr 8 do ustawy) opodatkowane są urządzenia elektroniczne takie jak jednostki centralne komputerów, monitory, drukarki, skanery, urządzenia komputerowe do pism Braille’a i urządzenia transmisji danych cyfrowych.
- Poza towarami i usługami, których dotyczą wymienione stawki wyróżnić można także takie, które nie podlegają opodatkowaniu tj. są z niego zwolnione (Art. 43. ust. 1. oraz załącznik nr 7 do ustawy). Należą do nich m.in. towary i usługi rolników ryczałtowych, transporty medyczne, dostawy złota do Narodowego Banku Polskiego, budowli lub budynków, usługi zarządzania funduszami i portfelami inwestycyjnymi, opieka medyczna, usługi prywatnego nauczania oraz towary wyprodukowane przez Organizację Narodów Zjednoczonych lub jej specjalistyczną agencję takie jak filmy edukacyjne, naukowe, kulturalne oraz środki ich przechowywania, zapisane nośniki rejestracji dźwięku, mapy i makiety.

Należy pamiętać, że stawki mogą się zmieniać w czasie w zależności od woli i potrzeb ustawodawcy, co nie zostało uwzględnione w algorytmie ani bazie danych, na której opiera się *Sygnalista VAT*. Wykorzystane zostały stałe wartości procentowe oraz przypisane do nich towary i usługi, obowiązujące w momencie powstawania aplikacji według wspomnianej nowelizacji ustawy. Dodatkowo program pomija inne stawki związane z szczególnymi przypadkami, gdyż dodanie ich nie jest wymagane dla komfortu pracy użytkownika. Funkcja ta wraz z uwzględnieniem zmienności w czasie może być przedmiotem dalszego rozwoju programu, na potrzeby konkretnych firm, w jakich nietypowe stawki miałyby zastosowanie.

1.4. Faktura VAT

Wszystkie czynności podlegające opodatkowaniu wiążą się z obowiązkiem wystawienia przez podatnika faktury VAT (Art. 106b.). Dokument ten powinien powstać w dwóch egzemplarzach, w formie papierowej lub elektronicznej, celem udokumentowania sprzedaży towaru lub świadczenia usługi między podatnikami lub podatnikiem a konsumentem, by wykazać wartość podatku oraz wskazać jaki podmiot zobowiązany jest wpłacić go do urzędu skarbowego.

Według Art. 106e. faktura powinna zawierać określone elementy, które podzielić można na homogeniczne grupy.

- Elementy identyfikacyjne – pozwalające odróżnić fakturę od innych – data wystawienia, kolejny numer serii, data wykonania/zakończenia dostawy/świadczenia usług lub data otrzymania zapłaty różna od daty wystawienia.
- Dane sprzedawcy i nabywcy – imiona i nazwiska lub nazwy podmiotów z adresami oraz NIP (Numer Identyfikacji Podatkowej) potrzebne do identyfikacji podatnika na potrzeby podatku.
- Towar lub usługa oraz przeliczenie – nazwy/rodzaje towarów i/lub usług oraz ich ilość/liczba, miara i cena jednostkowa netto, do tego wartość sprzedaży netto jako iloczyn ceny i ilości, stawka podatku dotycząca towaru lub usługi, kwota podatku jako iloczyn wartości netto i stawki podatku oraz wartość brutto będąca sumą kwoty netto i podatku.
- Sumy końcowe – sumy wartości netto, kwoty podatku i wartości brutto podzielone na poszczególne stawki, kwota należności ogółem jako całkowita suma wartości brutto wyrażona w formie liczbowej i słownej.
- Pozostałe – kwoty opustów lub obniżek nieuwzględnione w cenie jednostkowej netto, informacje o podstawie prawnej w przypadku towarów zwolnionych od podatku.

- Elementy dodatkowe – specjalne napisy w charakterystycznych przypadkach – „metoda kasowa” (zw. z powstaniem obowiązku podatkowego dopiero w momencie otrzymania całości lub części zapłaty), „samofakturowanie” (zw. z wystawieniem przez nabywcę faktury w imieniu i na rzecz sprzedawcy), „odwrotne obciążenie” (zw. ze sprzedażą towarów lub usług, dla których to nabywca rozlicza podatek), „mechanizm podzielonej płatności” (zw. ze sprzedażą towarów wrażliwych – załącznik nr 15 do ustawy – i jednocześnie, gdy kwota należności ogółem przekracza 15 tys. zł lub równowartość w innej walucie).

Na fakturze dodatkowo znajdować mogą się takie elementy jak: sposób płatności, numer konta bankowego, nazwa oraz oddział banku, jeżeli płatność odbyła się poprzez przelew bankowy, nazwa dokumentu, słowa „Kopia”, „Oryginał”, podpis i/lub pieczęć (Caryk, 2019). Przykładowy wygląd faktury zobaczyć można na załączniku nr 1.

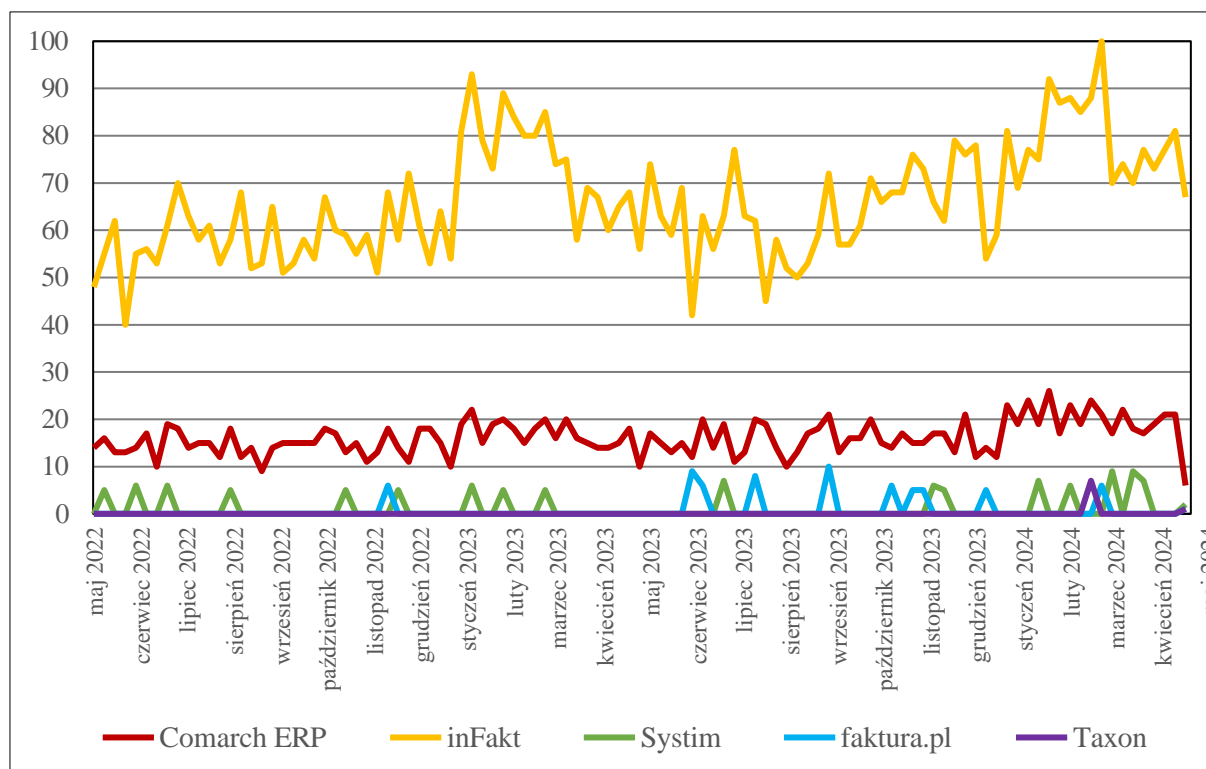
Powyżej przedstawione zostały najważniejsze charakterystyki podatku oraz faktury, na których bazuje program *Sygnalista VAT*. Na potrzeby stworzenia aplikacji wykorzystany został jedynie ułamek opcji i możliwości weryfikacji elementów faktury jakie prezentuje ustawa. Zabieg ten jest celowy by zachować prostotę programu oraz pozostawić miejsce na jego rozbudowę (zob. 4.4. Rozwój programu). Streszczone przepisy prawa podatkowego powinny być podstawą działania każdego oprogramowania do obsługi i wystawiania faktur.

1.5. Obecna oferta i rozwiązania

By zgłębić potrzebę powstania *Sygnalisty VAT* należy zapoznać się dostępnymi już na rynku programami do obsługi faktur VAT. W tej części pracy omówione zostaną wybrane z nich tj. Comarch ERP, inFakt, Systim, faktura.pl oraz Taxon które pojawiają się w m.in. wynikach wyszukiwania w Google.com. Ich popularność można analizować w kontekście częstości wyszukiwania fraz z nimi związanych, które zaprezentowane zostały na poniższym wykresie.

Wartości bliższe 100 oznaczają liczbę zapytań zbliżoną do najwyższej spośród rozpatrywanych, zatem 20 oznaczać będzie 1/5 maksymalnej popularności wskazanych fraz w zadanym okresie. W związku z tym, na pierwszy rzut oka zorientować się można, że w okresie maj 2022 r. – maj 2024 r. względnie najczęściej wyszukiwane były informacje o programach inFakt i Comarch ERP.

Wykres 1 Popularność fraz zw. z programami do obsługi faktury VAT w okresie 05.2022-05.2024.



Źródło: na podstawie danych z *Google Trends*, opracowanie własne.

Omówienie bieżącej oferty rozpocząć należy zatem od prawdopodobnie najważniejszej pozycji tj. oprogramowania funkcjonującej od 1993 r. krakowskiej firmy Comarch SA, która na przestrzeni 30 lat rozwinęła swoją działalność w Polsce, Europie i na całym świecie, w branżach: IT, administracji publicznej, służby zdrowia, ubezpieczeniach, handlu i przede wszystkim, z punktu widzenia tej pracy, w finansach³. Pośród szerokiej gamy produktów oferuje ona systemy ERP (ang. enterprise resource planning – planowanie zasobów przedsiębiorstwa), które obejmują m.in. sprzedaż, logistykę, relacje z kontrahentami oraz księgowość i rachunkowość. Oznacza to, że oferowane oprogramowanie ma bardzo szerokie zastosowanie, nie tylko do prowadzenia księgowości i obsługi faktur.

Jak przeczytać można na oficjalnej stronie firmy, oferuje ona przykładowo program w wersji XT⁴ pozwalający na szybkie wystawianie faktur, bazując na NIP kontrahenta, którego dane pobierane są z bazy GUS; wprowadzonych informacjach o towarach i/lub usługach oraz ich wartości netto, a wystawiona faktura ma gwarantowaną poprawność. Wydanie Optima⁵ pozwala na jeszcze dokładniejszą obsługę faktur: ich ewidencję, rozliczenia, weryfikację

³ <https://www.comarch.pl/o-firmie/> [Dostęp 30.12.2023]

⁴ <https://www.erpzt.pl/funkcje/faktura-online/> [Dostęp 30.12.2023]

⁵ <https://www.comarch.pl/erp/comarch-optima/ksiegowosc/> [Dostęp 30.12.2023]

kontrahentów oraz składanie deklaracji podatkowych. Podobne opcje oferuje również wersja XL⁶. Ceny programów to kolejno od 17 zł do 221 zł miesięcznie, 490 zł za licencję na jeden produkt oraz od 15 zł do ponad 600 tys. zł za wdrożenie systemu w firmie w zależności od jej wielkości.

Wśród oferowanych programów wyróżnić można także produkt inFakt, proponowany przez Spółkę z o.o. o tej samej nazwie. Według opisów zawartych na stronie internetowej firmy⁷ aplikacja asystuje użytkownikowi przy wystawianiu faktur różnego rodzaju, w tym VAT, w wersji elektronicznej z możliwością tworzenia dokumentów ze współpracownikami. Użytkownik podaje NIP na podstawie, którego program zaciąga dane podmiotów z bazy GUS, daty, rodzaje towarów i/lub usług, kwoty i stawki a program generuje fakturę. Program umożliwia wgląd w wystawiane faktury oraz ich status opłacenia. Koszty miesięczne użytkowania takiej aplikacji zależą od liczby wystawianych dokumentów i zaczynają się od 5 zł a kończą na 160+ zł. Dodatkową zaletą tej oferty są usługi księgowych, którzy mogą wspierać klienta w prowadzeniu przedsiębiorstwa.

Kolejnym przykładem dostępnego na rynku oprogramowania do pracy z fakturami jest System oferowany przez firmę Enadis Sp. z o.o. Jak informuje producent⁸ program pozwala na m.in. wystawianie faktur sprzedażowych, w tym VAT. Użytkownik może kontrolować status opłacenia faktur, oznaczyć je odpowiednimi dopiskami, wprowadzić obowiązujące jak i dodatkowe stawki, skorzystać z szablonu, generować faktury, powiązać je z innymi dokumentami oraz płatnościami, a także pobierać dane o kontrahentach z baz GUS i REGON. Co więcej może on generować zestawienia podatkowe, deklaracje i podsumowania roczne rejestrów VAT. Cena programu waha się od 9 zł do 50 zł miesięcznie w zależności od wybranego wariantu.

Poza rozwiązaniami dostępnymi na komputery osobiste znaleźć można również takie, które gwarantują wygodę dzięki funkcjonowaniu na urządzeniach mobilnych. Przykładem takiej aplikacji jest Taxon od Sp. z o.o. o takiej samej nazwie⁹. Producent gwarantuje prostotę, szybkość i bezproblemowość przy korzystaniu z jego oprogramowania. W aplikacji można generować faktury, w kilku wariantach – także na podstawie poprzednich operacji, z automatycznym nadawaniem kolejnego numeru identyfikującego. Aplikacja przesyła

⁶ <https://www.comarch.pl/erp/xl/finanse-i-ksiegowosc/> [Dostęp 30.12.2023]

⁷ <https://www.infakt.pl/program-do-fakturowania/> [Dostęp 30.12.2023]

<https://www.infakt.pl/program-do-faktur/> [Dostęp 30.12.2023]

⁸ <https://www.systim.pl/funkcjonalnosci-systim> [Dostęp 30.12.2023]

⁹ <https://www.taxoninvoice.com/pl/wystawianie-faktur-online/> [Dostęp 14.05.2024]

regularnie informacje o stanie finansowym prowadzonej działalności na podstawie zgromadzonych dokumentów, w tym o koniecznym do zapłacenia podatku VAT. Koszt to 29 zł miesięcznie lub 174 zł rocznie, niezależnie od ilości wystawianych dokumentów.

W końcu wspomnieć należy o Faktura.pl od Spółki z o.o. o takiej samej nazwie¹⁰. Programie funkcjonującym od 1999 r., z którego skorzystało (na moment pisania pracy) ponad 500 tys. użytkowników. Jest to kolejna propozycja opisywana jako wygodna, prosta, bezpieczna oraz pomocna. Jak informuje producent na stronie internetowej dzięki aplikacji faktury wystawiane są zgodnie z przepisami i na czas. Możliwe jest wystawienie kilku wariantów dokumentu, którego numerowanie można wybierać z kilku schematów. Jest to kolejna oferta korzystająca z bazy danych GUS, by na podstawie NIP weryfikować kontrahenta, co więcej daje wybór sposobu płatności wraz z możliwością podania danych banku w tym nr konta. Poza wystawianymi fakturami, użytkownik może gromadzić w aplikacji również otrzymywane dokumenty i przechowywać je w katalogach wraz z własnymi. Koszt użytkowania wynosi od 13,99 zł do 28,99 zł miesięcznie.

Przedstawione programy oferują użytkownikowi kontrolę poprawności faktury VAT przy jej generowaniu, co jest niezwykle istotnym czynnikiem, by wystawiane przez podmiot dokumenty były m.in. zgodne z prawem. W przypadku faktur przyjmowanych przez osobę prawną oczekiwać można, że są one poprawne, gdyż istnieje pewne zaufanie do kontrahentów, jednak czasami mogłaby zaistnieć potrzeba ich weryfikacji od strony nabywcy. Takowa konieczność wiązać może się z faktem zmiennych przepisów, co zostało wcześniej wspomniane, błędami ludzkimi oraz zawodzącą technologią po stronie wystawiającego fakturę. Groźna z perspektywy przedsiębiorstwa byłaby też sytuacja, gdyby fakturę wystawili oszuści, również w takiej sytuacji użyteczne byłoby sprawdzenie faktury przychodzącej. W przypadku firm, przez które przepływa wiele dokumentów manualne sprawdzanie byłoby czasochłonne, dlatego odpowiedzią na tę potrzebę mógłby być program *Sygnalista VAT*.

Proponowany w tej pracy program jest pewnym rozszerzeniem oferowanych na rynku produktów, ponieważ opiera się o bazę danych, w której sprawdza braki i poprawność informacji, wprowadza do procesu weryfikacji etap sprawdzania stawki VAT dla podanego towaru/usługi, łączy się z białą listą podatników VAT i korzysta ze spisu oddziałów banków w Polsce pochodzącego z NBP, na podstawie którego weryfikuje numery kont. Co więcej, dla wygody użytkownika, program jest prosty w obsłudze i wskazuje, jakie elementy faktury są błędne, potencjalnie niebezpieczne i należałoby dalej zweryfikować.

¹⁰ <https://faktura.pl/funkcje/> [Dostęp 14.05.2024]

2. Technologie

2.1. Python

2.1.1. Język programowania

Sygnalista VAT napisany został w *Pythonie*, jednym z najbardziej popularnych obecnie języków programowania, stworzonym w 1991 r. przez Holendra Guido van Rossuma w Centrum Matematycznym w Amsterdamie (niderl. Mathematisch Centrum, obecnie CWI – Narodowy Instytut Badań nad Matematyką i Informatyką) (Rossum i Drake, 2003). Jego autor inspirował się językami niższego poziomu (mniej intuicyjnymi i abstrakcyjnymi dla użytkownika, jeżeli chodzi o np. składnię oraz prostszymi obliczeniowo dla komputera) takimi jak ABC, C czy C++. Był rozwinięciem ABC¹¹, z którego zaczerpnął najlepsze rozwiązania takie jak np. składnia, struktury danych czy możliwość rozbudowy własnymi funkcjami. Wzbogacił go i zoptymalizował, aby nowy język nie wiązał się z np. skomplikowaniem, powolnym działaniem oraz koniecznością znajomości C. Podstawowym celem było po prostu utworzenie czytelnego i prostego w interpretacji języka (Kumar, 2023).

Jak pisze Srinath (2017), popularność *Pythona* objawia się częstotliwością wpisów na forach StackOverflow, liczbą grup na portalu Meetup, ilością bibliotek stworzonych przez użytkowników i udostępnionych na GitHub oraz częstym wymaganiem znajomości języka w ofertach pracy związanych z informatyką, bankowością i finansami, analityką itp. Związane jest to z jego cechami oraz tym co oferuje. Wyróżnia go przede wszystkim ogólne przeznaczenie, dynamika i elastyczność, zorientowanie na obiektach (co pozwala rozdzielić kod na fragmenty, by łatwiej rozwiązywać problemy) jak i możliwość programowania strukturalnego, a także prostota dla użytkownika (również początkującego). Nie wymagana jest zaawansowana znajomość działania kodu na poziomie procesora, użytkownik może korzystać ze składni, którą jest w stanie zinterpretować, a społeczność sympatyków rozwija się dzięki możliwości pisania rozszerzeń w postaci modułów z własnymi funkcjami.

Język programowania, w którym powstał *Sygnalista VAT*, wykorzystywany jest w wielu dziedzinach informatyki; do budowy systemów oprogramowania, interfejsów graficznych, aplikacji sieciowych, obsługi baz danych, gier komputerowych, a także do przeprowadzania analiz ekonometrycznych oraz w uczeniu maszynowym. Z języka korzystają chociażby Google, YouTube, NASA, iRobot, Intel, IBM czy Pixar.

Python jest oprogramowaniem typu „open source”, czyli darmowym, co więcej można go zainstalować na różnych systemach operacyjnych – Windows, Linux, Mac OS. *Sygnalista*

¹¹ <https://www.geeksforgeeks.org/history-of-python/> [Dostęp 17.02.2024]

VAT zbudowany został na wersji 3.11., dlatego zalecane jest zainstalowanie języka w wersji tej lub nowszej do obsługi programu. Dokładna instrukcja instalacji opisana została w rozdziale 4.1.

2.1.2. Base – podstawa programu

Poniżej opisane zostały obiekty, funkcje, biblioteki, dodatkowe oprogramowanie i procesy stojące za programem *Sygnalista VAT*. Prezentacja tych elementów bazuje na dokumentacji¹² jak i na literaturze wprowadzającej do programowania w *Pythonie* od Baranowskiego i Doryń (2020) oraz Severance’a (2016), w nawiasach podane zostały fragmenty tych pozycji według wzoru: „[Nr rozdziału i jego nazwa w dokumentacji, I: strony w tekście Baranowskiego i Doryń, II: strony w tekście Severance’a]”. Jako pierwsze należy przytoczyć funkcjonalności jakie oferuje podstawa programu, czyli tzw. *base*.

