WYŻSZA SZKOŁA TECHNOLOGII INFORMATYCZNYCH W KATOWICACH
WYDZIAŁ INFORMATYKI
KIERUNEK: INFORMATYKA

Nowak Marcin
Nr Albumu 08255
Studia niestacjonarne

Projekt i implementacja aplikacji wspomagającej zarządzanie budżetem domowym

Przedmiot: Projekt Systemu Informatycznego
pod kierunkiem
mgr. Jacek Żywczok
W roku akademickim 2022/23

Katowice 2022

Spis treści

1 *Wprowadzenie do tematyki projektu* **4**

2 *Zamierzony cel projektu* **5**

3 *Wstępne założenia i uwarunkowania* **6**
3.1 *Założenia* 6
3.2 *Uwarunkowania* 6

4 *Założone ograniczenia i możliwości ewaluacji projektu* **7**

5 *Plan pracy* **8**

6 *Wymagania funkcjonalne* **10**

7 *Wymagania niefunkcjonalne* **11**
7.1 *Sprzętowe wymagania niefunkcjonalne* 11
7.2 *Systemowe wymagania niefunkcjonalne* 11
7.3 *Organizacyjne wymagania niefunkcjonalne* 11

8 *Wymagania danych* **12**
8.1 *Baza danych* 12
8.2 *Kod aplikacji* 13

9 *Metody pracy, narzędzia i techniki* **14**
9.1 *Metody pracy* 14
9.2 *Narzędzia* 15
9.3 *Techniki* 15

10 *Opisy metod* **16**
10.1 *Główne klasy projektu* 16
10.2 *Baza danych* 16
10.3 *Logika aplikacji* 23
10.4 *Graficzny interfejs użytkownika* 25
10.5 *Metody projektu* 28
10.6 *Obiekty projektu* 30
10.7 *Struktury projektu* 30
10.8 *Algorytmy projektu* 30

Rozdział 1

Wprowadzenie do tematyki projektu

Finanse są dziedziną nauki ekonomicznej któa zajmuje się rozporządzaniem pieniędzmi [1]. Nauka ta w podobnym zakresie a różnej skali dotyczy państw, przedsiębiorstw jak i zwykłych obywateli - w efekcie jest to dziedzina o stosunkowo prostych podstawach jednak niesamowicie skomplikowana w każdym zakresie w którym można ją zagłębić. Wiedza z zakresu finansów staje się szczególnie przydatna gdy na rynku panuje trudna sytuacja ekonomiczna, w takich warunkach nierzadko decyduje ona o jakości oraz stanie życia poszczególnych osób fizycznych, rentowności przedsiębiorstw czy stabilności państw. W przypadku państw i firma przeważnie budżetem zarządzają dedykowane osoby lub też całe zespoły, posiadające ekspercką wiedzę w tej dziedzinie. Jednak osoby zarządzające budżetem domowym najczęściej dysponują wyłacznie nabytym doświadczeniem, rzadko jeśi wogóle wspomagając się jakimikolwiek narzędziami które ułatwiałyby to zadanie.

Rozdział 2

Zamierzony cel projektu

Celem projektu jest utworzenie modułu analitycznego aplikacji któa ułatwi zarządzanie budżetem domowym dostarczając użytkownikowi narzędzia do analizy wpływów i wydatków, wizualizacji trendów oraz automatycznie kategoryzujące wpływy i wydatki.

Docelowymi odbiorcami aplikacji są użytkownicy domowi, możliwe że w miarę rozwoju w późniejszych fazach projektu także średnie lub małe przedsiębiorstwa. Użytkownik po wprowadzeniu danych uzyska dostęp do możliwie czytelnego obrazu sytuacji finansowej co uwidoczni trendy wydatków i przychodów, pozwoli bardziej świadomie podejmować dalsze decyzje finansowe, planować budżet a także łatwo identyfikować obszary które wymagają usprawnień. W efekcie uwidocznione przez aplikację informacje i wyciągnięte z nich wnioski ułatwią użytkownikowi poprawę sytuacji finansowej swojego domostwa poprzez bardziej efektywne zarządzanie budżetem.

Rozdział 3

Wstępne założenia i uwarunkowania

3.1 *Założenia*

Z uwagi na specyfikę tematyki potencjalnymi odbiorcami aplikacji będą osoby w niewielkim lub średnim stopniu zaznajomione z obsługą komputera. Największy wpływ na projekt i rozwój graficznego interfejsu użytkownika oraz kierunek rozwoju aplikacji będzie miała łatwość obsługi. W efekcie aplikacja powinna być możliwie prosta, mieć przejrzysty, minimalistyczny interfejs dzięki czemu użytkownik nie zostanie przytłoczony mnogością dostępnych funkcji.

Początkowo użytkownik będzie wprowadzał dane do aplikacji samodzielnie poprzez dedykowany interfejs. Aplikacja zadba o jakość danych przyjmując jednak oznaczając i pomijając dane błędne, niepełne lub niepewne które zaprezentuje w dedykowanej zakładce gdzie użytkownik będzie mieć możliwość ich poprawy. Użytkownik będzie w stanie wybrać zestaw predefiniowanych typów i kategorii wydatkó lub utworzyć i edytować własne. Aplikacja udostępni predefiniowane wizualizacje, wliczając możliwość wizualizacji określonego przez użytkownika produktu lub całej kategorii produktów. We wstępnej wersji aplikacji interfejs będzie statyczny, podobnie jak konfiguracja, która przechowywana będzie w pliku.

3.2 *Uwarunkowania*

Jest to projekt edukacyjny którego celem jest dostarczenie minimalnego opłacalnego produktu [6], swego rodzaju prototypu, na tym etapie funkcje dodatkowe zostaną pominięte ze względu na ograniczony czas produkcji, zakres umiejętności technicznych autora. Bliski termin oddania wyklucza bardziej zaawansowane funkcje, a znajomość technologii będzie budowana w trakcie jego rozwoju co wpłynie między innymi na ograniczenia systemowe. Aplikacja będzie także z zasady obsługiwać wyłącznie pojedynczego użytkownika, a dane przez niego wprowadzone będa przechowywane wyłącznie lokalnie. Pominięte zostanie także automatyczne pobieranie danych z interfejsów innych aplikacji lub w formie ekstrakcji danych ze skanowanych dokumentów czy kodów EAN lub QR produktów. Aplikacja nie będzie także udostępniać żadnego rodzaju interfejsu programistycznego (API). Założono że na tym etapie interfejs aplikacji będzie statyczny bez możliwości zmiany przez użytkownika.

Rozdział 4

Założone ograniczenia i możliwości ewaluacji projektu

W miarę możliwości standard danych w aplikacji dopasowany zostanie do wiodącego globalnego standardu danych w obrębie tej samej tematyki. Typy obiektów będzie można grupować na kilku poziomach aby ułatwić użytkownikowi zarządzanie danymi i uprościć wizualizacje. Dla zaawansowanych użytkowników może okazać się przydatna możliwość definiowania i zapisywania własnych wizualizacji i raportów statystycznych - wymagać to będzie jednak implementacji dedykowanego modułu. Kolejnym obecnie pominiętym aspektem jest zabudowanie reguł przeprowadzających dogłębną analizę statystyczną danych które otwierają dalsze możliwości rozwoju oprogramowania. Aplikacja dostarczana będzie użytkownikom w formie spakowanej w archiwum skompilowanej pełnej wersji, jeśli zajdzie taka potrzeba i pojawi się możliwość stworzony zostanie także instalator. Możliwe że w aplikacji utworzony zostanie panel administracyjny prezentujący użytkownikowi dane statystyczne obrazujące ilość, zakres i jakość danych a także sugerujące kolejny krok ich usprawnienia.

Funkcjonalności importu i eksportu danych ze standardowych formatów będzie przydatna dla użytkownika podczas korzystania z projektu, wymaga określenia odpowiedniego formatu i standardu plików co może zająć sporo czasu dlatego zostały uznane za dodatkowe przez co możliwe że nie zostaną wdrożone w początkowej fazie projektu.

Obrane podejście pozwoli na rozwój aplikacji w różnych kiernkach, zależnie od opinii użytkowników. Aplikacja może zmienić model z aplikacji lokalnej która wspiera pojedynczego użytkownika na centralny obsługujących wielu użytkowników zdalnie. Ujednolicony interfejs umożliwia także powstanie potencjalnej wersji mobilnej co otworzy nowe możliwości wprowadzania danych oraz interakcji użytkownika z aplikacją.

Rozdział 5

Plan pracy

W toku prac stworzona zostanie lista zadań do zrealizowania, do określenia ich priorytetu posłuży metoda MoSCoW [4] lub Matryca Eisenhowera [5]. Przewidywany plan pracy nad projektem prezentuje się następująco:

1. Spis założeń w dokumentacji wstępnej
    - Założenia wstępne
    - Spis wymagań każdego typu
    - Przegląd rynku pod kątem dostępnych rozwiązań
    - Określenie metodologii pracy
    - Dokumentacja modelowania
    - Dokumentacja uruchomieniowa projektu
    - Przeprowadzone testy
    - Instrukcja obsługi dla użytkownika
    - Retrospekcja
2. Modelowanie
    - Utworzenie słownika modelowanej domeny
    - Określenie wymaganych kontenerów
    - Określenie wymaganych encji i atrybutów
    - Określenie wymaganych ograniczeń danych
    - Modelowanie powiazań encji
3. Wybór technologii
    - Wspierane systemy i wersje
    - Wybór języka
    - Biblioteki interfejsu użytkownika
    - Sposób przechowywania danych
    - Instalator, aktualizacja i utrzymanie
4. Wstępne wdrożenie
    - Utworzenie bazy danych
    - Utworzenie podstawowych struktur bazy danych - tabele
    - Wypełnienie danymi testowymi
    - Utworzenie złożonych struktur bazy danych - widoki
    - Projekt interfejsu użytkownika
    - Szkielet interfejsu użytkownika

- Połączenie interfejsu z bazą danych
- Projekt podstawowych wizualizacji
- Iteracyjna uzupełnienie interfejsu i bazy o dodatkowe funkcje
- Usprawnienia i refaktoryzacja

5. Testy rozwiązania
    - Utworzenie danych testowych
    - Określenie spodziewanych wyników
    - Porównanie wyników oczekiwanych z otrzymanymi
6. Iteracyjne usprawnienia projektu i uzupełnianie dokumentacji
7. Retrospekcja
    - Przydatność gotowej aplikacji
    - Wady i zalety podejścia
    - Sprawność rozwiązań
    - Sprawność technologii
    - Spis wniosków

Rozdział 6

Wymagania funkcjonalne

Zestawienie funkcji które powinien spełniać program, wraz z informacją któe z nich zostały spełnione. Nagłówki z powodu objętości zostały skrócone, legenda:
PRIO - Priorytet w jednej z kategorii MOSCOW [4]
IMPL - Oznaczenie czy wdrożono funkcjonalność

Funkcjonalność	**PRIO**	**IMPL**	**Opis**
Plik konfiguracji	M	TAK	Osobny plik konfiguracyjny
Dodawanie danych	M	-	Dodawanie danych
Podsumowanie wydatków	M	TAK	Okresowe podsumowanie wydatków
Podsumowanie przychodów	M	TAK	Okresowe podsumowanie przychodów
Statystyki typów	C	TAK	Statystyki wydatków na dany typ produktu
Statystyki produktów	C	TAK	Statystyki wydatków na dany produkt
Bilans okresowy	M	TAK	Okresowy bilnas zysków i strat
Definiowanie produktów	M	TAK	Definiowanie produktów
Definiowanie przychodów	M	TAK	Definiowanie przychodów
Definiowanie typów produktów	M	TAK	Definiowanie typów produktów
Definiowanie typów przychodów	C	NIE	Definiowanie typów przychodów
Panel konfiguracyjny	S	TAK	Osobny panel konfiguracyjny
Rejestr zdarzeń	S	NIE	Logi z działania aplikacji
Dostęp zdalny	C	NIE	Dostęp do zdalnych baz danych
Import danych	C	TAK	Import danych w standardowym formacie
Walidacja danych	C	-	Potwierdzenie jakości danych
Eksport danych	C	-	Eksport danych do standardowego formatu
Instalator	C	NIE	Prosty instalator aplikacji
Trendy	W	NIE	Predykcja trendów wydatkó i wpływów
Porady	W	NIE	Porady dla użytkownika dotyczące usprawnień budżetu
Wiele użytkowników	W	NIE	Wsparcie dla wielu użytkowników jednocześnie
Personalizacja interfejsu	W	TAK	Personalizacja interfejsu użytkownika
Aktualizacje	W	NIE	Automatyczne sprawdzanie wersji i aktualizacja

Tabela 6.1: Wymagania funkcjonalne

Rozdział 7

Wymagania niefunkcjonalne

7.1 *Sprzętowe wymagania niefunkcjonalne*

- Pamięć 50MB dowolnego typu
- Pamięć RAM 4GB (wliczajac system)
- Urządzenia peryferyjne: klawiatura, mysz komputerowa
- Dowolne urządzenie wyświetlające

7.2 *Systemowe wymagania niefunkcjonalne*

System Operacyjny Windows 10.

7.3 *Organizacyjne wymagania niefunkcjonalne*

Projekt aplikacji obejmuje interakcje z pojedynczym użytkownikiem w danym momencie, każdy z użytkowników będzie korzystał z własnej instancji bazy danych aplikacji która bedzie przechowywana w dowolnej dogodnej określonej przez użytkownika pamięci lokalnej określonej w pliku konfiguracyjnym, a także zdalnej jeśli wdrożona zostanie funkcjonalność dostępu zdalnego. Aplikacja wstępnie nie będzie wymagała stałego dostępu do sieci, jednak w przyszłości jej rozwój może zmienić to wymaganie - wymagać wtedy będzie krótkich okresów dostępu do sieci. Dostęp do danych będzie wymagany w krótkich okresach zapisu danych z pamięci podręcznej aplikacji do bazy oraz odpytania bazy o dane. Aplikacja powinna być wykorzystywana na systemach zabezpieczonych przed dostępem osób niepowołanych.

Rozdział 8

Wymagania danych

W tej sekcji spisano wymagania wstępne, jak w każdym projekcie wymagania zmieniały się wraz z jego rozwojem w trakcie modelowania i wdrażania. Aktualny reprezentatywny stan danych w aplikacji z wyróżnieniem wszystkich warstw opisuje rozdział *Opisy metod*.

Użytkownik będzie wprowadzał dane do aplikacji ręcznie lub za pomocą interfejsu importującego dane w formacie CSV (Comma Separated Values) [8]. Dane wprowadzone przez użytkownika po walidacji trafią odpowiedniej tabeli w bazie danych, natomiast te któe walidacji nie przejdą do zbioru tymczasowego w którym użytkownik będzie je mógł poprawić - po przejściu prawidłowo walidacji trafią do zbioru docelowego. Wszystkie dane wprowadzone przez użytkownika będą traktowane jako ciągi znaków a następnie rzutowane na typy odpowiadające docelowym polom w bazie danych do których zostaną przekazane. Wymagania danych w bazie oraz aplikacji podzielono na poszczególne sekcje poniżej.

8.1 *Baza danych*

Aplikacja wymaga sposobu na trwałe składowanie danych wprowadzonych przez użytkownika. W tym celu wraz z aplikacją każdy użytkownik otrzyma dedykowaną lokalną instancję bazy danych SQLite [9] do własnego użytku. Za wyborem tej technologii przemawiają niewielkie wymagania przestrzeni pamięci trwałej jakie zajmuje oraz w miarę kompletne wsparcie standardowego języka zapytań SQL [10].

Baza powinna zawierać tabele do przechowywania danych o przychodach, rachunkach i wydatkach oraz wszelkie dane pomocnicze wspomagające normalizację danych. Powinna zawierać także widoki wspomagające pracę z danymi obliczające w miarę możliwości automatycznie dane analityczne wykorzystywane później w aplikacji. Aby zachować spójność danych i raportów dane wprowadzane do bazy powinny być odpowiednio walidowane, jednak aby zadbać o ich kompletność użytkownik powinien mieć możliwość świadomie wprowadzić dane które wymagają poprawy do późniejszej obróbki.