- Zmienne – podstawowy typ obiektów wykorzystywany w programie, przyjmujące postać danych tekstowych lub numerycznych, do zapisywania konkretnej informacji jak np. data, dane o firmie lub kwota do zapłaty, używane następnie w dalszej pracy programu np. weryfikacja poprawności daty (maks. 31 dni w miesiącu) lub cyfry kontrolnej w numerze NIP/konta bankowego, porównanie informacji na fakturze z informacją pobraną z białej listy podatników VAT oraz wpisanie kwoty do zapłaty na interfejsie dla użytkownika do odczytu. [3.1.1. Numbers, 3.1.2. Text, I: s.9, 27-34, II: s.16-17]
- Listy – obiekty pozwalające na zgrupowanie i przechowywanie większej ilości informacji np. zestawienia towarów i usług, które związane są z określoną stawką podatków, towary wrażliwe, zbiór informacji o elementach faktury, które są poprawne lub błędne. [3.1.3. Lists, 5.1. More on Lists, I: s.35-42, II: s.75-78]
- Słowniki – obiekty pozwalające na zgromadzenie większych ilości danych oraz odwoływanie się do nich przy użyciu słów kluczowych np. słownik list z towarami przypisanymi do określonych stawek podatku, z którego po podaniu wybranej stawki zwracany jest zbiór towarów i usług z nią związany, czy też słownik z formułą zmiany cyfr na tekst (tj. 1: jeden, 2: dwa, ... 11: jedenaście itd.) służącą do zapisu kwoty do zapłaty słownie. [5.5. Dictionaries, I: s.43-47, II: s.89-90]
- Instrukcje warunkowe – typ funkcji do zmiany działania programu (wykonania określonego fragmentu kodu) w zależności od spełnienia kryteriów, zbudowany z jednego lub kilku/kilkunastu warunków na pojedynczym lub zagnieżdżonym poziomie np. podjęcie próby weryfikacji danych o banku, jeżeli zapłata dokonana została przelewem lub jej brak,

¹² <https://docs.python.org/3/tutorial/index.html> [Dostęp 09.03.2024]

gdy użyto gotówki albo weryfikacja czy faktura powinna mieć mechanizm podzielonej płatności – w oparciu o listę towarów wrażliwych oraz kwotę na fakturze. [4.1. *if* Statements, 5.7. More on Conditions, I: s.16-19, II: s.27-31]

- Funkcje błędów – polecenia do obsługi błędów zabezpieczające program przed wystąpieniem problemu, który mógłby zakłócić jego dalsze działanie, wykorzystane np. do pominięcia etapu sprawdzania poprawności danych przedsiębiorstwa w sytuacji, kiedy nie można się połączyć z białą listą podatników VAT. [8.3. Handling Exceptions, II: 31-32]
- Pętle – polecenia pozwalające na powtarzanie określonego fragmentu kodu wraz ze zmianą wartości wybranej zmiennej lub przejściem do kolejnego elementu w obiekcie np. liście w kolejnej iteracji. W ten sposób wykonane zostało sprawdzenie każdego towaru i usługi na fakturze pod względem przypisania stawki podatku – iterowanie po nr. pozycji lub liście towarów. [4.2. *for* Statements, 5.6. Looping Techniques, I: s.19-20, II: s.47-50]
- Własne funkcje – definiowane, własne algorytmy w celu wydzielenia wybranych procesów np. do sprawdzania poszczególnych elementów faktury – braków i formatów danych, cyfr kontrolnych, informacji o firmach sprzedawcy i nabywcy, banku, poprawności stawek podatku i ich przypisania itp., zwracających określoną wartość jako wynik działania funkcji. [4.7. Defining Functions, II: s.39-42]
- Surowy ciąg znaków (ang. raw string) oraz f-string – specjalne warianty zmiennych tekstowych, w których kolejno:
 - znaki speciale np. „/” w ścieżkach plików traktowane są jako zwykłe, dzięki czemu można zmienić folder, z którego zaczytywane są dane lub grafiki;
 - odwoływać się można do obiektów np. zmiennych i stosować w poleceniach, a następnie zawrzeć je w jednym tekście wynikowym np. w komunikacie informującym o błędzie wraz z wyświetlaniem danych będących źródłem problemu np. zły format nr NIP. [2.4.1. String and Bytes literals, 2.4.3. f-strings]

2.2. Biblioteki w Pythonie

Dzięki temu, że *Python* oferuje możliwość rozbudowy o własne funkcje i biblioteki, w *Sygnaliście VAT* zastosować można było dodatkowe funkcjonalności dostępne w modułach. Ograniczyło to ilość pracy i zagwarantowało wachlarz narzędzi. Aby tego dokonać wprowadzono polecenia importujące konieczne biblioteki, które zostały opisane poniżej.

2.2.1. Obsługa danych – *pandas*, *numpy*

Treści na fakturach zapisane zostały w postaci rozbudowanej, tabelarycznej bazy danych. Dlatego, aby zachować tę strukturę, wprowadzona jest ona w *Pythonie* do obiektu najbardziej zbliżonego formatem, czyli dwuwymiarowych ramek danych dostępnych w module *pandas*¹³. To na ich podstawie program *Sygnalista VAT* buduje mniejsze tabele dotyczące firmy i faktury wybranych przez użytkownika – filtrowana jest najpierw kolumna z nazwą firmy a następnie z numerem faktury. Taki zabieg pozwala działać na mniejszym, czytelniejszym obiekcie, w którym wystarczy sprawdzać jeden wiersz w przypadku większości elementów faktury, gdyż kolejne są jego replikacją, nie licząc danych o towarach i usługach (zob. 2.2.4.).

W związku z tym przy weryfikacji kolejnych elementów faktury następuje zasięganie do kolejnej wartości w wierszu np. data wystawienia, NIP, nazwa firmy itd., wykorzystując funkcję *pd.at*, która pozwala odwołać się do wartości w wybranym wierszu i kolumnie lub do kolejnych rubryk np. ceny, ilości, stawki i nazwy towarów lub usług poprzez wskazywanie nazwy kolumny, w której dane należy sprawdzić. W podobny sposób wykorzystywana jest funkcja *pd.isna* do ewaluacji, czy podany fragment ramki zawiera braki danych (wartości *na*) np. o fakturze, sprzedawcy, banku, o towarach lub usługach itp.

W końcu, na ramkach danych prowadzone są obliczenia. Obiekty te pozwalają na działania na kolumnach i dzięki temu można wyznaczyć wartość netto jako iloczyn ilości i ceny, kwotę podatku jako iloczyn wartości netto i stawki podatku oraz wartość brutto jako sumę wartości netto i kwoty podatku. Obliczone tak wartości dla poszczególnych pozycji na fakturze można następnie zagregować dla poszczególnych stawek podatku oraz zsumować wartości brutto do kwoty do zapłaty. Taki zabieg przygotowuje tabele i wartości jakie należy przedstawić na fakturze (zob. załącznik nr 1.), przy czym wszystkie liczby zaokrąglane są do dwóch miejsc po przecinku funkcją *round*. [10 minutes to pandas: od „Basic data structures in pandas” do „Missing data”, I: s.58-61]

Do wspierania wyżej opisanej biblioteki wykorzystany został jej siostrzany moduł *numpy*¹⁴, na którym bazuje. Wykorzystana została funkcja *np.nan* – wartość braku danych wstawiana w miejsce pustych wartości *None*, które nie były traktowane jako brak przez inne funkcje. Przydatna była również alternatywa funkcji przeszukiwania obiektów po indeksach z modułu *pandas* tj. *np.where* do przeszukiwania kolumn w celu znalezienia indeksu (numeru wiersza). Pozwoliło to wyszukać rekordy, dla których baza danych o bankach zawierała braki i należało je usunąć oraz do odnalezienia oddziału banku po jego numerze.

¹³ https://pandas.pydata.org/docs/user_guide/index.html#user-guide [Dostęp 10.03.2024]

¹⁴ <https://numpy.org/doc/1.26/user/index.html#user> [Dostęp 10.03.2024]

2.2.2. Pobieranie danych – *os*, *requests*, *json*, *xmldict*

Przedstawiony poniżej zestaw zaimportowanych w programie *Sygnalista VAT* modułów związany jest z zacytywaniem danych z różnych źródeł i formatów zapisu.

- *Os*¹⁵ – moduł pozwalający obsługiwać ścieżki dostępu do plików, z którego wykorzystane zostały chociażby funkcje *getcwd* do wczytania lokalizacji programu w pamięci oraz *chdir* do zmiany ścieżki np. przy przejściu do folderu z zapisanymi grafikami wyświetlanymi w interfejsie. Dzięki modyfikacji ścieżki możliwe jest również wczytywanie plików *csv* (poprzez funkcję *open*) znajdujących się w osobnym folderze, który jest łatwy do odnalezienia w momencie potrzeby aktualizacji list towarów i usług przypisanych do określonych stawek podatku (każda w osobnym pliku *csv*).
- *Requests*¹⁶ – moduł do wczytywania internetowych źródeł danych wykorzystywanych w programie tj. informacji o przedsiębiorstwach i bankach. Dane dotyczące firm zaciągane są z usługi API (ang. application programming interface) czyli interfejsu programistycznego aplikacji – *Sygnalista VAT* może połączyć się z białą listą podatników VAT opracowaną przez Ministerstwo Finansów¹⁷. W tym celu używana jest funkcja *requests.get*, której argumentem jest adres strony internetowej z podanym NIP do wyszukania. Następnie pobrane informacje podlegają dalszemu przetwarzaniu (zob. poniżej *json*). W taki sam sposób pobierana jest baza danych banków ze strony internetowej Narodowego Banku Polskiego¹⁸, jednak w jej wypadku w adresie podaje się nazwę pliku w formacie *xml*, w którym zapisane są dane. Program zabezpieczony jest na ewentualne problemy z połączeniem ze źródłami internetowymi, w pierwszym przypadku reaguje na *status_code*, czyli kod odpowiedzi – czy utworzone zostało połączenie z aplikacją MF lub co spowodowało jego brak, natomiast przy drugim źródle podejmowane są trzy próby pobrania danych i zapisania ich w obiekcie.
- *Json*¹⁹ – biblioteka do obsługi plików typu *json* np. dane w tym formacie wczytywane są z białej listy podatników VAT, a następnie przy użyciu funkcji *json.loads* zapisywane są w formie słownika do dalszego wykorzystania przez program np. wyszukiwanie po kluczach NIP, nazwy i adresu firmy.
- *Xmldict*²⁰ – moduł do przekształcania plików w formacie *xml* do postaci słownika np. baza banków oferowana przez NBP modyfikowana jest funkcją *xmldict.parse*. Dane z tego

¹⁵ <https://docs.python.org/3/library/os.html> [Dostęp 11.03.2024]

¹⁶ <https://requests.readthedocs.io/en/latest/> [Dostęp 11.03.2024]

¹⁷ <https://wl-api.mf.gov.pl/#nip?date> [Dostęp 11.03.2024]

¹⁸ <https://ewib.nbp.pl/faces/pages/daneDoPobrania.xhtml> [Dostęp 11.03.2024]

¹⁹ <https://docs.python.org/3/library/json.html> [Dostęp 11.03.2024]

²⁰ <https://pypi.org/project/xmldict/> [Dostęp 11.03.2024]

źródła zapisane są w zagnieżdżonych listach i słownikach w zależności od liczby oddziałów banków oraz ich numeru/numerów, dlatego poprzez pętle spisywany jest każdy możliwy numer identyfikacyjny wraz z nazwą oddziału i banku w kolejnych wierszach ramki danych.

- *Sqlalchemy* – rozszerzenie do wczytywania danych (o fakturach) zapisanych w bazach *SQL* – tworzenia zapytań do tych baz. (zob. 2.2.4.).

2.2.3. Przetwarzanie tekstu i dat – *re*, *datetime*

Jednym z etapów weryfikacji jest proces analizowania zapisu określonych ciągów znaków np. kodu pocztowego, NIP, numeru faktury w celu sprawdzenia czy zapisane zostały w poprawnym formacie. Istotne również jest wyszukiwanie konkretnych elementów adresu firm w danych pobieranych z białej listy podatników VAT – są one zapisane w jednym obiekcie tekstowym i konieczne jest rozdzielenie od siebie miasta, kodu pocztowego oraz ulicy. Wykorzystane zostały w tym celu funkcje z modułu *re*²¹, związanego z wyrażeniami regularnymi (ang. regular expressions – regex), czyli ciągami charakterystycznych symboli w tekście. Polecenia *re.fullmatch* i *re.findall* pozwoliły na wyszukanie takich łańcuchów znaków w danych widocznych na fakturze np. czy kod pocztowy zapisany został w formacie „dd-ddd”, gdzie „d” to cyfra. [I: s.89-106, II: s.106-114]

W podobny sposób wykorzystane zostały możliwości modułu *datetime*²², który skupia się na operowaniu datami. Funkcja *datetime.datetime* pozwala sprawdzić, czy dane na fakturze tj. dzień, miesiąc i rok, są w stanie utworzyć prawdziwą datę (np. liczba dni w miesiącu nie przekraczająca poza 31, a miesiące w roku poza 12).

2.2.4. Połączenie z bazą danych – *sqlalchemy*

Główna baza danych z jakiej korzysta *Sygnalista VAT* osadzona została w *Microsoft SQL Server* (zob. 2.3.), z związku z tym wymagane jest powiązanie jej z aplikacją, co dokonane zostało przy pomocy modułu *sqlalchemy*²³. Na wstępie poprzez funkcję *create_engine* zbudowany zostaje silnik, opierający się o serwer, na którym przechowywana jest baza, jej nazwę oraz sterownik. Następuje połączenie *connect*, za pomocą którego przesyłane jest zapytanie. Wynikiem jest zaciągnięcie pożądanej bazy do ramki danych. Tak wprowadzone dane są następnie korygowane – zapisane są one w formacie tekstowym, dlatego za pomocą funkcji *pd.to_numeric* kolumny z ilością i ceną towaru lub usługi zamieniane są na wartości liczbowe, które dalej wykorzystane mogą być do obliczeń. Co więcej puste wartości zamieniane są funkcją *replace* na brak danych *np.nan*.

²¹ <https://docs.python.org/3/library/re.html> [Dostęp 11.03.2024]

²² <https://docs.python.org/3/library/datetime.html> [Dostęp 11.03.2024]

²³ <https://docs.sqlalchemy.org/en/20/#> [Dostęp 28.03.2024]

Połączenie z serwerem *SQL* następuje również w kontekście bazy danych banków. Kiedy użytkownik decyduje się na korzystanie z udostępnionej (w załączonym folderze) lub dotychczasowej listy banków następuje koneksja jak powyżej. Natomiast jeżeli woła użytkownika jest aktualizacja tej bazy do najnowszej wersji, to po pobraniu i przetworzeniu danych od NBP zbierane są one w jeden obiekt, a następnie poprzez połączenie z serwerem dochodzi do nadpisania bazy.

2.3. *SQL*

Zbiór informacji o fakturach wykorzystywany przez program *Sygnalista VAT* przechowywany jest formie bazy danych w systemie zarządzania bazami *SQL Server* od Microsoftu²⁴, wprowadzonego po raz pierwszy w 1989 r. Program ten bazuje na strukturalnym języku zapytań – *SQL* (ang. Structured Query Language). Jak pisze Beaulieu (2009) jego historia sięga 1970 r. i badania dra E. F. Codda z IBM, w sprawie przechowywania danych w formie zbioru powiązanych ze sobą tabel relacyjnych. W wyniku prac tego naukowca powstał język *DSL/Alpha*, który po kolejnych modyfikacjach i ulepszeniach przekształcony został w *SQUARE*, a w końcu w *SEQUEL*, którego nazwa skrócona została do *SQL*. Na przestrzeni lat wydane zostało wiele wersji oprogramowania, jednak na potrzeby obsługi *Sygnalisty VAT* potrzebna jest co najmniej bezpłatna wersja *Microsoft SQL Server 2019 Developer*.

SQL jest językiem zapytań, co oznacza, że poprzez kod użytkownik odwołuje się do bazy danych, wskazując jakie informacje chce pozyskać, jak mają one zostać przetworzone (np. agregacja) i otrzymuje wynik zapytania, będący tabelą z danymi lub modyfikacją bazy. Relacyjność oznacza, że użytkownik ma do czynienia z szeregiem dwuwymiarowych zbiorów danych, w których kolumny to atrybuty a wiersze to rekordy. Poprzez relacje z kilku tabel wejściowych można utworzyć jedną wyjściową bazując na wspólnym atrybucie będącym kluczem do połączenia.

Zgodnie z pojęciem relacyjnych baz, dane wykorzystywane przez *Sygnalistę VAT* przechowywane są w mniejszych tabelach, w których znajdują się grupy atrybutów:

- Faktury,
- Towary i Usługi,
- Kontrahenci,
- Konta Bankowe.

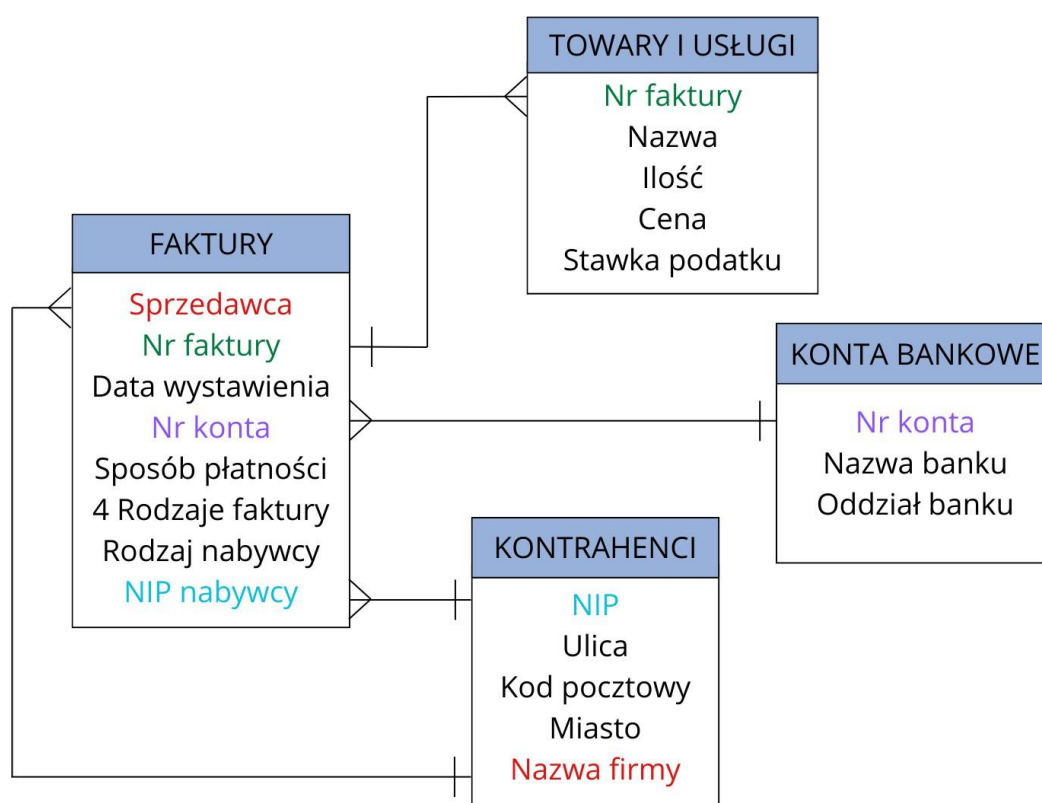
²⁴ <https://learn.microsoft.com/pl-pl/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16>
[Dostęp 28.03.2024]

Istnieje wiele sposobów budowania powiązań między ramkami danych, jednak w tym wypadku wykorzystane zostały połączenia typu:

- Left join, które dodaje do pierwszego obiektu wejściowego atrybuty drugiego, dla rekordów zawierających odpowiednią wartość atrybutu klucza łączącego tabele. Przykładowo, jeżeli w tabeli Faktury w pewnym rekordzie obecna będzie błędna lub nie pojawi się wcale wartość dla klucza NIP nabywcy, to do tego rekordu nie dodane zostaną dane z tablicy Kontrahenci.
- Inner join, które zwraca w wyjściowej tabeli wyłącznie rekordy zawierające wartości atrybutu klucza, które występują w obu łączonych tabelach (np. nie wyświetlają się faktury bez towarów i/lub usług na nich lub towary/usługi przypisane do faktury, która nie jest widoczna w tabeli Faktury).

Na poniższym schemacie zauważyć można, które tabele powiązane zostały ze sobą określonymi kluczami.

Rysunek 1. Relacje tworzące bazę danych wykorzystywaną przez Sygnalistę VAT



Źródło: opracowanie własne w Canvie

- Faktury – podstawowa tablica zawierająca główne informacje o fakturze – jakie przedsiębiorstwo ją wystawia, jaki jest jej numer i data wystawienia, sposób płatności, czy

występują specjalne napisy, informacja o nabywcy tj. osoba fizyczna lub prawna. Do niej przyłączane są pozostałe tabele.