Szczegóły implementacji wraz z kodem opisane zostaną w sekcji ***Baza danych***

8.2 *Kod aplikacji*

Python jest językiem elastycznym, dzięki czemu przyjmuje dane w dowolnej formie bez definiowania typu, co pozwala odroczyć projekt klas do momentu implementacji a nawet refactoringu programu. W języku Python [11] domyślnie wszystkie zmienne są typu Any co pozwala definiować typy zależnie od potrzeb w trakcie tworzenia programu. Wszystkie wczytywane z plików dane domyślnie przyjmują typ string lub są traktowane jako pojedyncze bajty [14]. W efekcie dane wprowadzane przez użytkownika przyjmą formę ciągów znaków klasy string, po czym w programie zostaną przekazane do konstruktorów klas gdzie zostaną rzutowane na odpowiednie do zadań typy i zapisane w polach.

Aby ułatwić rozwój aplikacji program powinien zapisywać i przyjmować z pliku dane konfiguracyjne w formie list ciągów znaków w formacie "opcja" : "wartość" które utworzą obiekt słownika, lub w wariancie bardziej zaawansowanym aplikacja może przyjmować konfigurację w postaci pliku JSON [13]. Rozwiązanie to jest na tyle elastycznie że pozwoli dynamicznie definiować dodatkowe konfiguracji w trakcie powstawania projektu - aplikacja zaczyta konfigurację z pliku i na jej podstawie utworzy słownik, dzięki czemu dodanie nowego parametru sprowadzi się do dopisania linijki tektu do pliku.

Aplikacja powinna udostępniać użytkownikowi intuicyjne wizualizacje danych dostępne przez interfejs użytkownika. Ponieważ jest to projekt edukacyjny aplikacja zostanie zaimplementowana w języku Python [11] którego autor chce się nauczyć. Do implementacji interfejsu użytkownika posłuży biblioteka PySimpleGUI [12] która pozwala tworzyć prosty i responsywny, niezależny od platformy interfejs o ograniczonych możliwościach. Jest to adapter (tak zwany wrapper) który łączy kilka popularnych bibliotek służących do tworzenia interfejsu, udostępnia spójny interfejs dzięki czemu warstwę prezentacji można budować wykorzystując tablice złożone z udostępnianych jako obiekty widżetów. W fazie prototypu interfejsu nie będzie można konfigurować z poziomu aplikacji, natomiast niewykluczone że użytkownik będzie w stanie zmienić podstawowe ustawienia edytując dostarczony plik konfiguracyjny. Szczegóły implementacji interfejsu opisano w sekcji ***Graficzny interfejs użytkownika***.

W samym kodzie aplikacji podstawymi strukturami wizualizacji będą kolekcje złożone z dat i wartości typu double. Wizualizacje będą przechowywane w słownikach, które na podstawie klucza w formacie string służącego także jako nazwa wizualizacji zwrócą wartość w formacie string którą będzie zapytanie do konkretnej tabeli w bazie danych. Szczegóły implementacji kodu opisano w sekcji ***Logika aplikacji***.

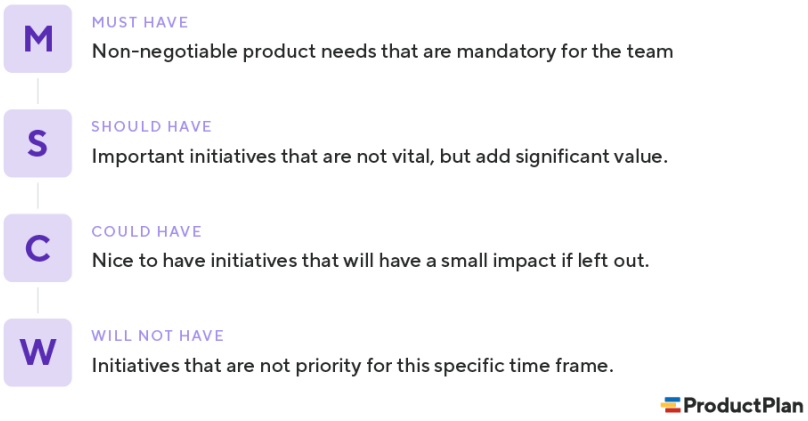
Rozdział 9

Metody pracy, narzędzia i techniki

9.1 *Metody pracy*

Aby dostarczyć minimalny opłacalny produkt [6] aplikacja będzie rozwijana poprzez wdrażanie wymaganych funkcjonalności w kolejności wynikającej z ich priorytetów. Podczas planowania projektu wykorzystana zostanie priorytetyzacja MoSCoW [4] która polega na określeniu priorytetu za pomocą jednej z poniżej wymienionych kategorii, jak miało to miejsce w sekcji *Wymagania funkcjonalne*:

Rysunek 9.1: MoSCoW



W fazie prototypu zostaną wdrożone wszystkie funkcjonalności wymagane (M), natomiast wdrożenie wszelkich pozostałych kategorii zostanie rozpatrzone w fazie rozwoju aplikacji. Plan uwzględnia także cykliczne przeglądy priorytetów aby lepiej dopasować aplikację do potrzeb użytkowników i kierunku rozwoju projektu. Zadania rozpisane zostaną w metodologii kanban [20]. Zgodnie z zasadami LEAN [21] w każdej iteracji kod będzie dodatkowo refaktoryzowany i upraszczany, jeśli zajdzie taka potrzeba i uprości to interfejsy funkcji i zwiększy czytelność dane zostaną także zebrane w dedykowane klasy.

9.2 *Narzędzia*

Podczas projektowania i wdrożenia aplikacji wykorzystane zostaną narzędzia typu Open Source oraz komerycjne dostępne nieodpłatnie dla użytkowników indywidualnych.

Kategoryzacja MoSCoW [4] dla poszczególnych funkcjonalności wykonywana będzie na zadaniach zarejstrowanych w tablicy kanban, w serwisie serwisu Trello [15]. Do stworzenia bazy SQLite [9] posłuży aplikacja DataGrid [19]. Aplikacja Visual Studio Code [18] posłuży do pisania kodu w Python [11] oraz dokumentacji w LaTeX [17]. Do rozwoju dokumentacji oraz kodu aplikacji posłuży system kontroli źródła GIT [22], a oba kody źródłowe przechowywane będą w osobnych projektach na platformie GitHub [23].

9.3 *Techniki*

W trakcie tworzeznia projektu wykorzystano model przyrostowy [24] w oparciu o klasyfikację funkcji do wdrożenia metodą MoSCoW [4]. Ponadto stosowane będą techniki programowania LEAN [21] poprawiające czytelność i jakość tworzonego kodu.

Rozdział 10

Opisy metod

10.1 *Główne klasy projektu*

Projekt składa się z warstw bazy danych oraz graficznego interfejsu aplikacji. Baza danych przechowuje dane wprowadzone przez użytkownika w tabelach oraz generuje podsumowania i zestawienia w formie widoków. Oba typy obiektów składają się na główne klasy projektu. Warstwa graficznego interfejsu użytkownika odpowiedzialna jest za interakcję z użytkownikiem oraz interakcję użytkownika z bazą - prezentację danych przechowywanych w bazie oraz wizualizacje danych. Dodatkowo zbudowano w niej klasy upraszczające interfejs poszczególnych funkcji.

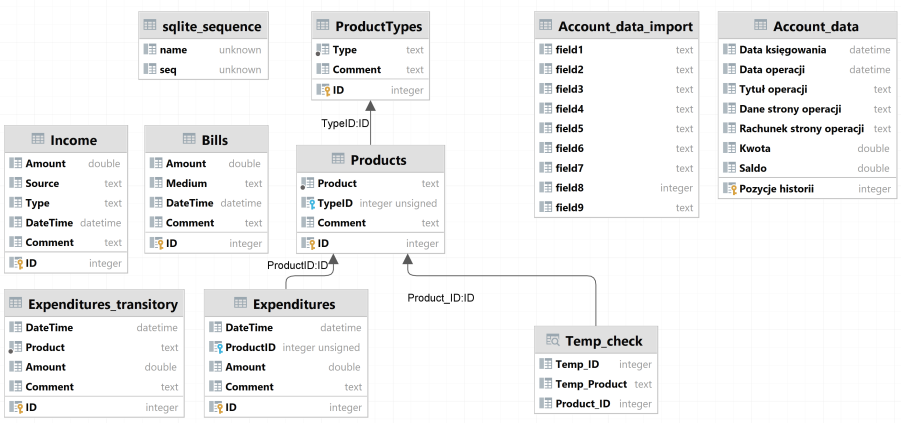
10.2 *Baza danych*

Podczas tworzenia bazy danych przyjęto kilka podstawowych założeń aby utrzymać spójną konwencję nazewniczą pól, tabel i widoków. Dzięki niej interfejs bazy jest prostszy a pisanie zapytań bardziej intuicyjne co w ogólnym rozrachunku powinno ograniczyć nakład pracy wymagany do wdrożenia dodatkowych funkcji.

Pole	**Opis**
ID	Identyfikator rekordu
Comment	Komentarz użytkownika
DateTime	Znacznik w standardzie daty międzynarodowej ISO 8601 [7]
Amount	Koszt, liczba zmiennoprzecinkowa
Pole specyficzne	Główna informacja, różne nazwy w każdej tabeli (Type,Product)

Tabela 10.1: Konwencja nazewnicza bazy danych

Rysunek 10.1: Klasy warstwy bazy danych - tabele




```
01 |  CREATE TABLE [ProductTypes] (
02 |      [ID]                INTEGER PRIMARY KEY AUTOINCREMENT,
03 |      [Type]              TEXT    NOT NULL,
04 |      [Comment]   TEXT    DEFAULT NULL
05 |  );
```

Listing 10.1: Tabela ProductTypes

Tabela ProductTypes zawiera typy produktów zdefiniowane przez użytkownika.

```
01 |  CREATE TABLE [Products] (
02 |      [ID]         INTEGER PRIMARY KEY AUTOINCREMENT,
03 |      [Product]    TEXT    NOT NULL,
04 |      [TypeID]     INTEGER UNSIGNED,
05 |      [Comment]    TEXT    DEFAULT NULL,
06 |  FOREIGN KEY([TypeID]) REFERENCES ProductTypes(ID)
07 |  );
```

Listing 10.2: Tabela Products

Tabela Products zawiera produkty zdefiniowane przez użytkownika, pole TypeID zawiera klucz obcy ID z tabeli ProductTypes.

```
01 |  CREATE TABLE [Bills] (
02 |      [ID]         INTEGER PRIMARY KEY AUTOINCREMENT,
03 |      [Amount]     DOUBLE,
04 |      [Medium]     TEXT,
05 |      [DateTime]   DATETIME,
06 |      [Comment]    TEXT DEFAULT NULL
07 |  );
```

Listing 10.3: Tabela Bills

Tabela Bills zawiera wydatki stałe wprowadzone przez użytkownika.

```
CREATE TABLE [Income] (
        [ID]            INTEGER PRIMARY KEY AUTOINCREMENT,
        [Amount]        DOUBLE,
        [Source]        TEXT,
        [Type]          TEXT,
        [DateTime]      DATETIME,
        [Comment]       TEXT DEFAULT NULL
);
```

Listing 10.4: Tabela Income

Tabela Income zawiera przychody wprowadzone przez użytkownika.

```
CREATE TABLE [Expenditures] (
        [ID]    INTEGER,
        [DateTime]      DATETIME,
        [ProductID]     INTEGER UNSIGNED,
        [Amount]        DOUBLE,
        [Comment]       TEXT DEFAULT NULL,
        PRIMARY KEY([ID] AUTOINCREMENT),
        FOREIGN KEY([ProductID]) REFERENCES [Products]([ID
    ])
);
```

Listing 10.5: Tabela Expenditures

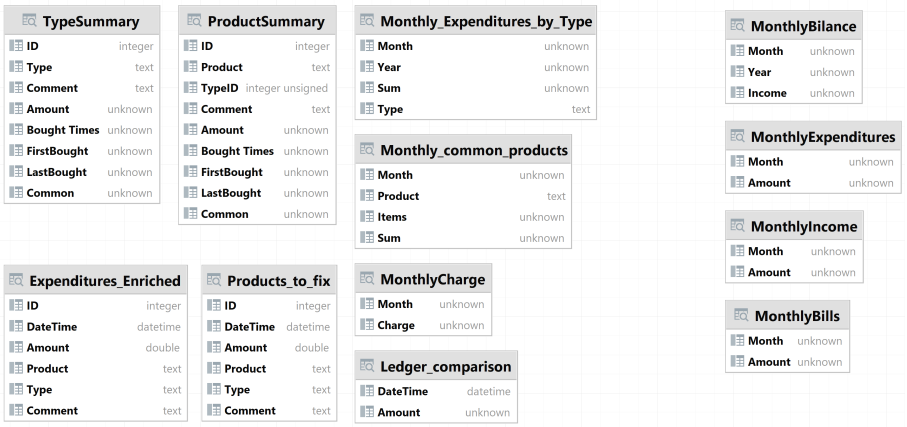
Tabela Expenditures zawiera wydatki wprowadzone przez użytkownika.

```
CREATE TABLE [Expenditures_transitory] (
        [ID]    INTEGER,
        [DateTime]      DATETIME,
        [ProductID]     INTEGER UNSIGNED,
        [Amount]        DOUBLE,
        [Comment]       TEXT DEFAULT NULL,
        PRIMARY KEY([ID] AUTOINCREMENT),
        FOREIGN KEY([ProductID]) REFERENCES [Products]([ID
    ])
);
```

Listing 10.6: Tabela Expenditures_transitory

Tabela Expenditures_transitory jest tabelą tymczasową, przechowuje wydatki wprowadzone przez użytkownika które nie przeszły walidacji. Użytkownik poprawia je po czym prawidłowe dane są przenoszone do tabeli Expenditures i usuwane z Expenditures_transitory.

Rysunek 10.2: Klasy warstwy bazy danych - widoki




```
01 |  CREATE VIEW [Expenditures_Enriched] AS
02 |  SELECT  [EXP].[ID]           as ID,
03 |          [EXP].[DateTime]     as DateTime,
04 |          [EXP].[Amount]       as Amount,
05 |          [PRD].[Product]      as Product,
06 |          [PTY].[Type]         as Type,
07 |          [EXP].[Comment]      as Comment
08 |  FROM  [Expenditures]             [EXP]
09 |  LEFT JOIN [Products]             [PRD]
10 |      ON [EXP].[ProductID]=[PRD].[ID]
11 |  LEFT JOIN [ProductTypes]         [PTY]
12 |      ON [PRD].[TypeID]=[PTY].[ID]
13 |  ORDER BY DateTime;
```

Listing 10.7: Widok Expenditures_Enriched

Widok Expenditures_Enriched prezentuje użytkownikowi czytelne dane z tabeli Expenditures wzbogacone o produkty zdefiniowane w tabeli Products i typy z tabeli ProductTypes.

```
01 |  CREATE VIEW [MonthlyExpenditures] AS
02 |  SELECT
03 |      SUBSTR([DateTime], 1, 7)     as [Month]
04 |      ,SUM([Amount])               as [Amount]
05 |  FROM [Expenditures_Enriched]
06 |  GROUP BY [Month]
07 |  ORDER BY [Month];
```

Listing 10.8: Widok MonthlyExpenditures

Widok MonthlyExpenditures podsumowuje dane o miesięcznych wydatkach użytkownika.

```
01 |  CREATE VIEW [MonthlyBills] AS
02 |  SELECT
03 |      SUBSTR([DateTime], 1, 7)     as [Month]
04 |      ,SUM([Amount])               as [Amount]
05 |  FROM [Bills]
06 |  GROUP BY [Month]
07 |  ORDER BY [Month];
```

Listing 10.9: Widok MonthlyBills

Widok MonthlyBills dane o rachunkach bieżących użytkownika w rozrachunku miesięcznym na podstawie danych zawartych w tabeli Bills.

```
01 |  CREATE VIEW [MonthlyIncome] AS
02 |  SELECT
03 |          SUBSTR([DateTime], 1, 7)
                  as Month
04 |          ,SUM([Amount])
                        as Amount
05 |  FROM     [Income]
06 |  GROUP BY [Month]
07 |  ORDER BY [Month];
```