- Towary i Usługi – z tej tabeli do każdego numeru faktury dodawane są nazwy znajdujących się na niej towarów i/lub usług, ich ilość, cena oraz przypisana im stawka podatku (o ile występują). Pozycje z przypisanym numerem dokumentu, który jest nieobecny w tabeli Faktury są pomijane. Połączenie wiele do jednego – na danej fakturze znajdować się może więcej niż jedna pozycja.
- Kontrahenci – tablica zawierająca dane o firmach, z niej dodawane są szczegółowe informacje o sprzedającym na podstawie jego nazwy tj. NIP i adres, o ile są one dostępne. Firmy nieobecne w tabeli Faktury są pomijane. Połączenie jeden do wielu – w bazie może być więcej niż jedna faktura wystawiona przez dane przedsiębiorstwo. Jeżeli nabywcą jest osoba prawna to poprzez NIP następuje dodatkowe połączenie by dołączyć informacje takie jak nazwa i adres odbiorcy. Połączenie jeden do wielu – dany podmiot może być nabywcą kilku faktur.
- Konta Bankowe – tabela, z którą następuje powiązanie w przypadku, gdy płatność za fakturę odbywa się przy pomocy przelewu. W ramce Faktury podawany jest numer konta, który jest kluczem do dołączenia informacji o banku i jego oddziale. Połączenie jeden do wielu – dany nr konta może być podany na wielu fakturach lub wcale nie musi zostać podany, jeżeli płatność dokonana została gotówką – w tej sytuacji nie dołączone zostają w ogóle rekordy z tablicy Konta Bankowe.

2.4. Interfejs graficzny

2.4.1. Projektowanie interfejsu – Qt Designer

Nieodłączną częścią *Sygnalisty VAT*, pozwalającą na swobodne użytkowanie, jest jego interfejs graficzny. Zaprojektowany został on w aplikacji *Qt Designer*²⁵, która daje możliwość swobodnej budowy wizualnej części programu. Ma prostą obsługę, gdyż cały proces polega na dodawaniu na oknach programu elementów interfejsu zwanych widżetami. Utworzone zostało w ten sposób główne okno, które posiada kilka wersji w zależności od etapu użytkowania programu. Kolejne widoki zawierają różne elementy, takie jak pola tekstowe, listy, przyciski czy tabele. Cała szata graficzna sformatowana została, aby zachować schludny wygląd programu, w tym: kolory, czcionki, wielkość widżetów, grafiki takie jak logo czy strzałka powrotu. Tak przygotowana wizualizacja zamieniona została na kod i zapisana jako osobny plik

²⁵ <https://doc.qt.io/qt-6/qt designer-manual.html> [Dostęp 28.03.2024]

w języku Python, który importuje i wykorzystuje *Sygnalista VAT*. Jednak by interfejs zawierał wypełnione widżety konieczne było połączenie ich z obiektami w kodzie programu.

2.4.2. Biblioteka *PyQt5*

W tym celu wykorzystywana jest biblioteka *PyQt5*²⁶, która służy do uruchomienia interfejsu oraz jego obsługi. Na początku podane zostaje jaka wizualizacja ma zostać wyświetlona na wstępie – ekran powitalny dla użytkownika zawierający logo i tytuł. Na każdym kolejnym etapie dostępne są przyciski, które pozwalają użytkownikowi przenieść się na następną wizualizację po kliknięciu, dzięki funkcjom przypisanym do tych akcji.

W wielu wypadkach, poza zwyczajnym przejściem do kolejnego ekranu, w tle zachodzą procesy pozwalające na żywo modyfikować treści związane z programem np. jeżeli użytkownik zdecyduje się zaktualizować bazę banków, działanie to jest wykonywane wraz z przełączeniem na kolejne okno. Są to również operacje związane z widżetami, które dają użytkownikowi opcję wyboru np. lista dostępnych przedsiębiorstw i faktur aktualizowana jest przy przechodzeniu między oknami, dzięki czemu możliwe jest filtrowanie na bieżąco wykorzystywanej bazy danych. Uzupełnianie widżetów widoczne jest także po wybraniu faktury do sprawdzenia, ponieważ wynikiem jej weryfikacji są listy elementów poprawnych i błędnych. Skutkuje to także wygenerowaniem wydruku wybranej faktury zawierającym pola tekstowe wypełnione danymi np. identyfikującymi fakturę jak nr, data, rodzaj czy informacjami o przedsiębiorstwach lub banku oraz tabele z towarami i/lub usługami a także wyliczonymi dla nich kwotami.

Poza podstawowym oknem, użytkownik może zobaczyć również okno błędu, na którym wyświetlany jest komunikat nakazujący użytkownikowi zaznaczyć/wybrać fakturę do sprawdzenia, jeżeli tego nie zrobił. Przy uruchamianiu interfejsu wykorzystywane są również funkcje z modułu *sys* umożliwiające start oraz zamknięcie programu.

²⁶ <https://www.riverbankcomputing.com/static/Docs/PyQt5/> [Dostęp 28.03.2024]

3. Instrukcja

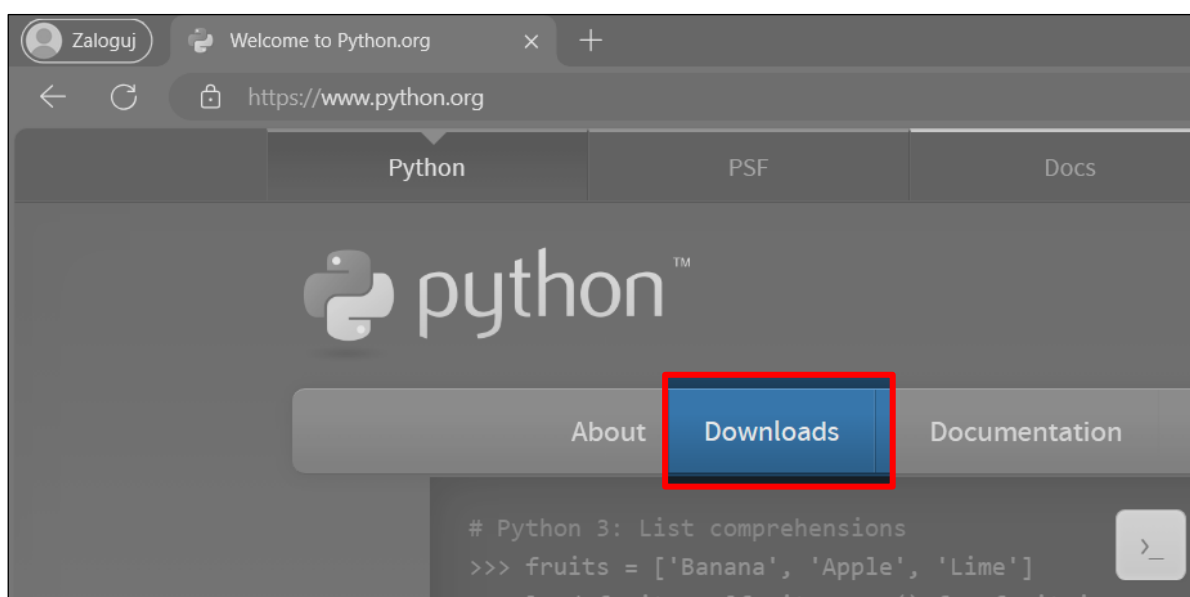
Poniższy rozdział zawiera szczegółową instrukcję, która krok po kroku wyjaśnia, jak pobrać wymagane oprogramowanie, zbudować bazę danych na serwerze SQL oraz jak posługiwać się aplikacją *Sygnalista VAT*. Po zapoznaniu się z nią użytkownik będzie w stanie samodzielnie weryfikować faktury dostępne w udostępnionej bazie lub nowe, które do niej doda.

3.1. Instalacja oprogramowania

3.1.1. *Python*

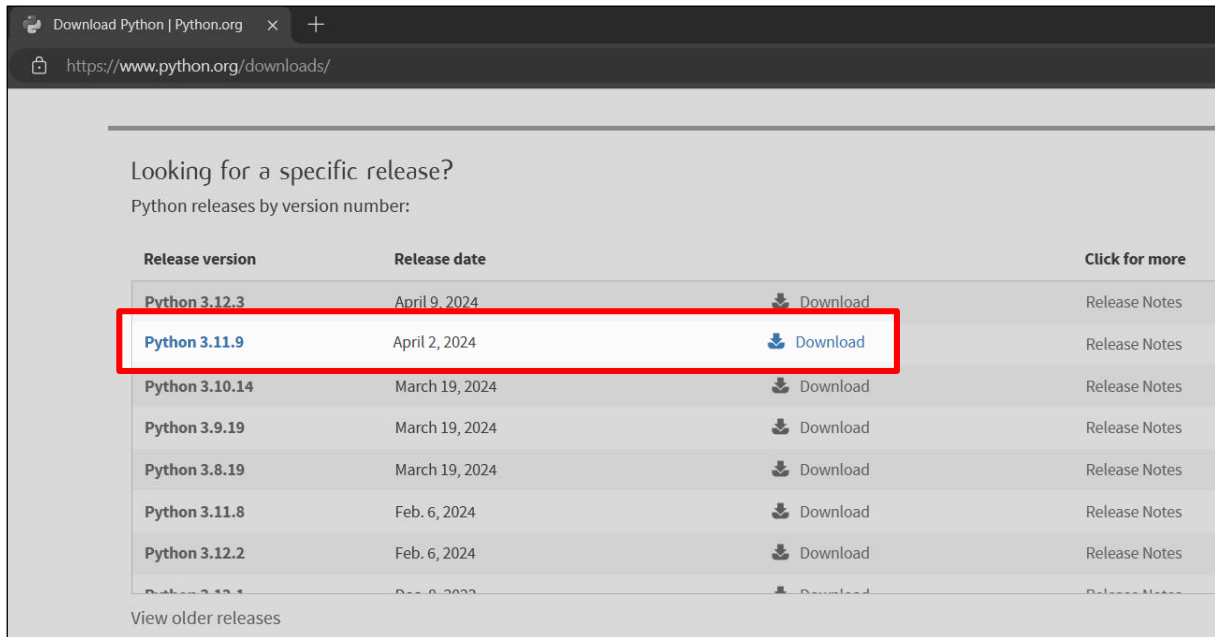
Przed rozpoczęciem pracy z *Sygnalistą VAT* konieczne jest pobranie i instalacja kluczowego oprogramowania. Po pierwsze, należy na urządzeniu zainstalować język programowania, w którym zbudowana została aplikacja. W tym celu należy na oficjalnej stronie internetowej *Pythona*²⁷ wejść w zakładkę *Downloads* (Rys. 2.) i odszukać wydanie 3.11.x (gdzie x to najnowsze, dostępne pliki danego wydania) lub wyższe (Rys. 3.), a następnie kliknąć *Download*.

Rysunek 2. Oficjalna strona internetowa języka Python



²⁷ <https://www.python.org/> [Dostęp 20.04.2024]

Rysunek 3. Lista wydań Pythona



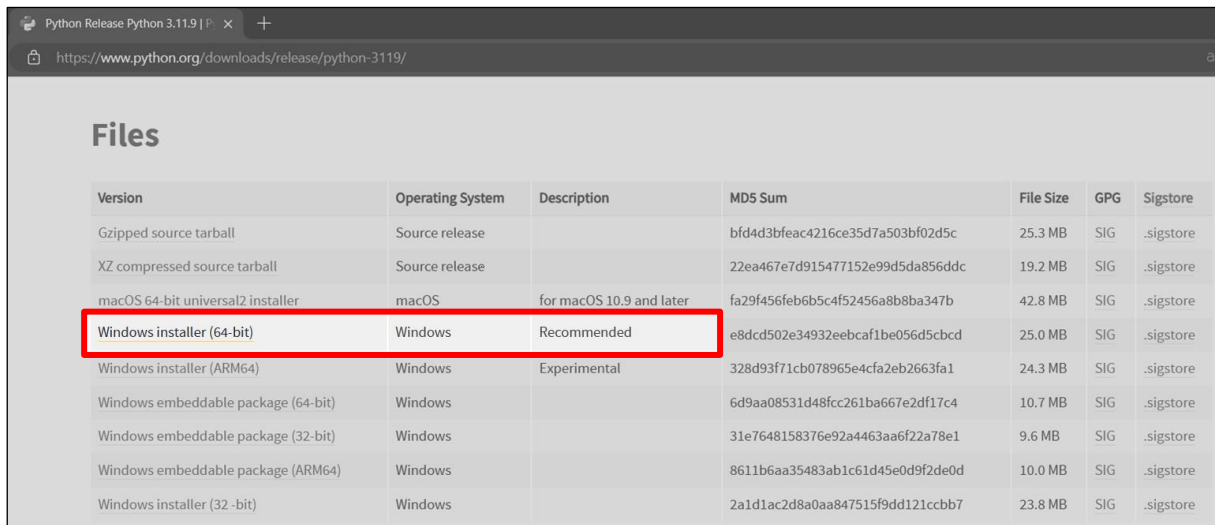
Looking for a specific release?
Python releases by version number:

Release version	Release date		Click for more
Python 3.12.3	April 9, 2024	Download	Release Notes
Python 3.11.9	April 2, 2024	Download	Release Notes
Python 3.10.14	March 19, 2024	Download	Release Notes
Python 3.9.19	March 19, 2024	Download	Release Notes
Python 3.8.19	March 19, 2024	Download	Release Notes
Python 3.11.8	Feb. 6, 2024	Download	Release Notes
Python 3.12.2	Feb. 6, 2024	Download	Release Notes

[View older releases](#)

Po przekierowaniu na kolejną stronę, należy odszukać odpowiednią wersję, np. dla systemu operacyjnego Windows będzie to *Windows installer (64-bit)*.

Rysunek 4. Lista wersji dla wybranego wydania Pythona

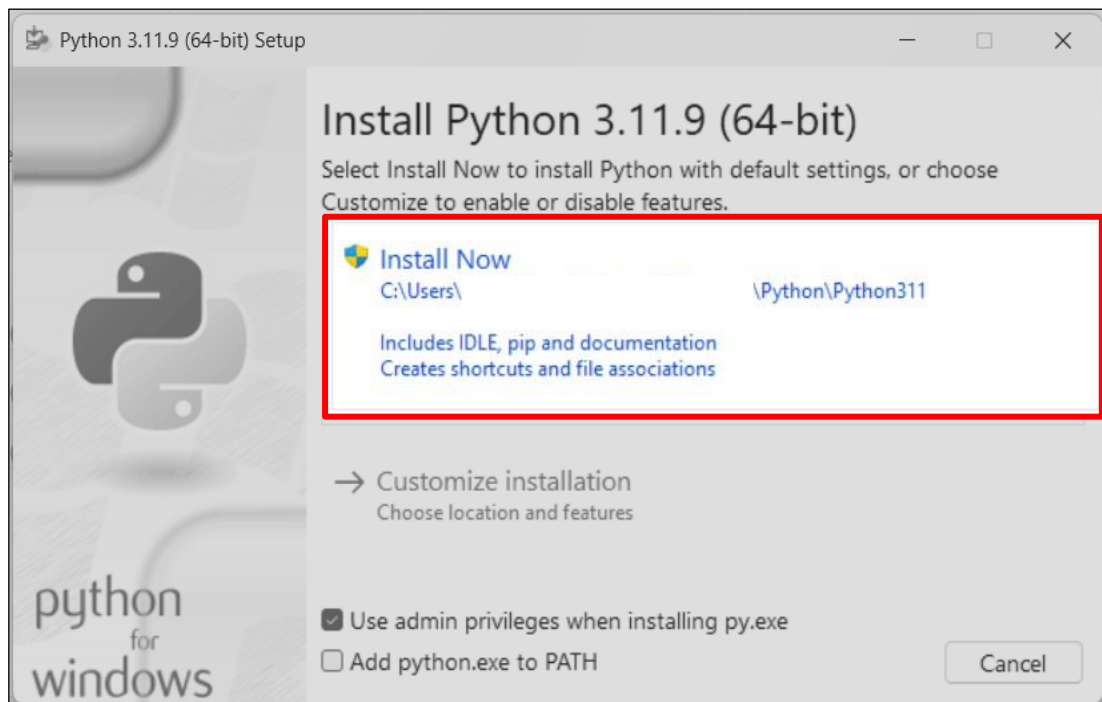


Files

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		bfd4d3bfeac4216ce35d7a503bf02d5c	25.3 MB	SIG	.sigstore
XZ compressed source tarball	Source release		22ea467e7d915477152e99d5da856ddc	19.2 MB	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	fa29f456feb6b5c4f52456a8b8ba347b	42.8 MB	SIG	.sigstore
Windows installer (64-bit)	Windows	Recommended	e8dcd502e34932eebcf1be056d5cbcd	25.0 MB	SIG	.sigstore
Windows installer (ARM64)	Windows	Experimental	328d93f71cb078965e4cfa2eb2663fa1	24.3 MB	SIG	.sigstore
Windows embeddable package (64-bit)	Windows		6d9aa08531d48fcc261ba667e2df17c4	10.7 MB	SIG	.sigstore
Windows embeddable package (32-bit)	Windows		31e7648158376e92a4463aa6f22a78e1	9.6 MB	SIG	.sigstore
Windows embeddable package (ARM64)	Windows		8611b6aa35483ab1c61d45e0d9f2de0d	10.0 MB	SIG	.sigstore
Windows installer (32-bit)	Windows		2a1d1ac2d8a0aa847515f9dd121ccbb7	23.8 MB	SIG	.sigstore

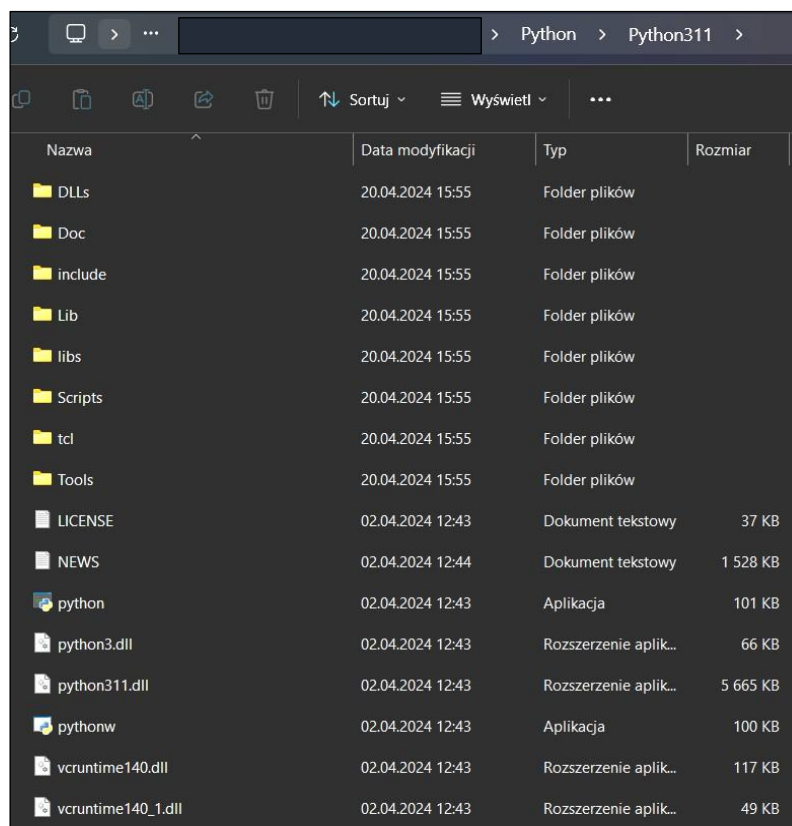
Po pobraniu, należy odnaleźć i uruchomić plik *python-3.11.x-amd64*, a następnie pozwolić instalatorowi na zainstalowanie rekomendowanych treści (opcja *Install Now*) lub samodzielnie wybrać ścieżkę i zawartość do wprowadzenia (*Customize installation*) (Rys. 5.).

Rysunek 5. Instalator Pythona



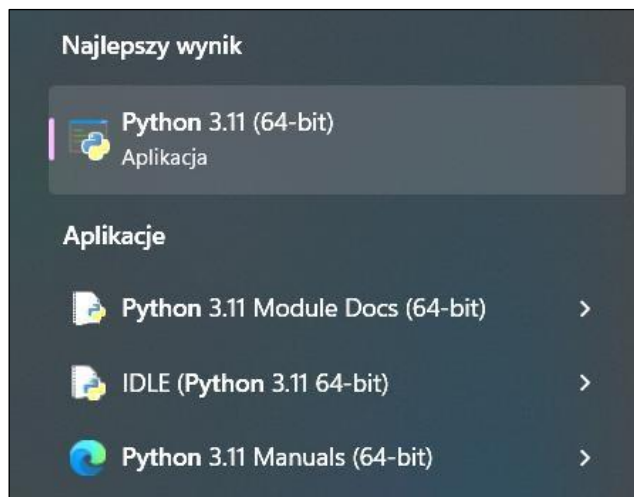
Po wykonaniu tej czynności, w podanej przy instalacji ścieżce, znaleźć będzie można zainstalowane pliki, w tym interpreter, w którym można wprowadzać i wykonywać kod. (Rys. 6.).

Rysunek 6. Folder z zainstalowanymi plikami Pythona



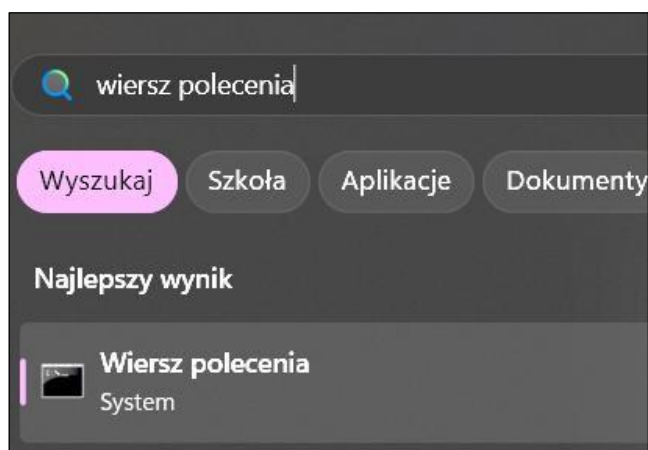
Pojawi się on również w wyszukiwarce systemowej (Rys. 7) wraz z listą obecnych modułów (Module Docs), środowiskiem, w którym można pisać kod (IDLE) oraz linkiem do dokumentacji (Manuals).

Rysunek 7. Wyniki wyszukiwania frazy "python" w wyszukiwarce systemowej po instalacji oprogramowania

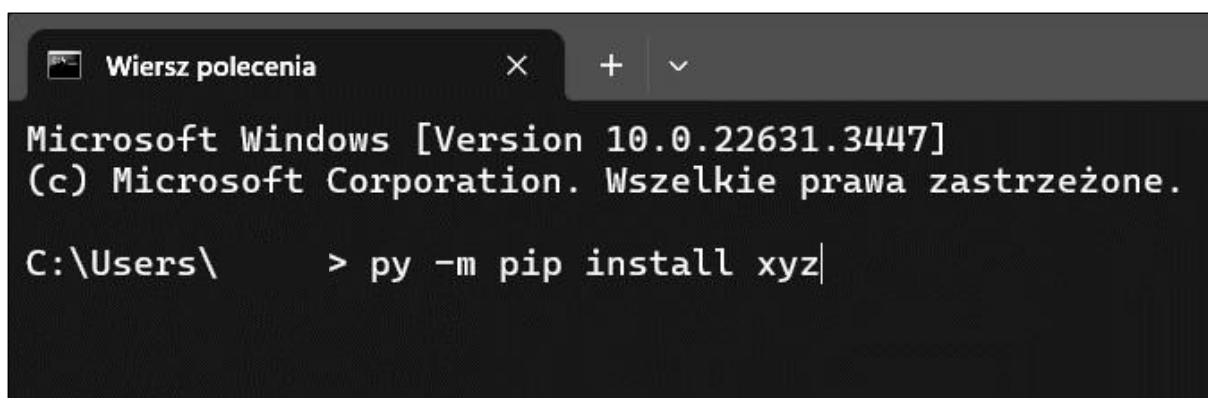


Po instalacji języka oraz odnalezieniu plików kolejnym krokiem jest dodanie modułów, które wykorzystuje *Sygnalista VAT*, a które nie są domyślnie dostępne. Aby tego dokonać należy otworzyć wiersz polecenia (ang. Command Prompt lub CMD) np. z wyszukiwarki systemowej (Rys. 8.) i wpisać ***py -m pip install xyz***, gdzie *xyz* to pożądane moduły takie jak: *pandas* (*numpy* również zostanie pobrany), *requests*, *xmltodict*, *sqlalchemy* oraz *pyqt5*. (Rys. 9.).

Rysunek 8. Wyniki wyszukiwania frazy "wiersz polecenia" w wyszukiwarce systemowej



Rysunek 9. Wiersz polecenia, w którym instalowane są moduły Pythona



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

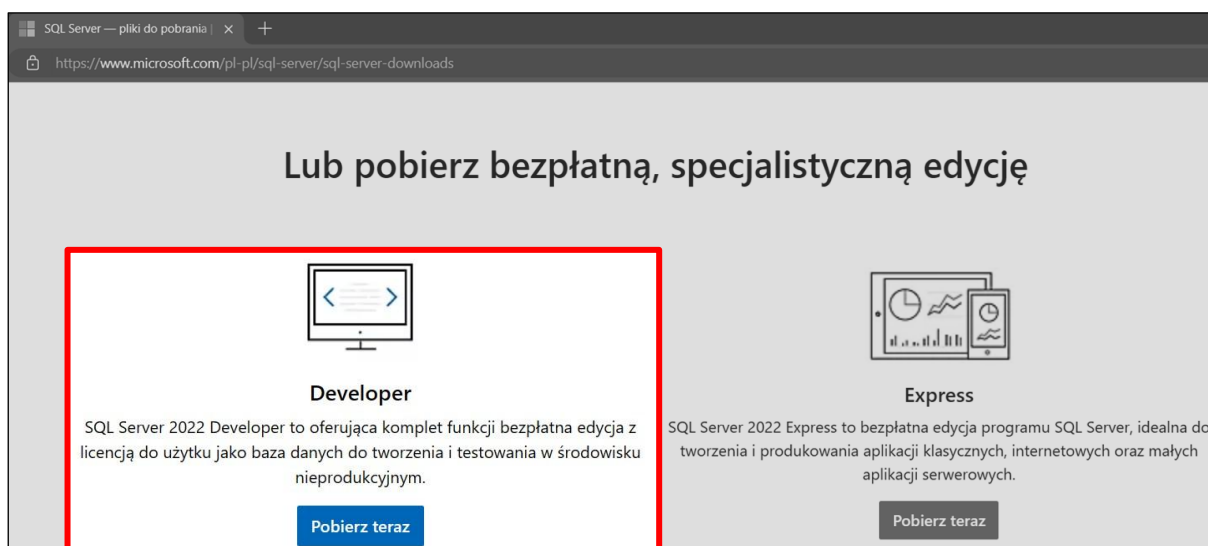
C:\Users\ > py -m pip install xyz|
```

Po zakończeniu tego procesu możliwe jest aktywowanie modułów w interpreterze oraz *Sygnaliście VAT* za pomocą polecenia `import [xyz]`. Wymagane do prawidłowego działania programu *Sygnalista VAT* wersje zarówno *Pythona* jak i jego bibliotek spisane zostały w pliku *requirements.txt* oraz załączniku nr 2.

3.1.2. MS SQL Server

Drugim wymaganym oprogramowaniem jest *Microsoft SQL Server*, który pobrać można na oficjalnej stronie firmy Microsoft²⁸. Wybrać można bezpłatną wersję *Developer* poprzez kliknięcie przycisku *Pobierz teraz* pod nazwą tego wydania (Rys. 10.).

Rysunek 10. Strona internetowa firmy Microsoft, gdzie pobrać MS SQL Server

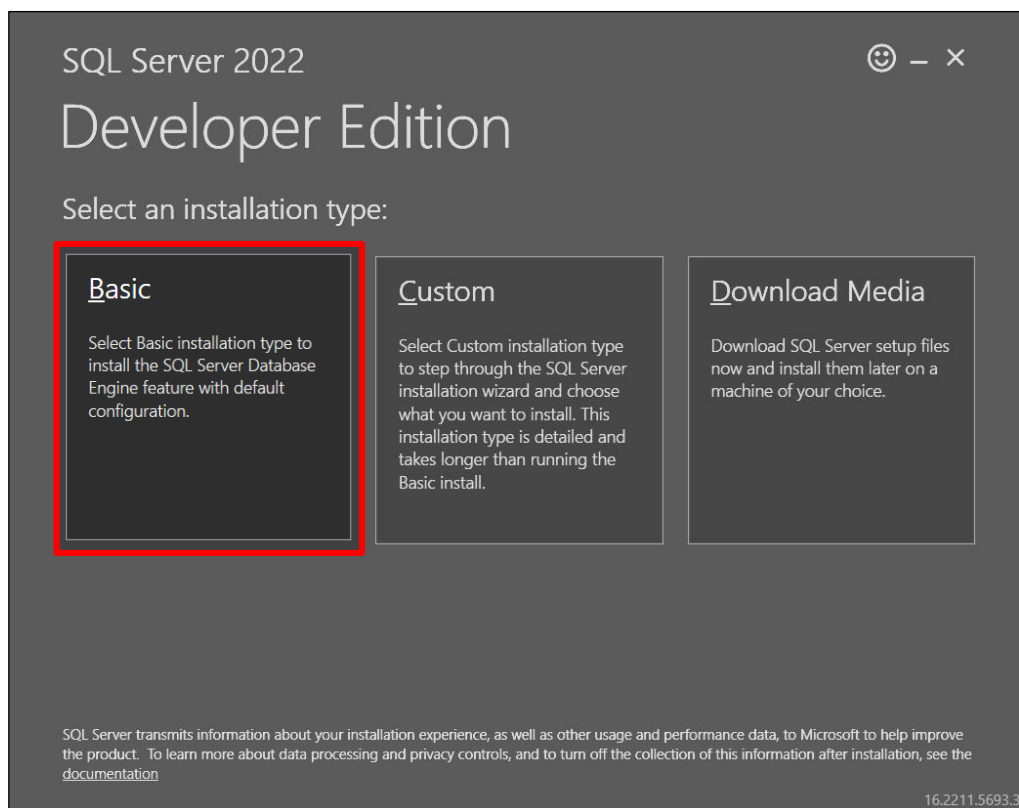


Po pobraniu, należy wyszukać i uruchomić plik **sql2022-ssei-dev**, aby rozpocząć instalację. Podobnie jak w przypadku *Pythona* dostępne możliwości instalatora to automatyczny proces z

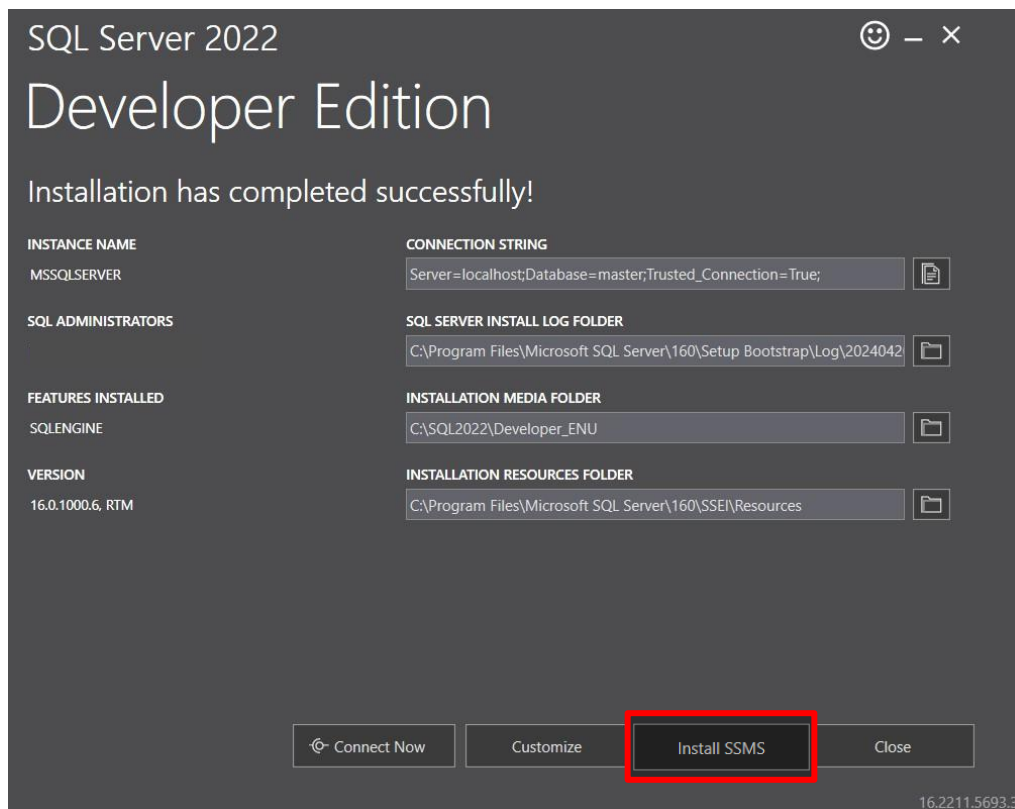
²⁸ <https://www.microsoft.com/pl-pl/sql-server/sql-server-downloads> [Dostęp 20.04.2024]

standardowymi opcjami (Basic) oraz samodzielny wybór pożądaných zawartości (Custom) (Rys. 11.).

Rysunek 11. Instalator oprogramowania MS SQL Server

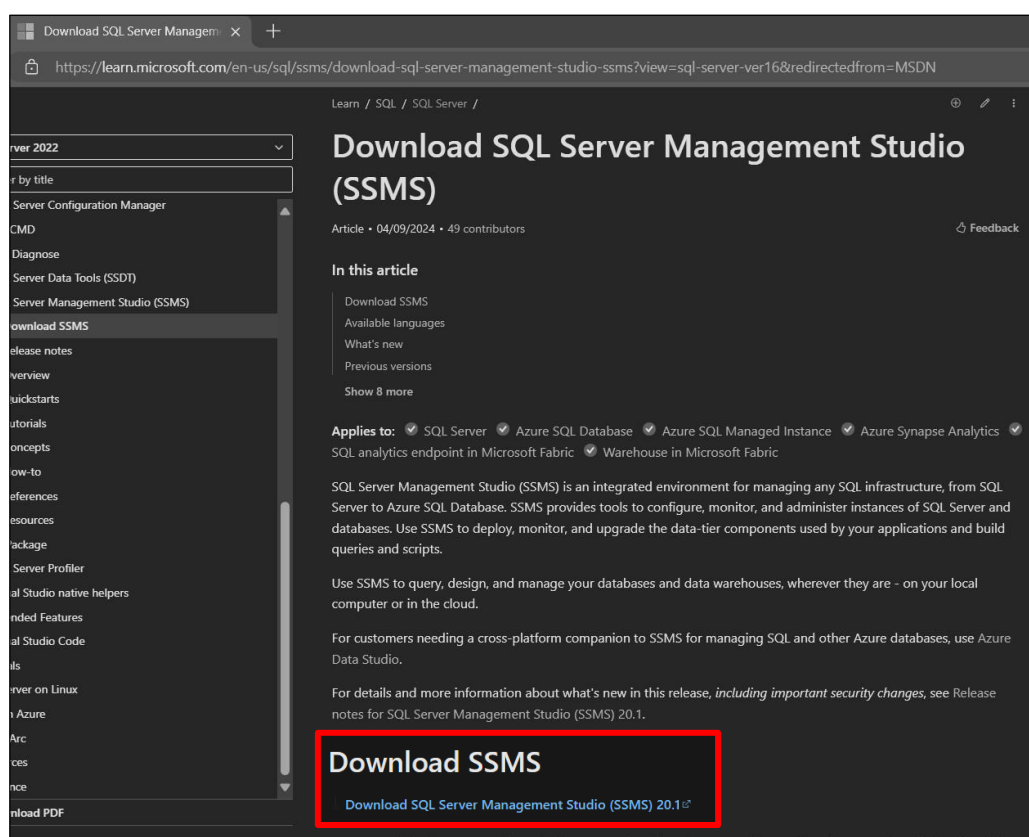


Rysunek 12. Ekran końcowy instalatora z opcją instalacji SQL Server Management Studio



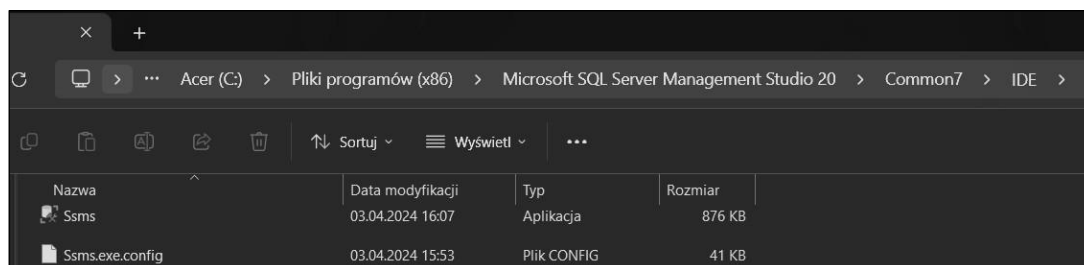
Po udanej instalacji *MS SQL Server* wyświetli się ekran końcowy (Rys. 12.) z możliwością instalacji dodatkowego oprogramowania *SQL Server Management Studio (SSMS)*. Jest to aplikacja pozwalająca m.in. zarządzać bazami danych na serwerze oraz tworzyć zapytania do nich. Po kliknięciu przycisku następuje przeniesienie do strony internetowej, gdzie pobrać można instalator (Rys. 13.).

Rysunek 13. Strona internetowa firmy Microsoft, na której można pobrać SSMS

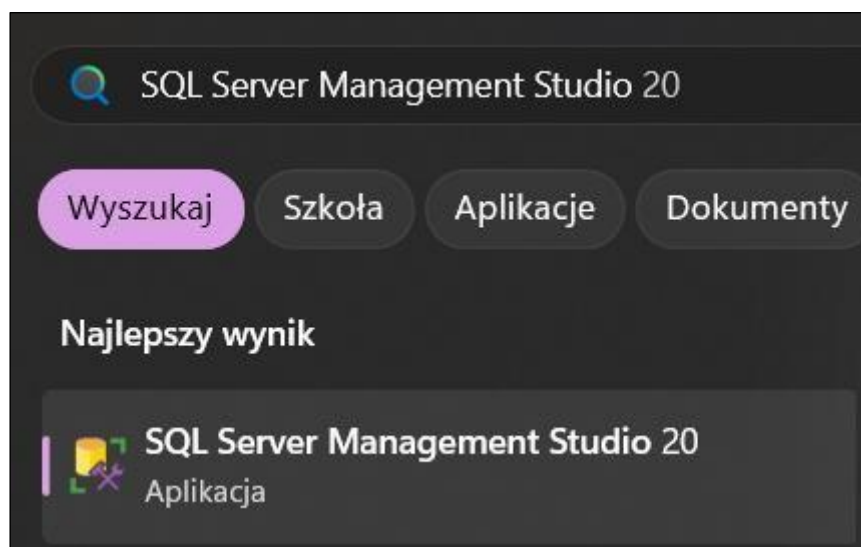


Po pobraniu pliku, należy go odszukać na urządzeniu i uruchomić a następnie postępować zgodnie ze wskazówkami instalatora. Program dostępny będzie w wybranym przy instalacji folderze (Rys. 14.) lub odnaleźć go można w wyszukiwarce systemowej (Rys. 15.).

Rysunek 14. Folder z plikami SSMS



Rysunek 15. Wyniki wyszukiwania frazy "SQL Server Management Studio" w wyszukiwarce systemowej



Tak skompletowane oprogramowanie pozwoli na budowę bazy danych oraz pracę z *Sygnalistą VAT*.

3.2. Instalacja programu

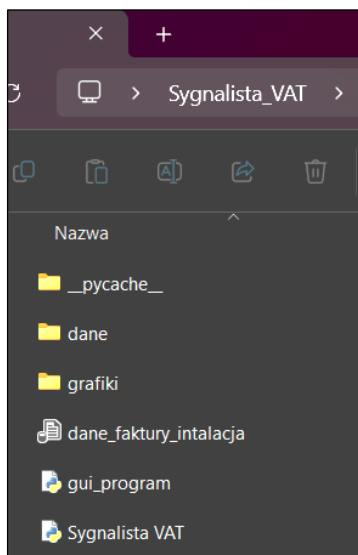
3.2.1. Folder z plikami

Wszystkie niezbędne do działania aplikacji pliki odnaleźć można w skompresowanym folderze *Sygnalista_VAT*. Należy rozpakować jego zawartość w dogodnym, wybranym miejscu na urządzeniu np. na pulpicie. W folderze odnaleźć można kilka elementów (Rys. 16.):

- Folder *dane* – zawierający pliki *csv* z danymi, które tworzą bazę oraz listy towarów i usług, użytkownik ma możliwość ich edycji, by np. dodać kolejną fakturę, przedsiębiorstwo itp. lub zaktualizować listy towarów podlegających danym stawkom w sytuacji zmiany w ustawie.
- Folder *grafiki* – zawierający pliki typu *png*, które program wyświetla jako logo czy strzałka powrotu, nie zalecane jest modyfikowanie tych elementów.
- Skrypt SQL *dane_faktury_instalacja* – zapytanie napisane w języku SQL, które pozwala utworzyć bazę danych na podstawie plików *csv* z folderu *dane*, jest on potrzebny w procesie opisanym w rozdziale 3.2.2.
- Kod interfejsu graficznego *gui_program* – plik, który aplikacja wykorzystuje do zbudowania interfejsu, z którym kontakt ma użytkownik, nie zalecane jest jego modyfikowanie.

- Aplikacja *Sygnalista VAT* – plik reprezentujący program, jego uruchomienie spowoduje otwarcie interfejsu i rozpoczęcie pracy z fakturami. Poza standardowym otwarciem poprzez podwójne kliknięcie można go otworzyć również w *IDLE* by edytować kod programu, co należy wykonać w późniejszej części instrukcji zgodnie ze wskazówkami.

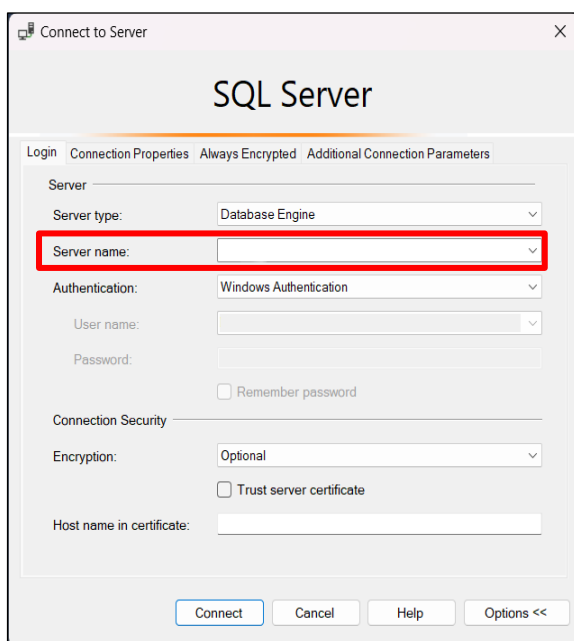
Rysunek 16. Folder z plikami, z których korzysta Sygnalista VAT



3.2.2. Utworzenie bazy danych

Po zapoznaniu z dostępnymi plikami kolejnym etapem przygotowującym do obsługi aplikacji jest wprowadzenie bazy danych na serwerze. W tym celu uruchomić należy program *SSMS* (Rys. 17.) i połączyć się z serwerem (konieczne jest zapisanie nazwy serwera zaznaczonej czerwoną ramką, która potrzebna będzie w dalszych krokach).

Rysunek 17. Ekran połączenia z serwerem wyświetlany przy uruchomieniu SSMS



Następnie wskazane jest otwarcie pliku *dane_faktury_intalacja* dostępnego w folderze z programem *Sygnalista VAT* i postępowanie zgodnie z instrukcją zawartą w tym skrypcie (Rys. 18.).

Rysunek 18. Fragment skryptu SQL ze wskazówkami

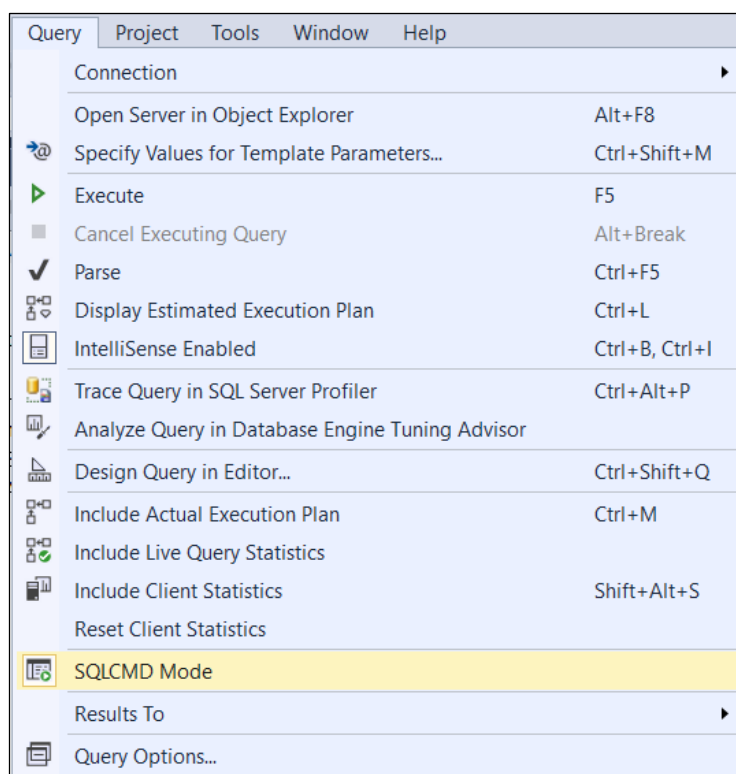
```
/* Witaj!
   Aby dodać bazę danych do programu faktura; w zakładce "Query" wybierz opcję "SQLCMD Mode", a następnie
   ustaw odpowiednią ścieżkę do folderu z danymi poniżej według wzoru: */