Listing 10.10: Widok MonthlyIncome

Widok MonthlyIncome podsumowuje dane o przychodach użytkownika w ujęciu miesięcznym.

```
01 |  CREATE VIEW [MonthlyBilance] AS
02 |  SELECT
03 |      Strftime('%Y-%m', [DateTime])   as [Month],
04 |      Strftime('%Y',    [DateTime])   as [Year],
05 |      ROUND(SUM([Amount]), 2)         as [Income]
06 |      --Previous_month_income - (bills + expenditures)
07 |  FROM (SELECT
08 |          DATE(Strftime('%Y-%m-01', [DateTime]),[-1 month])
       as [DateTime],
09 |          [Amount]
10 |        FROM [Income]
11 |        UNION SELECT
12 |                  [DateTime],
13 |                  -([Amount])
14 |               FROM [Expenditures_Enriched]
15 |        UNION SELECT
16 |                  [DateTime],
17 |                  -([Amount])
18 |               FROM [Bills])
19 |   GROUP BY [Month]
20 |   ORDER BY [Month] DESC;
```

Listing 10.11: Widok MonthlyBilance

Widok MonthlyBilance podsumowuje bilans miesięczny wydatków i wpływów użytkownika w formie pojedynczej liczby.

```
01 |  CREATE VIEW [Monthly_common_products] AS
02 |  SELECT *
03 |  FROM (
04 |      SELECT  Strftime('%Y-%m', [DateTime])   AS [Month],
05 |                                                 [Product],
06 |              COUNT([Product])                AS [Items],
07 |              SUM([Amount])                   AS [Sum]
08 |      FROM [Expenditures_Enriched]
09 |      GROUP BY [Product], [Month]
10 |      ORDER BY [Month] DESC, [Sum] DESC
11 |  ) WHERE [Items]>=4;
```

Listing 10.12: Widok Monthly_common_products

Widok Monthly_common_products podsumowuje dane o produkach które użytkownik kupował najczęściej każdego miesiąca. Uwzględnia wyłącznie produkty które zakupiono 4 razy - liczbę wybrano arbitralnie metodą kolejnych przybliżeń aby otrzymać zadowalający wynik.

```
01 |  CREATE VIEW [Monthly_Expenditures_by_Type] AS
02 |  SELECT  Strftime('%Y-%m', [DateTime]) as [Month],
03 |          Strftime('%Y',    [DateTime]) as [Year],
04 |          ROUND(SUM([Amount]), 2)       as [Sum],
05 |          [Type]
06 |  FROM (SELECT
07 |          [DateTime],
08 |          [Type],
09 |          [Amount]
10 |        FROM [Expenditures_Enriched])
11 |  GROUP BY [Type], [Month]
12 |  ORDER BY [Month] DESC, [Sum] DESC;
```

Listing 10.13: Widok Monthly_Expenditures_by_Type

Widok Monthly_Expenditures_by_Type podsumowuj dane o typach produktów zakupionych przez użytkownika w ujęciu miesięcznym.

```
01 |  CREATE VIEW [Temp_check]
02 |      (Temp_ID, Temp_Product, Product_ID)
03 |  AS
04 |  SELECT *
05 |  FROM (SELECT
06 |          Expenditures_transitory.ID      as [Temp_ID],
07 |          Expenditures_transitory.Product as [Temp_Product],
08 |          Products.ID                     as [Product_ID]
09 |        FROM [Expenditures_transitory]
10 |        LEFT OUTER JOIN [Products]
11 |        ON [Expenditures_transitory].[Product]==[Products].[
       Product]
12 |  )
13 |  WHERE [Product_ID] IS NULL;
```

Listing 10.14: Widok Temp_check

Widok Temp_check weryfikuje poprawność danych wprowadzonych przez użytkownika.

```
01 |  CREATE VIEW [Products_to_fix] AS
02 |  SELECT *
03 |  FROM [Expenditures_Enriched]
04 |  WHERE [Product] IN (NULL,
05 |                      'UNKNOWN')
06 |      OR [Comment] LIKE '%[TODO]%';
```

Listing 10.15: Widok Products_to_fix

Widok Products_to_fix zawiera dane wprowadzone przez użytkownika poprawnie i oznaczone jako dane do uzupełnienia specjalnymi etykietami - wartością w polu PRODUCT=UNKNOWN lub [TODO] w komentarzu.

```
01 |      CREATE VIEW IF NOT EXISTS [TypeSummary] AS
02 |      SELECT
03 |          [ProductTypes].[ID]                          AS [ID]
04 |          ,[ProductTypes].[Type]                     AS [Type]
05 |          ,[ProductTypes].[Comment]                AS [Comment]
06 |          ,IFNULL([Summary].[Amount], 0)           AS [Amount]
07 |       ,IFNULL([Summary].[Bought Times], 0) AS [Bought Times]
08 |          ,[Summary].[FirstBought]             AS [FirstBought]
09 |          ,[Summary].[LastBought]              AS [LastBought]
10 |          ,IFNULL([Summary].[Common], 'Absent')   AS [Common]
11 |      FROM [ProductTypes]
12 |      LEFT JOIN (SELECT
13 |          *
14 |          ,(CASE WHEN ([Bought Times]>(
15 |              SELECT
16 |                  AVG([Bought Times]) AS [Average]
17 |              FROM (SELECT
18 |                      [Type]
19 |                      ,Round(SUM([Amount]), 2) AS [Amount]
20 |                      ,COUNT([DateTime]) AS [Bought Times]
21 |                      ,MAX([DateTime])     AS [LastBought]
22 |                      ,MIN([DateTime])    AS [FirstBought]
23 |                  FROM [Expenditures_Enriched]
24 |                  GROUP BY [Type]
25 |                  ORDER BY [Bought Times] DESC)))
26 |          then 'Common' else 'Uncommon' end) as [Common]
27 |
28 |      FROM (      SELECT
29 |                [Type]
30 |                ,Round(SUM([Amount]), 2) AS [Amount]
31 |                ,COUNT([DateTime])       AS [Bought Times]
32 |                ,MAX(DateTime)           AS [LastBought]
33 |                ,MIN(DateTime)           AS [FirstBought]
34 |              FROM [Expenditures_Enriched]
35 |              GROUP BY [Type]
36 |              ORDER BY [Bought Times] DESC))
37 |      AS [Summary]
38 |      ON [ProductTypes].[Type]=[Summary].[Type];
39 |
```

Listing 10.16: Widok TypeSummary

Widok TypeSummary podsumowuj dane o typach produktów użytkownika.

```
01 |      CREATE VIEW IF NOT EXISTS [ProductSummary] AS
02 |      SELECT
03 |          [Products].[ID]                                AS [ID]
04 |          ,[Products].[Product]                       AS [Product]
05 |          ,[Products].[TypeID]                         AS [TypeID]
06 |          ,[Products].[Comment]                       AS [Comment]
07 |          ,IFNULL([Summary].[Amount], 0)               AS [Amount]
08 |       ,IFNULL([Summary].[Bought Times], 0) AS [Bought Times]
09 |          ,[Summary].[FirstBought]                 AS [FirstBought]
10 |          ,[Summary].[LastBought]                   AS [LastBought]
11 |          ,IFNULL([Summary].[Common], 'Absent')    AS [Common]
12 |      FROM [Products]
13 |      LEFT JOIN (SELECT
14 |          *
15 |          ,(CASE WHEN ([Bought Times]>(
16 |              SELECT
17 |                  AVG([Bought Times]) AS [Average]
18 |              FROM (SELECT
19 |                      [Product]
20 |                      ,Round(SUM([Amount]), 2) AS [Amount]
21 |                      ,COUNT([DateTime]) AS [Bought Times]
22 |                      ,MAX([DateTime])      AS [LastBought]
23 |                      ,MIN([DateTime])     AS [FirstBought]
24 |                  FROM [Expenditures_Enriched]
25 |                  GROUP BY [Product]
26 |                  ORDER BY [Bought Times] DESC)))
27 |          then 'Common' else 'Uncommon' end) as [Common]
28 |
29 |      FROM (       SELECT
30 |                  [Product]
31 |                  ,Round(SUM([Amount]), 2) AS [Amount]
32 |                  ,COUNT([DateTime])        AS [Bought Times]
33 |                  ,MAX(DateTime)            AS [LastBought]
34 |                  ,MIN(DateTime)            AS [FirstBought]
35 |              FROM [Expenditures_Enriched]
36 |              GROUP BY [Product]
37 |              ORDER BY [Bought Times] DESC))
38 |      as [Summary]
39 |      ON [Products].[Product]=[Summary].[Product];
40 |
```

Listing 10.17: Widok ProductSummary

Widok ProductSummary podsumowujący dla użytkownika statystyki produktów.