:setvar SqlSamplesSourceDataPath "C:\...tutaj podaj ścieżkę...\Sygnalista_VAT\dane\"

-- Teraz uruchom zapytanie klikając "Execute"
```

Należy najpierw w zakładce *Query* odnaleźć i wybrać opcję *SQLCMD Mode* (Rys. 19.), później podać ścieżkę do folderu, w którym znajdują się dane do wgrania na serwer z końcówką *\SygnalistaVAT\dane* i uruchomić zapytanie poprzez *Execute* lub klikając klawisz *F5*. Na serwerze zbudowana zostanie baza danych, z której korzystać będzie *Sygnalista VAT*. Skrypt można zapisać, jednak nie istnieje taka potrzeba.

Rysunek 19. Opcje w zakładce Query SSMS



Ostatnim krokiem przygotowującym do uruchomienia programu jest podanie nazwy serwera, na którym znajduje się baza danych. Uruchomić należy *Sygnalistę VAT* w *IDLE* i odnaleźć wskazówkę w pierwszej linijce kodu oraz wkleić lub wpisać nazwę serwera w drugiej tak by znajdowała się ona w cudzysłowie, a następnie zapisać i wyłączyć *IDLE*.

W tym miejscu podaj nazwę serwera z SQL Server Management Studio:
SQL_serwer = r"nazwa_serwera"

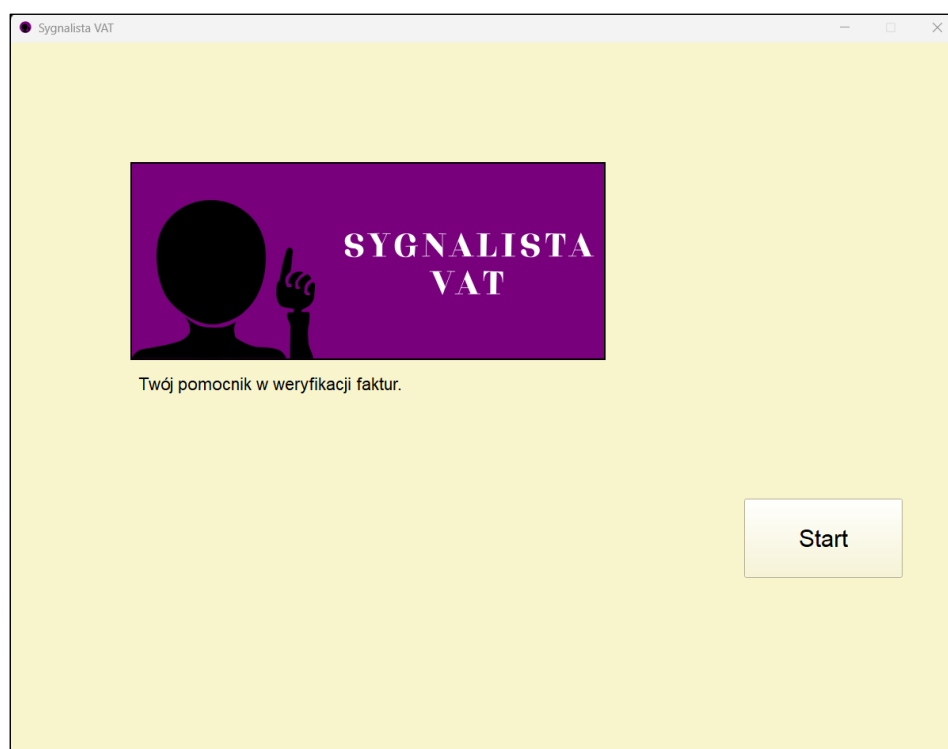
Zakłada się, że wraz z SSMS zainstalowane zostaną odpowiednie sterowniki. *Sygnalista VAT* operuje na *Microsoft ODBC Driver 17 for SQL Server*, pozwalającym współpracować z serwerem. Należy upewnić się, że jest on zainstalowany i widoczny na liście programów i funkcji w panelu sterowania. Jeżeli nie, konieczna będzie jego instalacja²⁹.

3.3. Obsługa programu

3.3.1. Uruchamianie

Proces obsługi aplikacji ukazany został na schemacie blokowym programu od strony interfejsu w załączniku nr 3 oraz od strony kodu w załączniku nr 4 (jak łączą się te procesy wskazują ścieżki opisane literami A-E). W celu uruchomienia programu należy kliknąć dwukrotnie lewym przyciskiem myszy na plik *Sygnalista VAT* lub jednokrotnie prawym przyciskiem myszy i wybrać opcję *Otwórz*. Wykonanie tych czynności skutkować będzie wyświetleniem interfejsu graficznego z ekranem powitalnym (Rys. 20.). Znaleźć na nim można logo, nazwę, a także przycisk *Start* kierujący do kolejnego widoku (ścieżka A na schemacie blokowym).

Rysunek 20. Ekran startowy *Sygnalisty VAT*

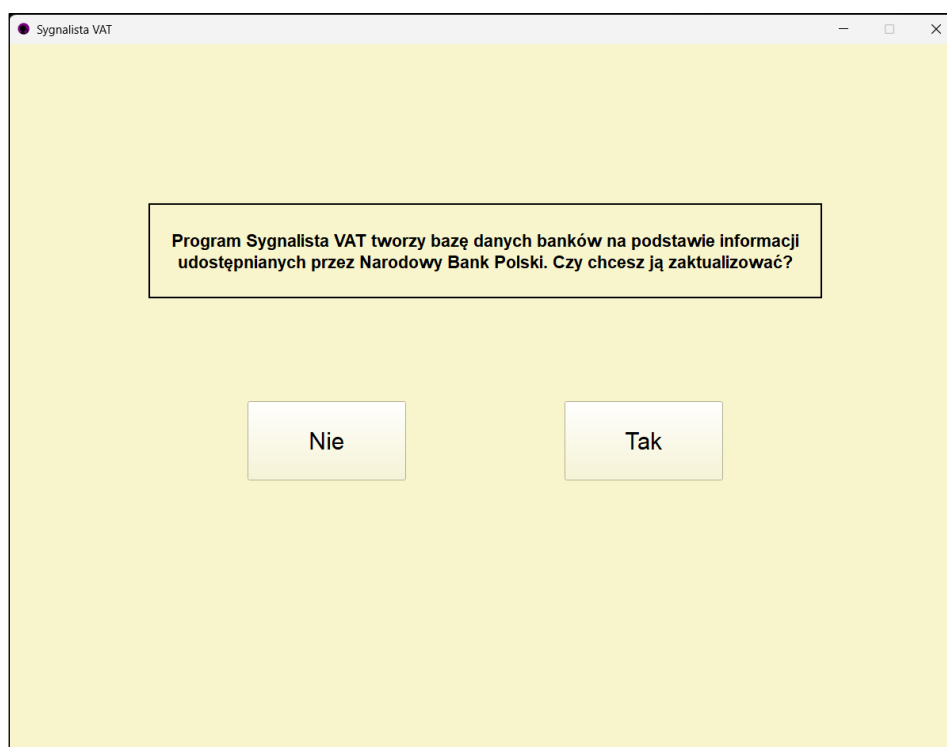


²⁹ <https://learn.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver16> [Dostęp 20.04.2024]

3.3.2. Aktualizacja bazy danych

Na kolejnym ekranie użytkownik dowiaduje się o wykorzystaniu zasobów informacyjnych NBP i możliwości ich aktualizacji w bazie danych oraz postawiony zostaje przed wyborem czy ma to zostać dokonane (Rys. 21.). Kliknięcie przycisku *Nie* skutkować będzie pominięciem kroku, baza nie zostanie odświeżona, natomiast po wyborze *Tak* nastąpi proces trzykrotnej próby uaktualnienia (ścieżka B). W obu wypadkach wynikiem będzie przejście do następnego widoku (ścieżka C).

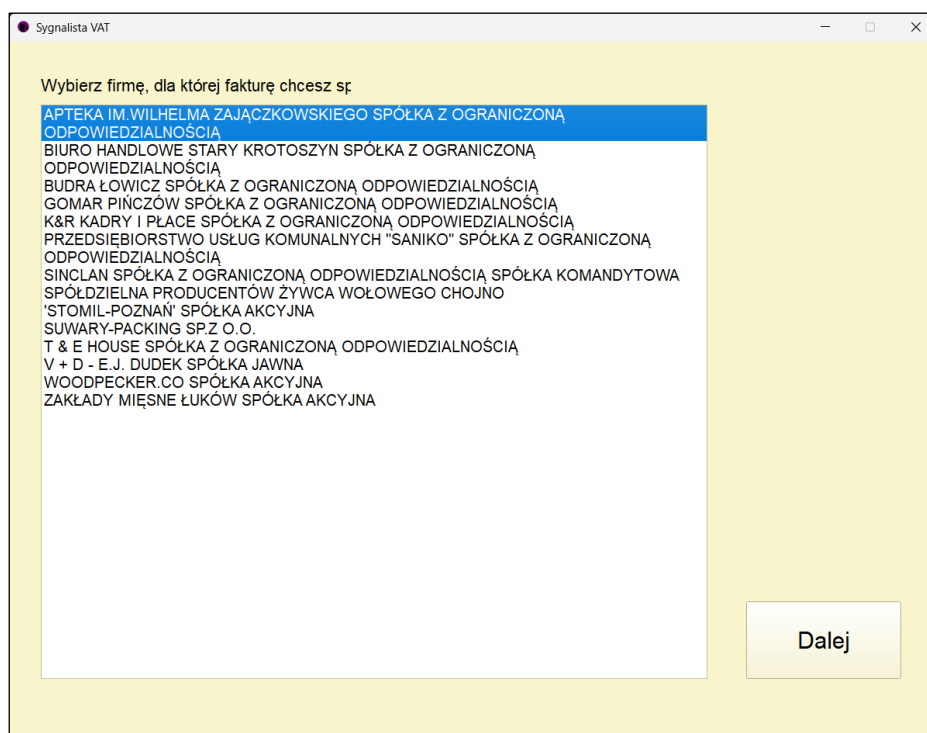
Rysunek 21. Ekran z opcją aktualizacji bazy danych banków



3.3.3. Wybór faktury

Przed użytkownikiem pojawi się lista firm obecnych w zintegrowanej z programem bazie, z której użytkownik wybrać może przedsiębiorstwo będące wystawcą faktur (Rys. 22.). Zaleca się kliknięcie na wybraną pozycję, przy czym możliwa jest zmiana decyzji poprzez kliknięcie na inną, a następnie przycisku *Dalej* (ścieżka D).

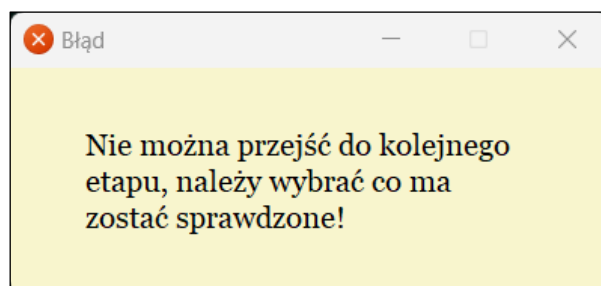
Rysunek 22. Ekran z wyborem przedsiębiorstwa



Kolejna będzie lista faktur wystawiona przez wybrane przedsiębiorstwo, zasady działania są takie same jak w przypadku zestawienia nazw firm.

Jeżeli nie zaznaczona zostanie żadna nazwa lub numer faktury i kliknięty przycisk *Dalej* pojawi się okno *Błąd* z informacją nakazującą wyboru przed przejściem do kolejnego etapu (Rys. 23.).

Rysunek 23. Okno z komunikatem błędu



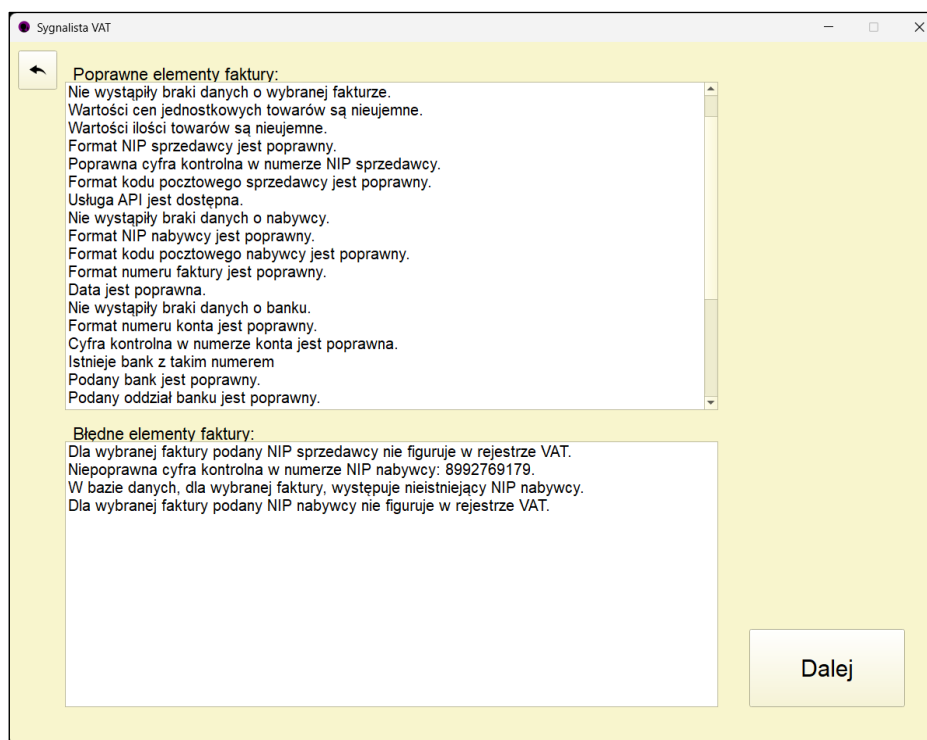
Co więcej, widok z wyborem faktury zawiera nową opcję tzn. powrotu do poprzedniego etapu w przypadku pomyłki lub chęci weryfikacji innego dokumentu. Oznaczona została ona strzałką z grotem skierowanym w lewą stronę i umieszczona w lewym górnym rogu. Pozwala ona w każdym kolejnym etapie powrót do poprzedzającego okna aż do listy przedsiębiorstw.

3.3.4. Wyniki weryfikacji

Gdy użytkownik zdecyduje się jaką fakturę chciałby zweryfikować i kliknie przycisk *Dalej*, rozpocznie się proces jej sprawdzania (ścieżka E), czego skutkiem są zestawienia

elementów, które okazały się być poprawne oraz błędne (Rys. 24.). Ten widok zawiera widżety wyłącznie informacyjne, kliknięcie na jakąkolwiek pozycję na listach nie będzie miało żadnego wpływu. Możliwe jest jedynie przejście do kolejnego widoku dzięki przyciskowi *Dalej* lub powrotu poprzez kliknięcie strzałki w lewym górnym rogu. Dzięki wylistowaniu błędnych elementów na fakturze użytkownik jest w stanie zorientować się co może być zagrożeniem lub co należy dalej zgłębić by błąd nie pojawił się w przyszłości.

Rysunek 24. Ekran z listami poprawnych i błędnych elementów faktur



Po kliknięciu przycisku *Dalej* wyświetli się końcowy widok z wydrukiem wybranej faktury zgodny z wymogami ustawy o elementach, które powinny się na niej znajdować (Rys. 25.). Zawiera on wyraźny podział na grupy – informacje o sprzedawcy, nabywcy i banku, jeżeli takowe są w bazie i/lub potrzebne, wartości identyfikujące dokument, tabele z wyliczeniami oraz kwota do zapłaty. Poza możliwością oglądu wydruku dostępny jest jedynie powrót do poprzedniego ekranu lub zakończenie pracy z *Sygnalistą VAT*.

Rysunek 25. Okno z wydrukiem faktury

←

Tvoja faktura:

Faktura

Faktura nr: 039/02/2022
Z dnia: 09.02.2022

Sprzedawca

Nazwa: APTEKA IM. WILHELM A ZAJĄCZKOWSKIEGO SPÓŁKA Z OGRANICZONĄ ODPOWIEDZIALNOŚCIĄ
Adres: Słowackiego 10, 38-100 Strzyżów
NIP: 9650126791

Nabywca

Nazwa: WOODPECKER.CO SPÓŁKA AKCYJNA
Adres: ul. Krakowska 29D, 50-424 Wrocław
NIP: 899-27-69-179

Towar lub usługa	Ilość	Cena	Wartość netto	Stawka podatku	Kwota podatku	Wartość brutto
1 urządzenia RTV i AGD	39	9.22	359.58	23%	82.7	442.28
2 półprzewodnikowe urządzenia pamięci trwałej	10	98.88	988.8	8%	79.1	1067.9
3 nuty drukowane	1	28.0	28.0	5%	1.4	29.4
4 zboża	20	68.99	1379.8	5%	68.99	1448.79
5 płyty gramofonowe	8	5.5	44.0	zw.	0.0	44.0

Bank

Numer konta: 93 9168 0004 9730 4487 9412 6080
W banku: Bank Spółdzielczy w Strzyżowie Centrala

Kwota do zapłaty: 3032.37 PLN
Słownie: trzy tysiące trzydzieści dwa 37/100 PLN

	Wartość netto	Stawka podatku	Kwota podatku	Wartość brutto
1	359.58	23%	82.7	442.28
2	988.8	8%	79.1	1067.9
3	1407.8	5%	70.39	1478.19
4	0.0	0%	0.0	0.0
5	44.0	zw.	0.0	44.0
Suma	2800.18	-	232.19	3032.37

4. Implementacja

4.1. Bazy danych - banki

Jak przedstawione zostało wcześniej, *Sygnalista VAT* wykorzystuje w swoim działaniu możliwości jakie daje API. Jak przebiega ten proces, wraz z sposobami ekstrakcji wartości z zagnieżdżonych obiektów ukazany został poniżej na przykładzie danych o bankach.

Etap tworzenia bazy banków rozpoczyna się od połączeniem ze źródłem danych, tj. stroną internetową NBP i wskazaniem pliku z rozszerzeniem xml, jaki ma zostać pobrany. Jako zmienna „url” zapisany zostaje adres *https* wraz z pojawiającym się po znaku zapytania kluczem z nazwą interesującego nas pliku. Taka zmienna tekstowa jest następnie przekazywana do funkcji *request.get()*, która łączy program ze stroną banku centralnego, a następnie pobrana zostaje wskazana zawartość (odpowiedź). Z pomocą funkcji *xmltodict.parse()* dane w formacie xml przekształcane są w słownik. Zawiera on w sobie wyłącznie jeden element tj. słownik „Instytucje”, a ten zawiera kolejny słownik o nazwie „Instytucja”. W celu optymalizacji, aby uniknąć odwoływania się do tych kluczy w każdym kolejnym poleceniu, zagnieżdżenia te zostają pomięte poprzez nadpisanie oryginalnego obiektu słownikiem „Instytucja”. Ostatnią częścią przygotowania do gromadzenia danych jest utworzenie trzech pustych list, w których wpisane zostaną dane tj. nazwy banków, numery i nazwy oddziałów.

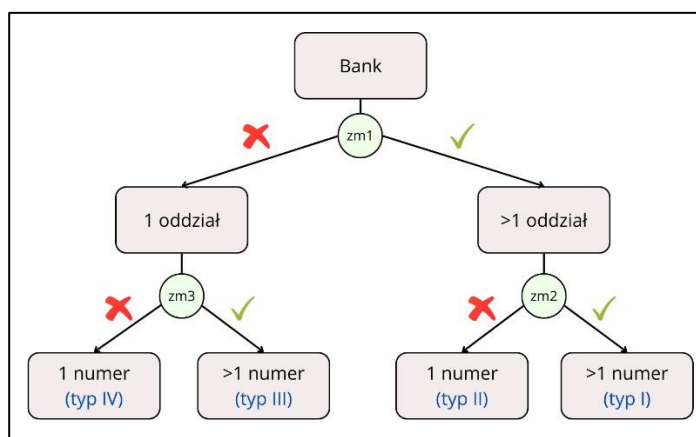
```
# Pobranie danych:
# adres strony, z której pobierane są dane
url = r"https://ewib.nbp.pl/plewiba?dokNazwa=plewiba.xml"
# połączenie ze stroną
response = requests.get(url, allow_redirects=True)
# zapisanie zawartości (pliku xml z danymi) jako słownik
banki_dane = xmltodict.parse(response.content)

# Przygotowanie do spisywania danych:
# zejście dwa poziomy niżej w zagnieżdżeniu zapisu danych
banki_dane = banki_dane["Instytucje"]["Instytucja"]

# listy, w których zapisane zostaną informacje o kolejnych bankach
lista_bankow = []
lista_nr_bankow = []
lista_oddzialow = []
```

Działanie algorytmu do ekstrakcji danych o bankach obrazuje rysunek 26. Poprzez pętlę program przechodzi przez każdy element słownika – dla każdego banku sprawdzane jest czy posiada dokładnie jedną, czy więcej niż jedną jednostkę, a następnie każda z nich podlega takiej samej weryfikacji pod względem liczby numerów do niej przypisanych.

Rysunek 26. Schemat rozpoznawania typu zagnieżdżenia danych o bankach



Źródło: opracowanie własne w Canvie

Możliwe są 4 typy zagnieżdżeń (zapisania informacji o numerach banków) w zależności od liczby oddziałów i numerów. Sprawdzane są one poprzez próby odwołania się do elementów niższego szczebla zagnieżdżenia (zm1, zm2, zm3), jeżeli istnieje więcej niż jeden oddział/numer to zapisane są one w liście. Odwołanie się do pierwszego elementu (zerowego indeksu) nie wywoła błędu, chyba że zamiast listy na niższym szczeblu znajdzie się słownik z wyłącznie jednym oddziałem/numerem. Obsługa tego błędu jest swoistym warunkiem, który kontroluje, z jakim typem zagnieżdżenia program ma do czynienia w danej iteracji.

Spisywanie danych kolejnych banków i ich oddziałów:

test na istnienie wielu oddziałów

```
zm1 = banki_dane[bank]["Jednostka"][0]
```

test na istnienie wielu numerów, jeżeli bank ma kilka oddziałów

```
zm2 = banki_dane[bank]["Jednostka"][oddzial]["NumerRozliczeniowy"][0]
```

test na istnienie wielu numerów, jeżeli bank ma jeden oddział

```
zm3 = banki_dane[bank]["Jednostka"]["NumerRozliczeniowy"][0]
```

W zależności od powodzenia prób następuje ekstrakcja danych według danego typu (Rys. 27.):

- I – wiele numerów jednego z wielu oddziałów – najbardziej zagnieżdżony sposób zapisu danych, ekstrakcja, jeżeli powiodły się testy *zm1* i *zm2*, powstają rekordy o wspólnej nazwie banku i oddziału;
- II – jeden numer jednego z wielu oddziałów – ekstrakcja, jeżeli powiódł się test *zm1*, ale nie *zm2*, powstają rekordy o wspólnej nazwie banku, ale nie oddziału;
- III – wiele numerów jednego oddziału – ekstrakcja, jeżeli nie powiódł się test *zm1*, a *zm3* tak, powstają rekordy o wspólnej nazwie banku i oddziału;
- IV – jeden numer jednego oddziału – ekstrakcja, jeżeli nie powiódł się testy *zm1* i *zm3*, powstaje jeden rekord.

Wydobyte informacje o jednostkach (bank, nazwa, numer) wprowadzane są w każdej iteracji do list poleceniem `.append()`.

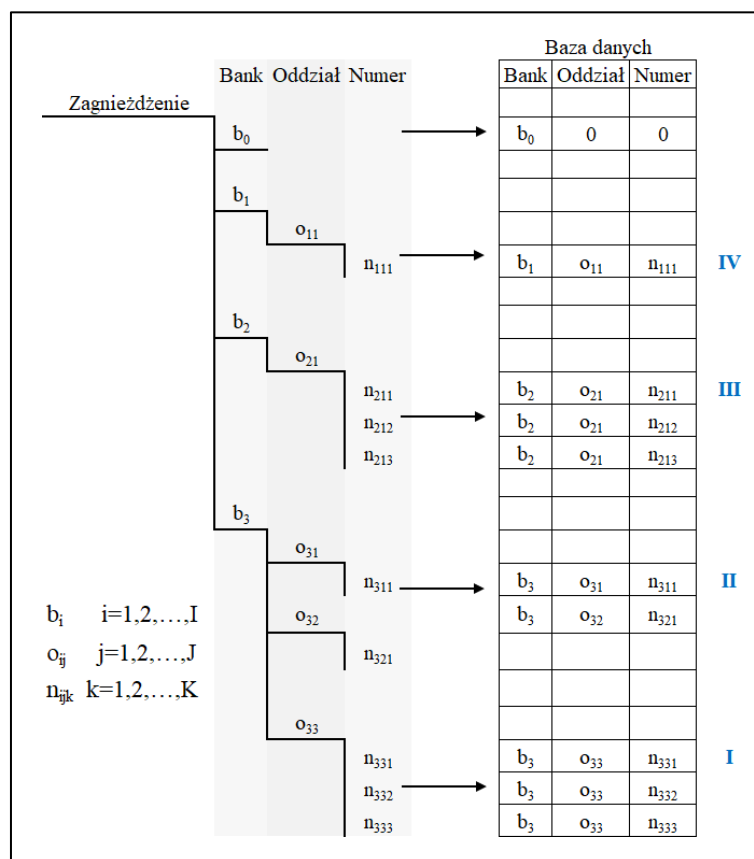
```
# zagnieżdżenie typu I – bank posiada kilka oddziałów i jeden z nich posiada kilka numerów
lista_bankow.append(banki_dane[bank]["NazwaInstytucji"])
lista_nr_bankow.append(banki_dane[bank]["Jednostka"][oddzial]["NumerRozliczeniowy"]
                        [nr_roz]["NrRozliczeniowy"])
lista_oddzialow.append(banki_dane[bank]["Jednostka"][oddzial] ["NumerRozliczeniowy"]
                       [nr_roz]["NazwaNumeru"])
```

```
# zagnieżdżenie typu II – bank posiada kilka oddziałów i jeden z nich posiada tylko jeden numer
lista_bankow.append(banki_dane[bank]["NazwaInstytucji"])
lista_nr_bankow.append(banki_dane[bank]["Jednostka"][oddzial]["NumerRozliczeniowy"]
                        ["NrRozliczeniowy"])
lista_oddzialow.append(banki_dane[bank]["Jednostka"][oddzial]["NumerRozliczeniowy"]
                       ["NazwaNumeru"])
```

```
# zagnieżdżenie typu III – bank posiada tylko jeden oddział, który posiada kilka numerów
lista_bankow.append(banki_dane[bank]["NazwaInstytucji"])
lista_nr_bankow.append(banki_dane[bank]["Jednostka"]["NumerRozliczeniowy"][nr_roz]
                        ["NrRozliczeniowy"])
lista_oddzialow.append(banki_dane[bank]["Jednostka"]["NumerRozliczeniowy"][nr_roz]
                       ["NazwaNumeru"])
```

```
# zagnieżdżenie typu IV – bank posiada tylko jeden oddział, który posiada tylko jeden numer
lista_bankow.append(banki_dane[bank]["NazwaInstytucji"])
lista_nr_bankow.append(banki_dane[bank]["Jednostka"]["NumerRozliczeniowy"]
                        ["NrRozliczeniowy"])
lista_oddzialow.append(banki_dane[bank]["Jednostka"]["NumerRozliczeniowy"]
                       ["NazwaNumeru"])
```

Rysunek 27. Schemat ekstrakcji danych ze słowników/list do ramki danych według różnych typów zagnieżdżeń



Źródło: opracowanie własne w MS Excel

Jeżeli dany oddział banku nie posiada żadnego numeru, to zamiast jego nazwy i numeru wpisywana jest wartość 0, rekordy takie zostają w późniejszym etapie usunięte z bazy. Po zebraniu informacji o wszystkich podmiotach, dane zostają zgromadzone w postaci ramki danych poleceniem `pd.DataFrame()`, a następnie wprowadzone na serwer SQL oraz mogą zostać wykorzystane do sprawdzania poprawności informacji o bankach.

sytuacja, gdy oddział nie posiada żadnego numeru

```
lista_nr_bankow.append(0)
```

```
lista_oddzialow.append(0)
```

zebranie danych do ramki danych z kolumnami nazwa banku, nazwa i numer oddziału

```
baza_banki = pd.DataFrame({"Bank": lista_bankow, "Oddzial_banku": lista_oddzialow,
                           "Nr_banku": lista_nr_bankow})
```

4.2. Proces weryfikacji

Dysponując bazą danych przystąpić można do najważniejszej części Sygnalisty VAT, czyli etapu sprawdzenia poprawności elementów faktury. Kod za tym stojący przedstawiono we fragmentach poniżej. Każdy element faktury poddawany jest weryfikacji za pomocą własnych funkcji zdefiniowanych przy pomocy słowa kluczowego `def`. Po wyborze faktury,

dane do niej przypisane są podawane jako argumenty w funkcjach. Każda z nich dodaje komentarz do listy poprawnych lub błędnych elementów, wskazujący czy wartości na fakturze są prawidłowe oraz co należy poprawić, jeżeli nie są.

Pierwszym przykładem takiej weryfikacji może być test braków w określonych grupach danych np. wartości liczbowe, informacje o nabywcy, sprzedawcy, banku czy samej fakturze (numer, data, rodzaj, specjalne napisy). Puste wartości w bazie mogą spowodować błędy, przez które nie możliwa będzie dalsza analiza poprawności danych i obliczenia. Brak NIP lub numeru konta spowoduje, że nie zweryfikowane zostaną powiązane z nimi dane, a nieobecność wartości liczbowych lub ich ujemne wartości (w szczególności ilości) spowodują, że na fakturze pojawią się kwoty podatku i wartości brutto, które zniekształcą kwotę do zapłaty i wartość podatku do odprowadzenia. Błędy drugiego typu są szczególnie groźne, dlatego jeżeli się pojawiają, to program zapisuje tę informację w postaci zmiennej *ilosc_bledow*. Każda nieprawidłowość zwiększa jej wartość o 1, w związku z tym, jeżeli nie wynosi ona 0 na koniec procesu weryfikacji, to nie wykonane zostaną obliczenia, a tabele z towarami lub usługami oraz agregacją nie zostaną wypełnione.

W poniższym przykładzie sprawdzane jest, czy wśród danych liczbowych wystąpiły puste wartości. Jako argument *tabelka* wprowadzony zostaje wycinek bazy danych – wyłącznie kolumny z cenami i ilościami. Obiekt przekazany do takiej funkcji wykorzystywany jest kolejno w poleceniach *pd.isna()*, która zwraca macierz wartości prawda/fałsz (przypadku braku/obecności wartości) a następnie w *.any()*, która zwraca prawdę, jeżeli jakikolwiek element tej macierzy jest również prawdą. Wynik tych poleceń służy dalej w funkcji warunkowej, która w przypadku zajścia powyższego warunku dodaje komentarz o brakach do listy błędów oraz zwiększa wartość zmiennej *ilosc_bledow* (która widoczna jest w dowolnym miejscu, nie tylko w tej funkcji, dzięki słowu kluczowemu *global*). W innym wypadku komentarz o poprawności trafia do drugiej listy.

```
# funkcja sprawdzająca, czy występują braki danych liczbowych, które uniemożliwiłyby
# policzenie wartości brutto
def check_braki_liczby(tabelka):
    if pd.isna(tabelka).any(axis=None):
        lista_bledne.append("W bazie, dla wybranej faktury, wystąpiły braki danych
                             liczbowych.")
        global ilosc_bledow
        ilosc_bledow += 1
    else:
        lista_poprawne.append("Nie wystąpiły braki danych liczbowych.")
    return
```

Kolejnym przykładem sprawdzanego elementu jest data wystawienia faktury. Oczekuje się, że nie będzie ona niemożliwa tj. podany miesiąc będzie miał wartość nie większą niż 12, a dzień 29/30/31 w zależności od miesiąca. Zakłada się przy tym, że stosowany jest zapis w formacie „ddmmyyyy”, w którym najpierw podaje się dzień a następnie miesiąc, znak separujący nie ma znaczenia. Z zapisanej w postaci ciągu znaków daty separowane są trzy wartości reprezentujące dzień, miesiąc i rok, które następnie zapisane zostają jako liczby całkowite (ang. *integer* – *int()*). Zmienne te traktuje się jako globalne, ponieważ korzysta z nich również inna funkcja. Poprzez polecenie *.datetime()* z biblioteki *datetime* podejmowana jest próba utworzenia daty z wartości rok, miesiąc i dzień. Wynik tego testu zapisywany jest jako komentarz na liście poprawnych lub błędnych elementów oraz wartością logiczną zmiennej *zla_data* (*prawda* tj. *Ture* oznacza błąd).

funkcja sprawdzająca, czy data przypisana do wybranej faktury jest poprawna, wykorzystując do tego bibliotekę datetime

```
def check_data(data):
    global dzien, miesiac, rok
    dzien = int(data[0:2])
    miesiac = int(data[3:5])
    rok = int(data[6:10])
    while True:
        try:
            data2 = datetime.datetime(year=rok, month=miesiac, day=dzien)
            lista_poprawne.append(„Data jest poprawna.”)
            global zla_data
            zla_data = False
            break
        except ValueError:
            zla_data = True
            lista_bledne.append(f”W bazie danych, dla wybranej faktury, podano błędną datę: „
                                f”{data}. Dostępny format: ddmrrrrr (separator „.”, „-”, „/”), „
                                f”dni z przedziału 01-31, miesiące z przedziału 01-12”)
            break
    return
```

Sygnalista VAT weryfikuje także formaty zapisu różnych danych np. NIP (oddzielony myślnikami, spacjami lub nie oraz jako ciąg dziesięciu liczb „xxx-xx-xx-xxx”), kod pocztowy (jako „xx-xxx”), numer konta (26 cyfr, z których dwie pierwsze zapisane są dwójkami a pozostałe czwórkami tj. „xx xxxx xxxx...”). Wykorzystywane są w tym celu możliwości wyrażeń regularnych (*regex*) z pakietu *re*, a konkretnie funkcja *.fullmatch()*, w której argumentem jest ciąg znaków do wyszukania oraz tekst do przeszukania. Jeżeli udaje się je dopasować, to funkcja zwraca szukany fragment tekstu, w innym wypadku zwraca brak. W związku z tym pusta wartość oznacza błędny element, a niepusta poprawny, które to komentowane są w listach. Całość zapisana jest oczywiście we własnej funkcji, która może

zostać wykorzystana dla nabywcy lub sprzedawcy, dlatego korzysta ona dodatkowo z argumentu „kogo” by wskazać, jaki podmiot poddawany jest analizie.

```
# funkcja sprawdzająca, czy kod pocztowy ma poprawny format, wykorzystując wyrażenia
regularne
def check_kod_format(kod, kogo):
    if re.fullmatch(r"\b\d{2}-\d{3}\b", kod) is None:
        lista_bledne.append(f"W bazie danych, dla wybranej faktury, podano błędny format kodu"
                             f"pocztowego {kogo}: {kod} Dostępny format: xx-xxx.")
    else:
        lista_poprawne.append(f"Format kodu pocztowego {kogo} jest poprawny.")
    return
```

NIPy oraz numery kont bankowych zawierają w sobie cyfry kontrolne, które związane są z procesem ich tworzenia. Po nadaniu im odpowiedniego „id” dodawane są do nich cyfry, które będą pomagały weryfikować poprawność całości. Przykładem może być cyfra kontrolna w NIP testowana przez funkcję poniżej. Na wstępie ciąg znaków oczyszczany jest z myślników oraz spacji, a kolejne cyfry spisane są w liście poprzez pętlę. Wagi zapisane zostają w formie krotki. Zsumowane zostają iloczyny pierwszych dziewięciu cyfr i wag a następnie wartość ta dzielona jest przez 11. Reszta z tego dzielenia powinna równać się ostatniej cyfrze NIP. Taki proces weryfikacji jest powszechnie stosowany, o czym świadczyć może dostępność kodów w zasobach Wikipedii³⁰.