10.3 *Logika aplikacji*

W toku prac stopniowo wykorzystywane w projekcie zmienne w formie kolekcji słowników zamieniano w klasy które spajają dane. Wyłoniły się one w wyniku refaktoryzacji i tworzenia abstrakcji upraszczających interfejs funkcji. Klasy projektowano tak, by były w miarę możliwości oczywiste i zrozumiałe, co ma na celu poprawić czytelność i zrozumiałość kodu.

```
01 |  class Database():
02 |      def __init__(self,
03 |                   fullpath,
04 |                   schema,
05 |                   selects,
06 |                   inserts,
07 |                   updates):
08 |          self.fullpath = fullpath
09 |          self.schema = schema
10 |          self.selects = selects
11 |          self.inserts = inserts
12 |          self.updates = updates
13 |
```

Listing 10.18: Klasa Database

Obiekty klasy Database zawierają komplet informacji wymaganych do interakcji z bazą danych wykorzystywaną w aplikacji. Pole fullpath to w pełni kwalifikowana ścieżka do bazy danych, schema jest kolekcją obiektów typu string która przechowuje schemat bazy danych. Pozostałe pola: selects, inserts, updates to słowniki które pozwalają po nazwie odwołać się do odpowiednio zapytań (SELECT), dodawania rekordów do tabel (INSERT), oraz aktualizacji danych w tabelach (updates).

```
01 |  class ChartSelect():
02 |      def __init__(self,
03 |                   database,
04 |                   select,
05 |                   label
06 |                  ):
07 |          self.database=str(database),
08 |          self.select=str(select),
09 |          self.label=str(label)
10 |
```

Listing 10.19: Klasa ChartSelect

Obiekty klasy ChartSelect posiadają trzy atrybuty typu string: database przechowuje w pełni kwalifikowaną ścieżkę do bazy danych aplikacji, select to zapytanie SQL do bazy, natomiast pole label to etykieta wykresu danych wyświetlanego użytkownikowi.

```
01 |  class Chart():
02 |      def __init__(self,
03 |                   selects,
04 |                   caption):
05 |          self.selects = selects
06 |          self.caption = caption
07 |
```

Listing 10.20: Klasa Chart

Obiekty klasy Chart definiują dane do wizualizacji. Atrybut caption przyjmuje wartości typu string wyświetlane jako nagłówek wizualizacji, natomiast atrybut selects jest listą obiektów typu ChartSelect - zbioru zapytań które zostaną wyświetlone w ramach pojedynczej wizualizacji.

```
01 |  class CellEdition():
02 |      def __init__(self,
03 |                   table,
04 |                   ID,
05 |                   field,
06 |                   newvalue,
07 |                   oldvalue):
08 |          self.table = table
09 |          self.ID = ID
10 |          self.field = field
11 |          self.newvalue = newvalue
12 |          self.oldvalue = oldvalue
13 |
14 |      def __repr__(self):
15 |          return "Table % s modified. ID: % s field: % s
       oldvalue: % s newvalue: % s" % (self.table,
16 |                   self.ID,
17 |                   self.field,
18 |                   self.newvalue,
19 |                   self.oldvalue)
20 |
```

Listing 10.21: Klasa CellEdition

Obiekty klasy CellEdition przechowują dane edytowanego przez użytkownika używającego interfejsu aplikacji rekordu bazy danych. Każdy obiekt przechowuje w polach odpowiednio:

- table - tabelę któej dotyczy zmiana
- ID - identyfikator modyfikowanego rekordu
- field - pole które jest zmieniane
- newvalue - nowa wartość pola
- oldvalue - wartość pola przed zmianą

Funkcja składowa __repr__ formatuje dane które zawiera obiekt do postaci tekstu. Obecnie nieużywana, możliwe że w późniejszych etapach zostanie wykorzystana do rejestrowania zdarzenia w logu aplikacji.

10.4 *Graficzny interfejs użytkownika*

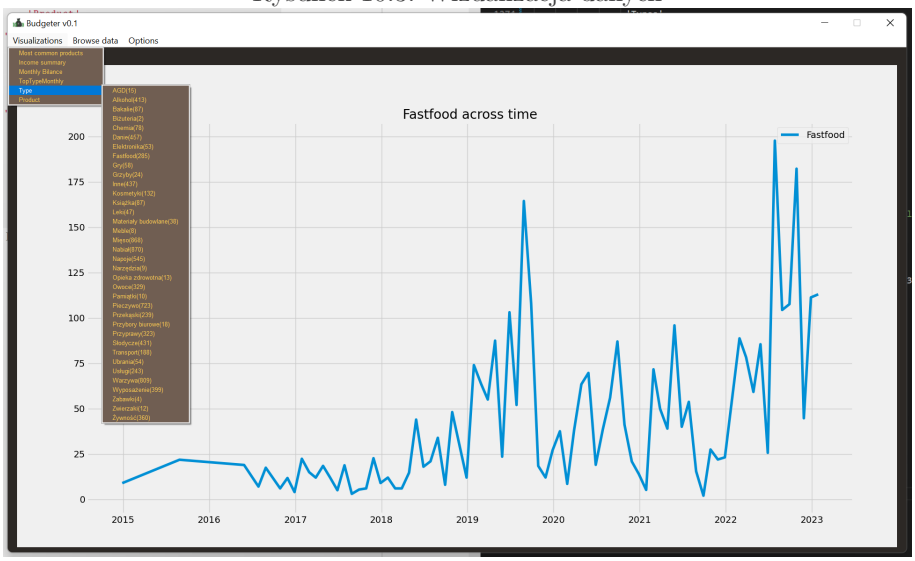
Graficzny interfejs użytkownika (GUI, Graphical User interface) aplikacji utworzono z wykorzystaniem biblioteki PySimpleGUI [12]. Dzięki temu interfejs definiowany jest w postaci kolekcji jak listy, lub listy list, obiektów klas zawartych w bibliotece - jako przykład przedstawiono opcje listy rozwijanej na poniższym listingu. Aby zapewnić responsywność interfejs budowany jest w kilku etapach, a cała budowa wydzielona do specjalnych funkcji generujących wywoływanych później zależnie od potrzeb. Funkcje opisane są w dalszej części w sekcji ***Metody projektu***.

```
01 |       menu = [['Visualizations',
02 |                     ['Most common products',
03 |                      'Income summary',
04 |                      'Monthly Bilance',
05 |                      'TopTypeMonthly',
06 |                      'Type'
07 |                         ,[types],
08 |                      'Product'
09 |                         ,[products]]],
10 |               ['Browse data',
11 |                   #TODO: Add views as uneditable
12 |                   ['Expenditures',
13 |                    'Bills',
14 |                    'Income',
15 |                    'Types' ,
16 |                    'Products',]],
17 |               ['Options',
18 |                   [#'Configure',
19 |                    'Change Theme',
20 |                       [themes],
21 |                   'Version',
22 |                   'About...',
23 |                   'Manual']] #TODO: Wishful thinking - built
       in manual
24 |               ]
25 |
```

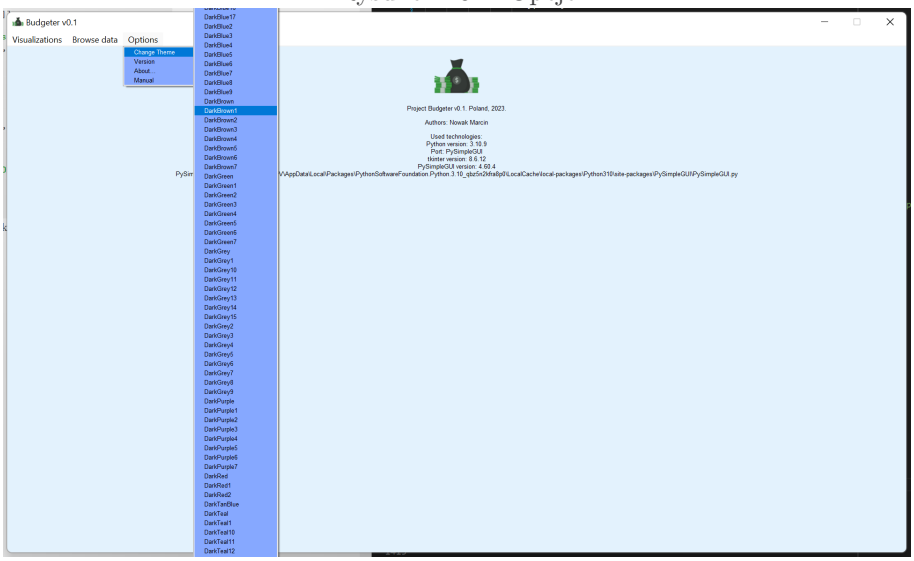
Listing 10.22: Lista rozwijana przykład definicji interfejsu

Interfejs przedstawiono poglądowo na poniższyczh grafikach.

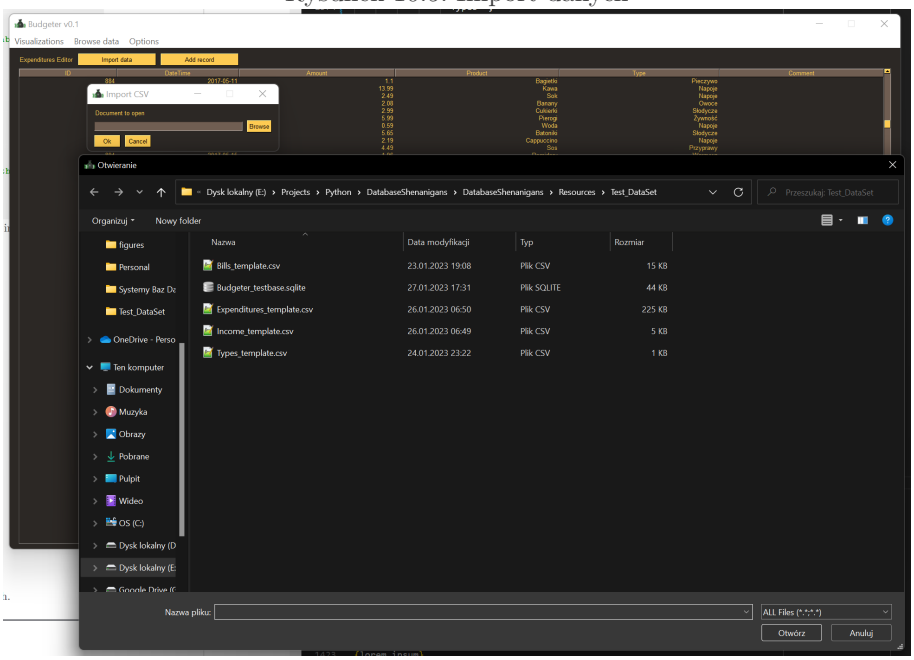
Rysunek 10.3: Wizualizacja danych



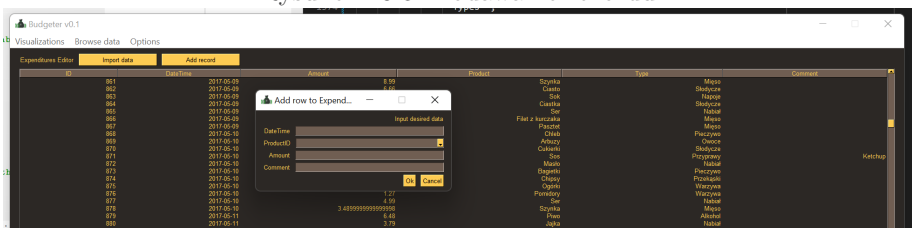
Rysunek 10.4: Opcje



Rysunek 10.5: Import danych



Rysunek 10.6: Dodawanie rekordu



10.5 *Metody projektu*

Kod aplikacji podzielono na funkcje aby zgodnie z dobrymi praktykami zebrać logikę realizującą konkretne zadanie w jednym miejscu.

Funkcja `PrepareStatement(query, values)` przygotowuje i zwraca zapytania dodające lub modyfikujące jeden lub wiele rekordów bazy danych na podstawie zapytania podanego w argumencie `query` i tablicy wartości w argumencie `values`.

Funkcja `GetFromDB(database, select)` pobiera dane z bazy wskazanej argumentem `database` zapytaniem podanym jako argument `select`. Oba argumenty są typu string. Zwraca dane jako tablicę złożoną z poszczególnych pól tablicy w bazie jako pola i poszczególnych wpisów w jako rekordów.

Procedura `SendToDB(database, todb)` wysyła dane do bazy wskazanej argumentem `database` zapytaniem podanym jako argument `select`. Oba argumenty są typu string. Nie zwraca wartości.

Funkcja `GetCollectionFromDB(collection)` przyjmuje jako argument `collection` obiekt klasy Chart który zawiera listę złożoną z zapytań do bazy i odpowiadających im opisów wykresów. Funkcja ta zwraca listę krotek złożonych z tablicy danych pobranych z bazy funkcją `GetFromDB` oraz ciągu znaków klasy string z etykietą danych prezentowaną na wykresie.

Funkcja `GetDBInfo(database)` w argumencie `database` przyjmuje ciąg znaków który jest ścieżką do bazy danych. Wykorzystując funkcję `GetFromDB` oraz predefiniowane zapytania buduje pobiera schemat bazy pobierając listę dostępnych tabel i widoków, a następnie nazwy kolumn każdej z nich. Tak zebrane dane tworzą słownik w którym pod nazwą danej tabeli lub indeksem w słowniku kryje się obiekt z nazwą tablicy jako parametr name, i listą kolumn w polu columns.

Funkcja `Prepare_plot(set, title)` przetwarza dane do wykresu. Przyjmuje w argumencie `set` listę tablic złożonych z listy dat w formie ciagu znaków klasy strting i wartości liczbowych, i pojedynczego ciągu znaków który stanie się etykietą wykresu - dane zwrócone z funkcji `GetCollectionFromDB`. Parsuje daty każdego rekordu do odpowiedniego formatu przy pomocy zewnętrznej biblioteki. Następnie z danych tworzona jest linia wykresu oznaczona etykietą. Na koniec linie łaczone są w jeden wykres, a argument `title` przypisywany jest jako jego nagłówek - tak przygotowany wykres jest zwracany przez funkcję.

Funkcja `draw_figure(canvas, figure)` przetwarza obiekt danych wykresu dostarczony w argumencie `figure` na grafikę prezentowaną w specjalnym elemencie

interfejsu wskazanym w argumencie `canvas`. Zwraca referencję do tak przygotowanego obiektu na ekranie.

Funkcja `Listfromtable(table, addvalues=True)` tworzy z danych tablicy podanej w argumencie `table` listę, jeśli wywołano ją z argumentem `addvalues`, lub nie podano tego argumentu wogóle, dopisuje w nawiasach wartości z tabeli. Zwraca tak utworzoną listę ciągów znaków klasy string.

Funkcja `PrepareCharts()` wstępnie dynamicznie przetwarza część zapytań wywoływanych z interfejsu. Tworzy obiekty klasy ChartSelect - pobiera dane z bazy o kilku najpopularniejszych produktach, dane o najczęściej kupowanym typie produktu, dane o przychodach w ujęciu miesięcznym oraz bilansie miesięcznym przychdó i wydatków. Tak przygotowany słownik obiektów złożony z nazw w formie ciągów znaków klasy string i przypisanych im obiektów klasy ChartSelect jest zwracany.

Funkcja `GivenProduct(product)` przygotowuje w całości obiekt klasy Chart reprezentujący zapytanie do bazy o podsumowanie danych produktu wskazanego w argumencie `product`, po czym wyciąga wskazane dane z bazy funckją `GetCollectionFromDB`, przygotowuje wykres funkcją `Prepare_plot(set, title)` i zwraca go do wyświetlenia.

Funkcja `GivenType(type)` przygotowuje w całości obiekt klasy Chart reprezentujący zapytanie do bazy o podsumowanie danych typu produktów wskazanego w argumencie `type`, po czym wyciąga wskazane dane z bazy funckją `GetCollectionFromDB`, przygotowuje wykres funkcją `Prepare_plot(set, title)` i zwraca go do wyświetlenia.