```
# funkcja sprawdzająca, czy nip ma poprawną cyfrę kontrolną
def check_nip(nip_b, kogo):
    nip_b = nip_b.replace(" ", "").replace("-", "")
    cyfry = [int(cyfra) for cyfra in nip_b]
    wagi = (6, 5, 7, 2, 3, 4, 5, 6, 7)
    if sum(cyfra * waga for cyfra, waga in zip(cyfry, wagi)) % 11 == cyfry[9]:
        lista_poprawne.append(f"Poprawna cyfra kontrolna w numerze NIP {kogo}.")
    else:
        lista_bledne.append(f"Niepoprawna cyfra kontrolna w numerze NIP {kogo}: {nip_b}.")
    return
```

Dane przedsiębiorstw sprawdzane są dzięki połączeniu z białą listą podatników VAT udostępnianą przez API Ministerstwa Finansów. Oficjalne informacje pobierane zostają według poniższego kodu. Oczyszczony z niepotrzebnych znaków NIP dopisywany jest do strony API MF, z którą *Sygnalista VAT* nawiązuje połączenie, a następnie zapisuje odpowiedź z danymi na aktualny dzień w zmiennej *strona*. Nim jednak informacje zostaną wysłane, sprawdzane jest czy połączenie się powiodło, co jest jednoznaczne z statusem 200. Status połączenia weryfikowany jest z użyciem polecenia *status_code*, jeżeli zwracana wartość jest inna np. 500, 400 lub 429 to użytkownik zostaje poinformowany o rodzaju błędu połączenia jaki powstał. W

³⁰ https://pl.wikibooks.org/wiki/Kody_%C5%BAr%C3%B3d%C5%82owe/Implementacja_NIP [Dostęp 02.06.2024]

innym wypadku, gdy udaje się uzyskać odpowiedź, jej zawartość zapisywana jest do zmiennej *dane* z poleceniem *.text*, a następnie zapisana jako słownik *dane_sownik*, dzięki funkcji *.loads()* z biblioteki *json*.

funkcja sprawdzająca czy podany nip istnieje, czy jest obecny na białej liście podatników VAT, łączy się poprzez API ze stroną Ministerstwa Finansów i pobiera dane o firmie o danym nr nip o ile takowa istnieje i usługa jest dostępna

```
def check_istnienie_nip(nip_c, kogo):
    nip_c = int(nip_c.replace(" ", "").replace("-", ""))
    api_1 = fr"https://wl-api.mf.gov.pl/api/search/nip/{nip_c}"
    strona = requests.get(api_1, params={"date": str(datetime.date.today())})
    if strona.status_code == 500:
        lista_bledne.append(f"Usługa API jest niedostępna, nie można sprawdzić poprawności "
                           f"danych firmy {kogo} dla wybranej faktury.")
    elif strona.status_code == 400:
        lista_bledne.append(f"W bazie danych, dla wybranej faktury, występuje nieistniejący "
                           f"NIP {kogo}.")
    elif strona.status_code == 429:
        lista_bledne.append(f"Wykorzystano limit sprawdzeń API, nie można sprawdzić "
                           f"poprawności danych firmy {kogo} dla wybranej faktury, "
                           f"wróć jutro.")
    else:
        dane = strona.text
        global dane_sownik
        dane_sownik = json.loads(dane)
        lista_poprawne.append("Usługa API jest dostępna.")
    return
```

Pobrane w ten sposób oficjalne dane przedsiębiorstw można porównać z informacjami w bazie, z której korzysta Sygnalista VAT np. nazwa firmy zgodnie z poniższą funkcją. Ze słownika z danymi wywoływana zostaje nazwa podmiotu a następnie porównana z wartością na fakturze, czego wynikiem jest komentarz o poprawnym lub błędnym elemencie. Jeżeli pobrane dane zawierają istotny błąd tj. NIP nie jest obecny na białej liście lub źle został zapisany przy wyszukiwaniu, to próba odwołania do niego w słowniku kończy się błędem. Ich wystąpienie kontrolowane jest przez konstrukcję obsługi wyjątków *try except*, która w momencie powstania nieprawidłowości *TypeError* lub *ValueError* reaguje i nie dopuszcza do porównania brakującej wartości z elementem faktury a zwraca odpowiedni komentarz do listy błędnych elementów.

funkcja sprawdzająca, czy nazwa firmy w bazie danych przypisana do faktury jest zgodna z informacjami ze strony MF

```
def check_nazwa(nazwa, kogo):
    try:
        global nazwa_api
        nazwa_api = dane_sownik['result']['subject']['name']
```

```

if nazwa.upper() == nazwa_api:
    lista_poprawne.append(f"Zgadza się nazwa {kogo}.")
else:
    lista_bledne.append(f"Dla wybranej faktury nie zgadza się nazwa {kogo}."
                        f"Podana: {nazwa.upper()}Poprawna: {nazwa_api}.")
except TypeError:
    lista_bledne.append(f"Dla wybranej faktury podany NIP {kogo} nie figuruje w rejestrze "
                        f"VAT.")
    pass
except NameError:
    lista_bledne.append(f"Dla wybranej faktury podany NIP {kogo} ma nieprawidłowy "
                        f"format lub cyfrę kontrolną, nie figuruje w rejestrze VAT.")
    pass
return

```

Na podobnej zasadzie weryfikowane są dane banków na podstawie numerów kont, jednak w tym wypadku źródłem prawidłowych wartości jest baza danych pobrana wcześniej, a nie poprzez bezpośrednie połączenie z API w czasie rzeczywistym.

Istotnym etapem prowadzącym do określenia poprawności faktury jest weryfikacja stawek podatku. Ich niepoprawne wartości lub błędne przypisanie do towaru lub usługi może wiązać się z obliczeniem nieprawidłowej wartości brutto oraz kwoty podatku do odprowadzenia. *Sygnalista VAT* bazuje na czterech podstawowych stawkach: 23%, 8%, 5%, 0% oraz zwolnieniu z podatku. Pierwszy krok polega na sprawdzeniu czy stawki podane na fakturze należą do listy uznawanych stawek, poprzez funkcję `.isin()`. Zwraca ona fałsz, jeżeli dla danej pozycji stawka jest nieobecna na liście, co poddawane jest negacji `~`, aby polecenie `any()` (zwracające prawdę, jeżeli, którakolwiek wartość jest prawdą) zweryfikowało czy wystąpił przynajmniej jeden błąd. Wynik zapisywany jest na liście poprawnych lub błędnych elementów, a nieprawidłowość dodatkowo blokuje możliwość wyliczenia kwot podatku i brutto.

dostępne stawki podatku VAT

```
lista_stawki = ["0.23", "0.08", "0.05", "0.00", "zw."]
```

funkcja sprawdzająca, czy stawki VAT przypisane do faktury są poprawne

```

def check_stawki(stawka):
    if any(~stawka.isin(lista_stawki)):
        lista_bledne.append(f"Dla wybranej faktury nie zgadzają się stawki podatku. "
                            f"Dostępne stawki: {lista_stawki}, podane stawki: {list(stawka)}")
        global ilosc_bledow
        ilosc_bledow += 1
    else:
        lista_poprawne.append("Stawki podatku są poprawne.")
    return

```

Drugą częścią kontroli stawek jest weryfikacja ich przypisania do towarów i usług. Każda pozycja na fakturze sprawdzana jest pod kątem braku danych w postaci nazwy towaru lub usługi oraz czy stawka jest uznawana. Następnie pozycje ze stawką 23% weryfikowane są pod względem obecności na liście możliwych towarów lub usług z taką stawką, a pozostałe w słowniku z listami towarów lub usług związanych ze stawkami 8%, 5%, 0% lub zwolnieniem. Każda pozycja opisywana jest komentarzem, który wskazuje, że dana stawka została przypisana do towaru poprawnie/błędnie.

funkcja sprawdzająca, czy stawki VAT przypisane do faktury są poprawne pod względem przypisania ich do towarów/usług

```
def check_przypisanie_stawek(towary, stawki):
    for pozycja in range(0, len(towary)):
        if pd.isna(towary[pozycja]):
            lista_bledne.append(f"Brak nazwy dla towaru nr {pozycja + 1} na fakturze.")
        elif stawki[pozycja] not in lista_stawki:
            lista_bledne.append(f"Błędna stawka dla towaru nr {pozycja + 1} na fakturze: "
                                f"{str(int(float(stawki[pozycja])*100))+('%' if stawki[pozycja] != 'zw.' else stawki[pozycja])} dla towaru/usługi: {towary[pozycja]}")
        else:
            if stawki[pozycja] != "0.23":
                if towary[pozycja] in stawki_towarow[stawki[pozycja]]:
                    lista_poprawne.append(f"Poprawne przypisanie stawki "
                                           f"{str(int(float(stawki[pozycja])*100))+('%' if stawki[pozycja] != 'zw.' else stawki[pozycja])} "
                                           f"do towaru/usługi: {towary[pozycja]}")
                else:
                    lista_bledne.append(f"Błędne przypisanie stawki "
                                           f"{str(int(float(stawki[pozycja])*100))+('%' if stawki[pozycja] != 'zw.' else stawki[pozycja])} "
                                           f"do towaru/usługi: {towary[pozycja]}")
            else:
                if towary[pozycja] in towary_nie_23:
                    lista_bledne.append(f"Do towaru, dla którego stawka podatku wynosi 23% "
                                           f"przypisano błędną stawkę: "
                                           f"{str(int(float(stawki[pozycja])*100))+('%' if stawki[pozycja] != 'zw.' else stawki[pozycja])}")
                else:
                    lista_poprawne.append(f"Poprawne przypisanie stawki 23% do towaru/usługi: "
                                           f"{towary[pozycja]}")
```

Zgodnie z ustawą faktury zawierać mogą dopiski, związane z niestandardowym sposobem wystawienia i/lub opłacenia dokumentu np. odwrotne obciążenie, samofakturowanie czy podzielona płatność, której weryfikowanie przedstawiono poniżej. Jako argumenty podaje się wartość łącznej kwoty netto, by sprawdzić, czy jest ona mniejsza niż 15 tys. zł; pozycje na fakturze by skontrolować, czy którakolwiek z nich jest produktem wrażliwym oraz wartość

tak/nie zapisaną w kolumnie informującej o zapisie „mechanizm podzielonej płatności” na fakturze. Jeżeli spełnione są warunki konieczne dodania takiego tekstu a jest on nieobecny lub gdy nie zostaną spełnione a dopisek będzie na fakturze, to dodany zostanie komentarz o błędzie lub w przeciwnym wypadku o poprawności.

funkcja sprawdzająca, czy na faktura podlega mechanizmowi podzielonej płatności (zawiera towary wrażliwe o wartości co najmniej 15 000 PLN netto)

```
def check_podzielona(netto, produkty, podzielona):
    if (netto >= 15000) and (any(produkty.isin(towary_podzielona))):
        if podzielona == "tak":
            lista_poprawne.append("Do wybranej faktury poprawnie zastosowano mechanizm podzielonej płatności.")
        else:
            lista_bledne.append("Do wybranej faktury należy zastosować mechanizm podzielonej płatności, dane wskazują na jego brak.")
    else:
        if podzielona == "nie":
            lista_poprawne.append("Do wybranej faktury poprawnie nie zastosowano mechanizmu podzielonej płatności.")
        else:
            lista_bledne.append("Do wybranej faktury nie należy stosować mechanizmu podzielonej płatności, dane wskazują na jego występowanie.")
```

Wśród pozostałych funkcjonalności, o których warto wspomnieć są warunki stosowalności niektórych funkcji do weryfikacji elementów np. jeżeli nabywca jest osobą fizyczną a nie prawną, to program nie podejmie próby sprawdzania danych przedsiębiorstwa, również tak samo zadzieje się w przypadku danych o banku, gdy płatność za fakturę dokonana zostanie gotówką a nie przelewem. Istotnym etapem jest również zmiana kwoty do zapłaty na tekst, który polega na drobiazgowym przejściu przez liczbę od lewej strony do prawej by wyszukać kolejne cyfry w słowniku oraz dopisać do nich np. tysiące w zależności od potrzeb gramatycznych oraz grosze w formie „grosze/100”.

4.3. Interfejs graficzny

Nieodzowną częścią *Sygnalisty VAT* zapewniającą komfort jego użytkowania jest interfejs graficzny. Jego bazą jest klasa, w której skład wchodzi parametry widocznych okien i widżetów, które nadają mu charakterystyczny wygląd. Określone są wymiary interfejsu, kolor tła, logo aplikacji, czcionki, widżety wraz z ich wielkością, położeniem, kolorami, wypełnieniem w postaci tekstu, nazwami. Wszystkie widoczne elementy dodawane są do klasy poprzez odwołanie *self*.

W poniższym kodzie zauważyć można kilka przykładowych poleceń stojących za interfejsem. Traktowany jest on jako klasa z przypisaną metodą, która dodaje do niego

poszczególne zawartości. Na wstępie tworzone jest okno oraz nadawany jest mu wygląd utworzony w programie Qt Designer (zob. 2.4.1.), zapisany jako osobny plik i pobierany na początku jak biblioteka. Dalej rozpoczyna się proces, który pozwoli użytkownikowi działać z aplikacją, samoistnie wywoływany zostaje ekran początkowy. Następnie pojawiają się akcje, które wykonywane są wraz z aktywnością użytkownika. Podawane są też obiekty jakie mogą zostać utworzone na danym etapie np. lista przedsiębiorstw w bazie danych wprowadzona do widżetu *lista_firm*, z którego wybierać można przedsiębiorstwo, jakiego fakturę chce się weryfikować. Na ekranie pojawić może się kilka interakcyjnych widżetów np. przyciski „tak” i „nie” na oknie aktualizacji banków albo „dalej” i strzałka powrotu na kolejnych ekranach.

```
class Sygnalista(object):
    def __init__(self):
        self.main_win = QtWidgets.QMainWindow() # dodanie okna
        self.ui = Ui_mainwin() # nadanie oknu wyglądu pobranemu na początku
        self.ui.setupUi(self.main_win)

        self.ui.okna_zmienne.setCurrentWidget(self.ui.witaj) # przedstawienie użytkownikowi
        powitalnego ekranu programu

        self.ui.przycisk_start.clicked.connect(self.do_banki) # przejście do ekranu z aktualizacją
        bazy danych banków, gdy użytkownik kliknie przycisk start

        lista_firm_str = baza_danych["Przedsiębiorstwo"].unique() # lista dostępnych firm z
        bazy, dla których użytkownik może sprawdzić faktury

        self.ui.lista_firm.addItems(lista_firm_str) # wprowadzenie firm na widget listy w
        interfejsie

        self.ui.przycisk_tak.clicked.connect(self.do_firmy_t) # przejście do ekranu z wyborem
        firm z pobraniem aktualizacji bazy danych banków, gdy użytkownik kliknie przycisk tak na
        ekranie z aktualizacją bazy

        self.ui.przycisk_nie.clicked.connect(self.do_firmy_n) # przejście do ekranu z wyborem
        firm bez pobrania aktualizacji bazy danych banków, gdy użytkownik kliknie przycisk nie na
        ekranie z aktualizacją bazy

        self.ui.przycisk_dalej_firmy.clicked.connect(self.do_faktury) # przejście do ekranu z
        wyborem faktur, gdy użytkownik wybierze firmę i kliknie przycisk dalej

        self.ui.przycisk_powrot_faktury.clicked.connect(self.do_firmy_n) # powrót do ekranu z
        wyborem firm, gdy użytkownik kliknie przycisk powrotu na ekranie z wyborem faktur
```

Każda interakcja z widżetami, które na to pozwalają skutkować może uruchomieniem akcji przypisanej do nich. Kliknięcie przycisku „start” skutkuje uruchomieniem funkcji *do_banki*, która zmienia aktualnie wyświetlany ekran na okno z komentarzem o aktualizacji

bazy danych banków. Poza przejściem do kolejnego widoku możliwe jest również uruchomienie poleceń w tle np. przy kliknięciu „tak” na ekranie z aktualizacją informacji o bankach rozpocznie się proces aktywowany przekazaniem wartości „T” do funkcji się tym zajmującej.

```
# funkcja przejścia do ekranu z aktualizacją bazy danych banków
```

```
def do_banki(self):  
    self.ui.okna_zmienne.setCurrentWidget(self.ui.bd_bankow)
```

```
# funkcja przejścia do ekranu z listą firm do wyboru po aktualizacji bazy danych banków
```

```
def do_firmy_t(self):  
    aktualizacja_banki("T")  
    self.ui.okna_zmienne.setCurrentWidget(self.ui.wybor_firma)
```

Zachodzące w tle działania mogą być bardziej skomplikowane, co obrazuje poniższy przykład. Gdy użytkownik wybierze firmę, której fakturę chce weryfikować i kliknie przycisk dalej następuje utworzenie nowej bazy danych składającej się wyłącznie z rekordów, dla których nazwa przedsiębiorstwa jest taka sama jak wybrana na liście. Dodatkowo przygotowywany jest kolejny ekran – określona zostaje lista dostępnych faktur, jakie użytkownik może wybrać i wprowadzone zostają one na widżet, który został chwilę wcześniej oczyszczony z ewentualnych poprzednich działań (co może się zdarzyć, jeżeli użytkownik będzie próbował weryfikować kilka faktur podczas jednej sesji). W końcu następuje przejście do ekranu z wyborem faktury. Jeżeli program wykryje, że użytkownik nie zaznaczył żadnej firmy, to wywołane zostaje okno informujące o błędzie i informujące co trzeba zrobić (wygląd okna błędu jako klasa powstaje w kodzie *Sygnalisty VAT* i wywoływany jest nazwą *Ui_okn_blad()*).

```
# funkcja przejścia do ekranu z listą faktur
```

```
def do_faktury(self):  
    if self.ui.lista_firm.currentItem() is not None: # gdy wybrana zostaje firma  
        wybrana_firma = self.ui.lista_firm.currentItem().text() # pobierana jest jej nazwa  
        global nowa_baza0  
        nowa_baza0 = baza_danych[baza_danych.Przedsiębiorstwo == wybrana_firma] # i  
        filtrowana jest baza danych by zawierała wyłącznie faktury przypisane do firmy  
        lista_faktur_str = nowa_baza0["Nr_faktury"].unique() # powstaje lista faktur  
        self.ui.lista_faktur.clear() # która pokazywana jest na widżecie (który wcześniej został  
        oczyszczony)  
        self.ui.lista_faktur.addItem(lista_faktur_str)  
        self.ui.okna_zmienne.setCurrentWidget(self.ui.wybor_faktura) # po czym następuje  
        przejście do ekranu z listą faktur  
    else:  
        self.okno_blad = QtWidgets.QMainWindow() # jeżeli nie wybrana zostaje żadna  
        faktura pojawia się okno z błędem
```