Funkcja `Visualize(chart)` przyjmuje dane klasy Chart jako argument `chart`, wyciąga wskazane dane z bazy funckją `GetCollectionFromDB`, przygotowuje wykres funkcją `Prepare_plot(set, title)` i zwraca go do wyświetlenia.

Funkcja `TableToLayout(table)` tworzy na podstawie tabeli podanej w argumencie `table` obiekt biblioteki PySimpleGUI - tablicę osadzaną w interfejsie użytkownika.

Funkcja `GenerateTableEditor(table)` tworzy i zwraca interfejs edycji tabeli wskazanej w argumencie `table` w formie ciągu znaków klasy string. Na interfejs składają się: tytuł, przycisk Import Data do importu danych z pliku, przycisk Add record wywołujący okno dodawania rekodru do tabeli oraz osadzoną tabelę uzyskaną wywołaniem funkcji `TableToLayout`.

Funkcja `TableInputWindow(name)` tworzy okno aplikacji służące do edycji danych w tabeli wskazanej atrybutem `name` który jest ciągiem znaków klasy string. Funkcja wykorzystuje dane o schemacie bazy do zbudowania listy pól do wprowadzania danych przez użytkownika. Jeśli napotka pola zawierające identyfikatory które są odwołaniem do innych tabel podmienia je na listę rozwijaną która prezentuje użytkownikowi wyłącznie prawidłowe wartości jako nazwy, natomiast ich wybranie przypisuje w danym polu odpowiadającą mu wartość ID. Funkcja otwiera tak przygotowane okno i zwraca rekord jako słownik w którym nazwa kolumny jest indeksem a wartość wpisanym przez użytkownika tekstem w polu.

Funkcja `ChangeLayout(window, element)` zmienia interfejs użytkownika. Na okno aplikacji składa się kilka zakładek z których jednocześnie tylko jedna z nich jest

aktywna. Funkcja wyłącza obecny aktywny element w oknie wskazanym argumentem `window`, iaktywuje element wskazany argumentem `element` co skutkuje zmianą interfejsu użytkownika wyświetlanego na ekranie.

Funkcja `GetDataFromCSV(filename)` wczytuje dane z pliku CSV którego ścieżkę podano w argumencie `filename`. Funkcja dostosowana jest do obsługi tabel bazy projektu, dlatego zwracany obiekt jest krotką - pierwsza linia pliku traktowana jest jako nagłówek, pozostała zawartość jako dane.

Funkcja `EditCell(window, key, row, col, edition)` jest funkcją dodatkową wyykraczającą poza ramy podstawowego prototypu aplikacji. Pozwala użytkownikowi modyfikować pole rekordu istniejącego w tabeli bazy danych apliacji poprzez kliknięcie na wyświtlony w itnerfejsie rekord. Jako parametry przyjmuje kolejno: w atrybucie `window` okno z którym użytkownik wszedł w interakcję, w atrybucie `key` nazwę tabeli któa jest element wywołującym akcję, wartości numeryczne w atrybutach `row` i `col` które wskazują edytowaną komórkę, i wreszcie w atrybucie `edition` nową wartość wskazanego pola.

Finalnie funkcja nie została udostępniona ponieważ bezpośrednio wykorzystuje biblioteki niskopoziomowe używane w ramach obiektu który reprezentuje element interfejsu biblioteki PySimpleGUI, co wymaga zapoznania się z ich specyfiką i zmniejszenia poziomu abstrakcji systemu. Z uwagi na edukacyjny charakter projektu oraz ograniczone ramy czasowe implementacji planowanych rozwiązań, uznano że funckja ta jest zbyt wymagająca.

Funkcja `PrepareWindow(theme=chosentheme)` przygotowuje główne okno aplikacji. Jako jedyny argument `theme` przyjmuje ciąg znaków klasy string który jest nazwą jednego z dostępnych w bibliotece motywów, a jego wartość domyślna jest zapisana w pliku konfiguracyjnym. Gotowe wygenerowane okno jest zwracane do głównej pętli programu która odczytuje działania użytkownika i podejmuje odpowiednie akcje.

10.6 *Obiekty projektu*

lorem ipsum

10.7 *Struktury projektu*

lorem ipsum

10.8 *Algorytmy projektu*

lorem ipsum

Bibliografia

[1] Wikipedia, Nauki Ekonomiczne
https://pl.wikipedia.org/wiki/Nauki_ekonomiczne

[2] Główny Urząd Statystyczny https://stat.gov.pl/obszary-tematyczne/warunki-zycia/dochody-wydatki-i-warunki-zycia-ludnosci/sytuacja-gospodarstw-domowych-w-2021-r-w-swietle-badania-budzetow-gospodarstw-domowych,3,21.html

[3] Opcje24, Budzetowanie https://www.opcje24h.pl/budzetowanie-przewodnik-planowanie-budzetu/

[4] Product Plan, MOSCOW Prioritetization https://www.productplan.com/glossary/moscow-prioritization/

[5] Praca.pl, Matryca Eisenhowera - czym jest, zasada, prioryteryzacja zadań
https://www.praca.pl/poradniki/rynek-pracy/matryca-eisenhowera-czym-jest,zasada,prioryteryzacja-zadan_pr-2012.html

[6] Wikipedia, Minimal Viable Product
https://en.wikipedia.org/wiki/Minimum_viable_product

[7] NASA.gov, A summary of the international standard date and time notation https://fits.gsfc.nasa.gov/iso-time.html

[8] Y. Shafranovich, SolidMatrix Technologies, Inc., Common Format and MIME Type for Comma-Separated Values (CSV) Files
https://www.rfc-editor.org/rfc/rfc4180

[9] sqlite.org, SQLite https://www.sqlite.org/index.html

[10] wikipedia.org, SQL - Structured Query Language
https://en.wikipedia.org/wiki/SQL

[11] python.org, Python https://www.python.org/

[12] pysimplegui.org, PySimpleGUI Python GUIs for Humans
https://www.pysimplegui.org/en/latest/

[13] json.org, Introducing JSON https://www.json.org/json-en.html

[14] pythonspot.com, Python tutorials, How to Read a File in Python
https://pythonspot.com/read-file/

[15] Atlassian, Trello.com https://trello.com/

[16] MKLabs Co.,Ltd, StarUML https://staruml.io/

[17] The LaTeX Project https://www.latex-project.org/

[18] Microsoft, Visual Studio Code https://code.visualstudio.com/

[19] JetBrains, DataGrid https://www.jetbrains.com/datagrip/

[20] Lean Action PLan, Kanban – układ nerwowy sterowania produkcją w koncepcji Lean Manufacturing https://leanactionplan.pl/kanban/

[21] Wikipedia, Lean software development https://pl.wikipedia.org/wiki/Lean_software_development

[22] git-scm.com, git https://git-scm.com/

[23] https://github.com/ https://github.com/

[24] Wikipedia, Model Przyrostowy https://pl.wikipedia.org/wiki/Model_przyrostowy

Spis rysunków

9.1 MoSCoW 14

10.1 Klasy warstwy bazy danych - tabele 17
10.2 Klasy warstwy bazy danych - widoki 19
10.3 Wizualizacja danych 26
10.4 Opcje 27
10.5 Import danych 27
10.6 Dodawanie rekordu 28

Spis tabel

6.1 Wymagania funkcjonalne 10

10.1 Konwencja nazewnicza bazy danych 16

Listings

10.1 Tabela ProductTypes 17
10.2 Tabela Products 17
10.3 Tabela Bills 17
10.4 Tabela Income 18
10.5 Tabela Expenditures 18
10.6 Tabela Expenditures_transitory 18
10.7 Widok Expenditures_Enriched 19
10.8 Widok MonthlyExpenditures 19
10.9 Widok MonthlyBills 20
10.10Widok MonthlyIncome 20
10.11Widok MonthlyBilance 20
10.12Widok Monthly_common_products 21
10.13Widok Monthly_Expenditures_by_Type 21
10.14Widok Temp_check 21
10.15Widok Products_to_fix 22
10.16Widok TypeSummary 22
10.17Widok ProductSummary 23
10.18Klasa Database 24
10.19Klasa ChartSelect 24
10.20Klasa Chart 24
10.21Klasa CellEdition 25
10.22Lista rozwijana przykład definicji interfejsu 26