```
self.ui_blad = Ui_okno_blad()
self.ui_blad.setupui(self.okno_blad)
self.okno_blad.show()
```

Każda kolejna interakcja z programem powoduje, że w tle zachodzą procesy pozwalające na bieżące reagowanie na aktywność użytkownika. Najważniejsze z nich znaleźć można na schemacie w załączniku nr 3, który pokazuje działania programu od strony kodu. Po uruchomieniu programu następuje przygotowanie wszystkich podstawowych narzędzi, dzięki którym funkcjonować będzie *Sygnalista VAT*. Następuje import bibliotek i kodu interfejsu, wczytywane są dane faktur, listy stawek, dane banków oraz budowane są bazy, definiowane zostają funkcje do weryfikacji elementów oraz aktualizacji informacji o bankach oraz opracowana zostaje klasa okna błędu. Po tym etapie użytkownik widzi ekran powitalny. Gdy przejdzie do okna z opcją aktualizacji bazy danych banków, poprzez wybór „tak” może uruchomić funkcję, która dokona próby pobrania danych, ich przetworzenia oraz zapisania w formie bazy na serwerze SQL. Po tym, jak i po wyborze „nie” następuje wyznaczenie i wprowadzenie na widżet listy dostępnych przedsiębiorstw do wyboru i przejście do ekranu wyboru firmy. Dalej tworzony jest wycinek głównej bazy na podstawie nazwy wybranego przedsiębiorstwa oraz wyznaczana i wprowadzana na widżet jest lista dostępnych faktur do sprawdzenia. Tutaj możliwy jest powrót do wyboru przedsiębiorstwa, który zapełni ich listę od nowa. Wybranie faktury w kolejnym oknie skutkować będzie etapem weryfikacji, w którym oczyszczone zostaną listy poprawnych i błędnych elementów oraz wyzerowana zmienna *ilosc_bledow*, przeprowadzone zostanie sprawdzanie, dokonane zostaną obliczenia i agregacja oraz przygotowana zostanie kwota słownie, w końcu oczyszczone zostaną i uzupełnione widżety w wynikami. Ponownie użytkownik może powrócić do poprzedniego ekranu, co skutkuje przedstawieniem dostępnych faktur do sprawdzenia oraz nadpisaniem wyciętego fragmentu bazy danych. Finalnie użytkownik może zakończyć pracę z programem poprzez jego zamknięcie, co możliwe jest także na każdym etapie.

Powyżej przedstawione funkcjonalności są jedynie ułamkiem procesów jakie zachodzą podczas użytkowania aplikacji. Poprzez otwarcie jej w IDLE możliwe jest prześledzenie całego algorytmu krok po kroku, wspomagając się dostępnymi komentarzami autora. Zaznaczyć należy, że na moment pisania pracy wersja aplikacji uznana jest za ostateczną, jednak nie ogranicza to możliwości jej aktualizacji i rozwoju w przyszłości.

4.4. Rozwój programu

Sygnalista VAT jest programem, który bazuje na przepisach prawa w szczególności ustawy o podatku od towarów i usług oraz oprogramowaniu takim jak język programowania

python oraz zapytań *SQL*. Obejmuje rozwiązania dostępne na rynku, jak weryfikowanie poprawności faktur VAT pod wieloma względami np. dane kontrahenta w oparciu o API chociażby GUS lub REGON, obliczane kwoty podatku i brutto czy kontrolowanie formatów. Daje jednak dodatkowe możliwości kontroli faktur takie jak przypisanie stawek do towarów lub usług, dane banków, specjalne rodzaje faktur, rozróżnienie na osobę prawną i fizyczną oraz sposób płatności gotówką lub przelewem. Oferuje również połączenie z serwerem *SQL*, własne listy towarów i usług jakie można przypisać do stawek, przekształcanie kwoty do zapłaty na tekst, a przede wszystkim wskazywanie jakie elementy są błędne i wymagają uwagi użytkownika.

Algorytm stojący za *Sygnalistą VAT* nie jest idealny, znaleźć można miejsce na poprawki i optymalizację. Wiele procesów np. przeprowadzanych w pętlach lub niektórymi poleceniami można wykonać innymi funkcjami, dostępnymi w innych bibliotekach lub metodami nieznanymi autorowi na dzień powstania przedstawionego programu. Wygląd interfejsu czy zaproponowane procesy mogą również być kwestią sporną, jeżeli chodzi o ich jakość czy istotność. Użytkownik lub inny programista może mieć inne spojrzenie na temat i uznać, że zaprezentowane rozwiązania można zmienić, poprawić lub doszlifować. Są to na pewno zagadnienia, które będzie można rewidować i rozwijać.

Polem do kontynuacji pracy nad programem mogą być chociażby kolejne elementy jakie znaleźć się muszą/mogą na dokumentach np. podstawy prawne zwolnień od podatku, daty płatności, słowa kluczowe „oryginał”/„kopia” i wiele innych. Program może objąć swoim działaniem także inne faktury jakie proponuje ustawa np. korygujące (przy których warunek nieujemności cen zostałby podważony), pro forma czy zaliczkowe. Mógłby uwzględniać zmienność przepisów w kontekście zróżnicowania stawek w czasie (np. podniesienie podstawowej stawki z 22% na 23% w 2011 r. zgodnie z art. 146ef ust. 1 ustawy lub 0% na żywność w ramach tarczy antyinflacyjnej od 02.2022 do 03.2024 r. według rozporządzenia Ministra Finansów³¹). Ulepszyć można także weryfikacje dat (do rozpoznawania dni roboczych i świąt, by wskazać, czy faktura mogła być wtedy wystawiona), uwzględnić większą liczbę formatów (numerów identyfikacyjnych faktur, NIP, kont bankowych) oraz sprawdzać, czy dany numer konta istnieje w bankach poprzez połączenia z ich aplikacjami.

Wdrożenie *Sygnalisty VAT* do praktycznego użytkowania przez przedsiębiorców mogłoby być poważnym wyzwaniem. Konieczne byłoby przygotowanie programu na różne możliwości połączenia ze źródłami danych np. dokumenty przechowywane na serwerach z

³¹ <https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20220002495> [Dostęp 04.06.2024]

ustrukturyzowanymi danymi lub wprowadzenie możliwości odczytywania elementów na podstawie obrazów, plików tekstowych, zdjęć czy też skanów. Dostępne są dwie ścieżki rozwoju w tym kierunku – pierwsza, na której należałoby dodać opcję wprowadzania dokumentów by program stał się samodzielnym oprogramowaniem do pełnej obsługi faktur i finansów oraz druga, gdzie mógłby pozostać dodatkiem do dostępnych już ofert, które wspierałby swoimi zaletami. Bazowanie na aplikacjach pozwalających wystawianie i weryfikowanie dokumentów może stać się niezwykle istotne przy okazji wdrożeniu Krajowego Systemu e-Faktur.³²

³² <https://ksef.podatki.gov.pl/informacje-o-ksef/> [Dostęp 04.06.2024]

Bibliografia

Baranowski P., Doryń W. (2020) *Przetwarzanie danych i uczenie maszynowe w języku Python. Aplikacje w ekonomii i zarządzaniu*. Olsztyn: Instytut Badań Gospodarczych, fragmenty, <http://economic-research.pl/Books/index.php/eep/catalog/book/60> [Dostęp 10.03.2024]

Beaulieu A. (2009) *Learning SQL, Second Edition* Sebastopol: O'Reilly Media, s. 1-14, https://www.r-5.org/files/books/computers/languages/sql/mysql/Alan_Beaulieu-Learning_SQL-EN.pdf [Dostęp 18.03.2024]

Cambridge University Press & Assessment, *API* w Cambridge Dictionary, <https://dictionary.cambridge.org/dictionary/english/api?q=API> [Dostęp 12.03.2024]

Cambridge University Press & Assessment, *web scraping* w Cambridge Dictionary, <https://dictionary.cambridge.org/dictionary/english/web-scraping> [Dostęp 12.03.2024]

Caryk A. (2019) Czy podpis na fakturze jest wymagany? Zobacz 7 rzeczy, których faktura mieć nie musi, aby była ważna. Pobrane z: <https://www.infakt.pl/blog/elementy-faktury-vat/> [Dostęp 17.11.2023]

Kumar S. (2023) *ABC Programming Language*. Pobrane z: <https://www.linkedin.com/pulse/abc-programming-language-santosh-kumar/> [Dostęp 17.02.2024]

Paramanick S. (2024) *History of Python*, Pobrane z: <https://www.geeksforgeeks.org/history-of-python/> [Dostęp 17.02.2024]

Rossum G., Drake F.L. (2003) *An Introduction to Python*. Bristol: Network Theory Ltd. s. 3-6, 101 <http://atk.fam.free.fr/fichiers/stage/Python/JF/site/pytut.pdf> [Dostęp 17.02.2024]

Severance C. (2016) *Python dla wszystkich. Odkrywanie danych z Python 3*. tłum. A. Wójtowicz, University of Michigan, fragmenty <https://open.umn.edu/opentextbooks/textbooks/336> [Dostęp 10.03.2024]

Srinath K. R. (2017) *Python – The Fastest Growing Programming Language*. w: *International Research Journal of Engineering and Technology (IRJET) Volume: 04 Issue: 12* <https://www.irjet.net/archives/V4/i12/IRJET-V4I1266.pdf> [Dostęp 25.02.2024]

Ustawa z dnia 11 marca 2004 r. o podatku od towarów i usług (Dziennik Ustaw 2004 r., nr 54, poz. 535 ze zm.) <https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20040540535> [Dostęp 11.11.2023]

Dyrektywa Parlamentu Europejskiego i Rady (UE) 2019/1937 z dnia 23 października 2019 r. w sprawie ochrony osób zgłaszających naruszenia prawa Unii. (Dz.U.UE.L.2019.305.17) <https://sip.lex.pl/akty-prawne/dzienniki-UE/dyrektywa-2019-1937-w-sprawie-ochrony-osob-zglaszajacych-naruszenia-prawa-69247962> [Dostęp 28.03.2024]

Rozporządzenie Ministra Finansów z dnia 2 grudnia 2022 r. w sprawie obniżonych stawek podatku od towarów i usług w roku 2023 (Dziennik Ustaw 2022 r., poz. 2495) <https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20220002495> [Dostęp 04.06.2024]

Obwieszczenie Marszałka Sejmu RP z dnia 7 lipca 2023 r. w sprawie ogłoszenia jednolitego tekstu ustawy o podatku od towarów i usług (Dziennik Ustaw 2023 r., poz.1570) <https://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20230001570> [Dostęp 11.11.2023]

Netografia

Comarch ERP Optima – strona internetowa z prezentacją programu, <https://www.comarch.pl/erp/comarch-optima/ksiegowosc/> [Dostęp 30.12.2023]

Comarch ERP XL – strona internetowa z prezentacją programu, <https://www.comarch.pl/erp/xl/finanse-i-ksiegowosc/> [Dostęp 30.12.2023]

Comarch ERP XT – strona internetowa z prezentacją programu, <https://www.erpxt.pl/funkcje/faktura-online/> [Dostęp 30.12.2023]

Comarch SA – strona internetowa firmy, <https://www.comarch.pl/o-firmie/> [Dostęp 30.12.2023]

Faktura.pl – strona internetowa z prezentacją programu, <https://faktura.pl/funkcje/> [Dostęp 14.05.2024]

inFakt – strona internetowa z prezentacją programu do faktur, <https://www.infakt.pl/program-do-faktur/> [Dostęp 30.12.2023]

inFakt – strona internetowa z prezentacją programu do fakturowania, <https://www.infakt.pl/program-do-fakturowania/> [Dostęp 30.12.2023]

Systim – strona internetowa z prezentacją programu, <https://www.systim.pl/funkcjonalnosci-systim> [Dostęp 30.12.2023]

Taxon – strona internetowa z prezentacją programu, <https://www.taxoninvoice.com/pl/wystawianie-faktur-online/> [Dostęp 14.05.2024]

API Ministerstwa Finansów – adres aplikacji MF zawierającej białą listę podatników VAT, <https://wl-api.mf.gov.pl/#nip?date> [Dostęp 11.03.2024]

Datetime – dokumentacja biblioteki, <https://docs.python.org/3/library/datetime.html> [Dostęp 11.03.2024]

Internetowa Baza Ewidencji Numerów Instytucji Finansowych Narodowego Banku Polskiego – strona internetowa NBP z plikami danych z ewidencji, <https://ewib.nbp.pl/faces/pages/daneDoPobrania.xhtml> [Dostęp 11.03.2024]

Json – dokumentacja biblioteki, <https://docs.python.org/3/library/json.html> [Dostęp 11.03.2024]

MS SQL Server – strona internetowa oprogramowania, <https://www.microsoft.com/pl-pl/sql-server/sql-server-downloads> [Dostęp 20.04.2024]

Numpy – dokumentacja biblioteki, <https://numpy.org/doc/1.26/user/index.html#user> [Dostęp 10.03.2024]

ODBC Driver – strona internetowa biblioteki DLL, <https://learn.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver16> [Dostęp 20.04.2024]

Os – dokumentacja biblioteki, <https://docs.python.org/3/library/os.html> [Dostęp 11.03.2024]

Pandas – dokumentacja biblioteki, https://pandas.pydata.org/docs/user_guide/index.html#user-guide [Dostęp 10.03.2024]

Python – dokumentacja, <https://docs.python.org/3/tutorial/index.html> [Dostęp 09.03.2024]

Python – strona internetowa języka programowania, <https://www.python.org/> [Dostęp 20.04.2024]

PyQt5 – dokumentacja biblioteki, <https://www.riverbankcomputing.com/static/Docs/PyQt5/> [Dostęp 28.03.2024]

Qt Designer – strona internetowa i dokumentacja oprogramowania, <https://doc.qt.io/qt-6/qtdesigner-manual.html> [Dostęp 28.03.2024]

Re – dokumentacja biblioteki, <https://docs.python.org/3/library/re.html> [Dostęp 11.03.2024]

Requests – dokumentacja biblioteki, <https://requests.readthedocs.io/en/latest/> [Dostęp 11.03.2024]

SQLAlchemy – dokumentacja biblioteki, <https://docs.sqlalchemy.org/en/20/#> [Dostęp 28.03.2024]

SQL Server Management Studio – strona internetowa oprogramowania, <https://learn.microsoft.com/pl-pl/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16> [Dostęp 28.03.2024]

Xmltodict – dokumentacja biblioteki, <https://pypi.org/project/xmltodict/> [Dostęp 11.03.2024]

Canva – https://www.canva.com/pl_pl/ [Dostęp 14.05.2024]

Gofin.pl – wzór faktury VAT, <https://druki.gofin.pl/faktura-3-pozycje,wzor,923,171.html> [Dostęp 14.05.2024]

Google Trends, <https://trends.google.com/home?hl=pl> [Dostęp 17.07.2024]

Krajowy System e-Faktur – strona internetowa Ministerstwa Finansów, <https://ksef.podatki.gov.pl/informacje-o-ksef/> [Dostęp 04.06.2024]

Wikibooks – Implementacja algorytmu obliczania cyfry kontrolnej w numerze NIP,
https://pl.wikibooks.org/wiki/Kody_%C5%BAr%C3%B3d%C5%82owe/Implementacja_NIP
[Dostęp 02.06.2024]

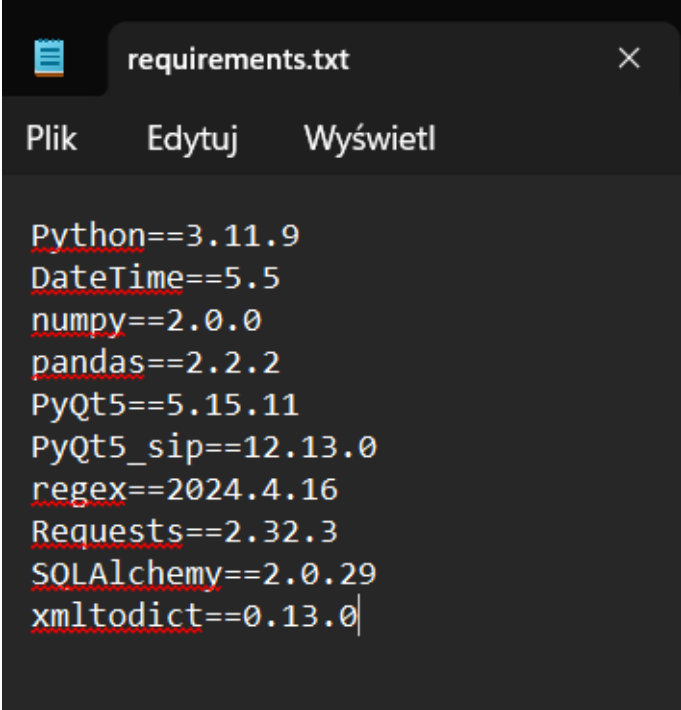
Spis rysunków

Rysunek 1. Relacje tworzące bazę danych wykorzystywaną przez Sygnałistę VAT.....	19
Rysunek 2. Oficjalna strona internetowa języka Python	22
Rysunek 3. Lista wydań Pythona	23
Rysunek 4. Lista wersji dla wybranego wydania Pythona	23
Rysunek 5. Instalator Pythona	24
Rysunek 6. Folder z zainstalowanymi plikami Pythona.....	24
Rysunek 7. Wyniki wyszukiwania frazy "python" w wyszukiwarce systemowej po instalacji oprogramowania	25
Rysunek 8. Wyniki wyszukiwania frazy "wiersz polecenia" w wyszukiwarce systemowej	25
Rysunek 9. Wiersz polecenia, w którym instalowane są moduły Pythona.....	26
Rysunek 10. Strona internetowa firmy Microsoft, gdzie pobrać MS SQL Server	26
Rysunek 11. Instalator oprogramowania MS SQL Server.....	27
Rysunek 12. Ekran końcowy instalatora z opcją instalacji SQL Server Management Studio .	27
Rysunek 13. Strona internetowa firmy Microsoft, na której można pobrać SSMS.....	28
Rysunek 14. Folder z plikami SSMS.....	28
Rysunek 15. Wyniki wyszukiwania frazy "SQL Server Management Studio" w wyszukiwarce systemowej	29
Rysunek 16. Folder z plikami, z których korzysta Sygnałista VAT.....	30
Rysunek 17. Ekran połączenia z serwerem wyświetlany przy uruchomieniu SSMS.....	30
Rysunek 18. Fragment skryptu SQL ze wskazówkami	31
Rysunek 19. Opcje w zakładce Query SSMS.....	31
Rysunek 20. Ekran startowy Sygnałisty VAT	32
Rysunek 21. Ekran z opcją aktualizacji bazy danych banków	33
Rysunek 22. Ekran z wyborem przedsiębiorstwa	34
Rysunek 23. Okno z komunikatem błędu.....	34
Rysunek 24. Ekran z listami poprawnych i błędnych elementów faktur.....	35
Rysunek 25. Okno z wydrukiem faktury	36
Rysunek 26. Schemat rozpoznawania typu zagnieżdżenia danych o bankach.....	38
Rysunek 27. Schemat ekstrakcji danych ze słowników/list do ramki danych według różnych typów zagnieżdżeń.....	40

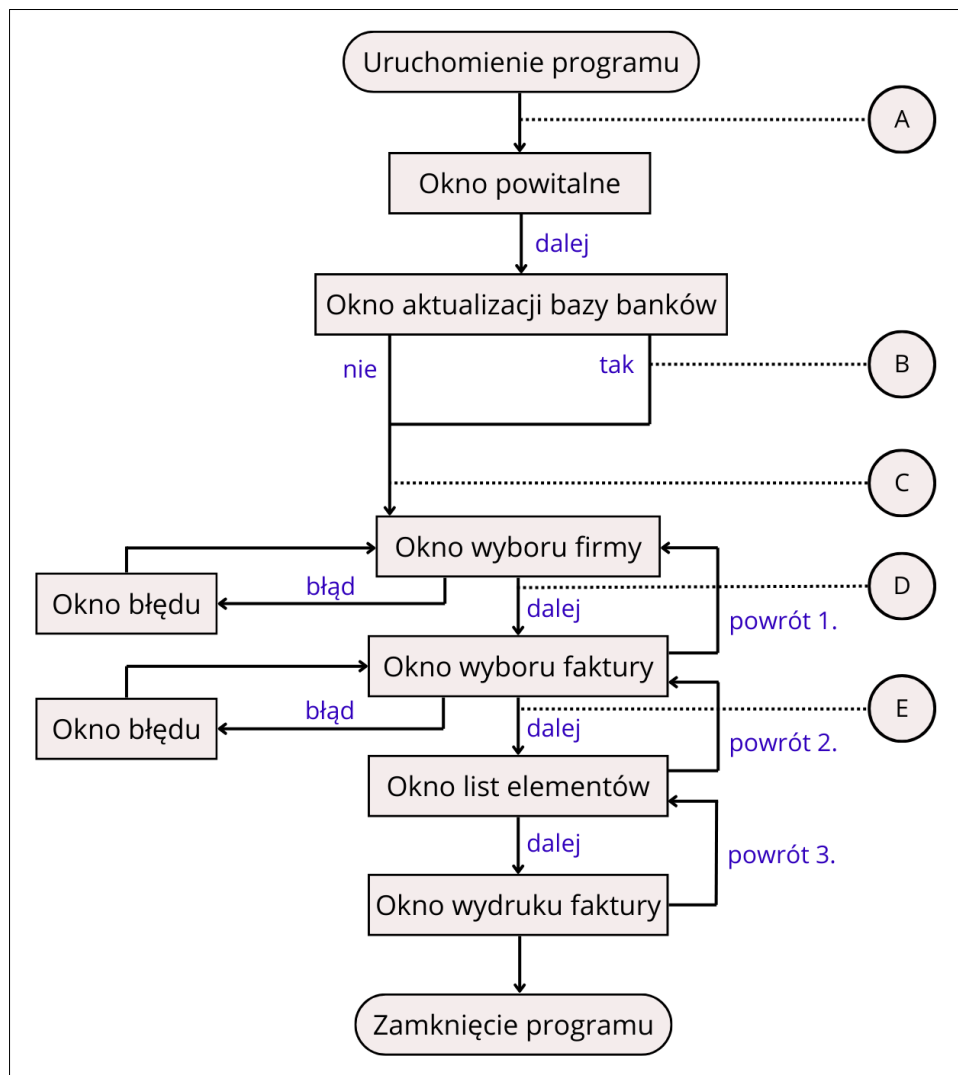
Załącznik nr 1: Przykładowy wzór faktury VAT

Źródło: Gofin.pl, <https://druki.gofin.pl/faktura-3-pozycje,wzor,923,171.html> [Dostęp 14.05.2024]

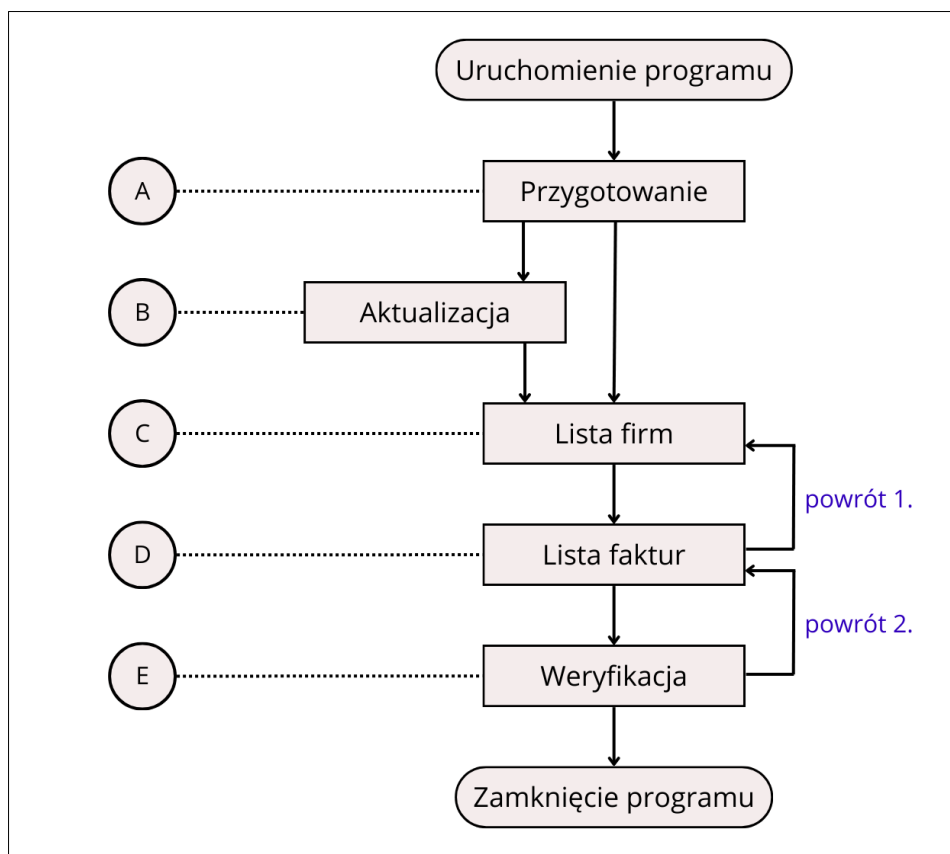
Załącznik nr 2: Wymagane wersje Pythona oraz bibliotek do prawidłowego funkcjonowania programu Sygnalista VAT



```
Python==3.11.9
DateTime==5.5
numpy==2.0.0
pandas==2.2.2
PyQt5==5.15.11
PyQt5_sip==12.13.0
regex==2024.4.16
Requests==2.32.3
SQLAlchemy==2.0.29
xmltodict==0.13.0|
```



Źródło: opracowanie własne w Canvie



Źródło: opracowanie własne w Canvie