

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	<i>Wprowadzenie do tematyki projektu</i>	2
2	<i>Zamierzony cel projektu</i>	3
3	<i>Wstępne założenia i uwarunkowania</i>	4
3.1	<i>Założenia</i>	4
3.2	<i>Uwarunkowania</i>	4
4	<i>Założone ograniczenia i możliwości ewaluacji projektu</i>	5
5	<i>Plan pracy</i>	6
6	<i>Wymagania funkcjonalne</i>	8
7	<i>Wymagania нефункционалне</i>	9
7.1	<i>Sprzętowe wymagania нефункционалне</i>	9
7.2	<i>Systemowe wymagania нефункционалне</i>	9
7.3	<i>Organizacyjne wymagania нефункционалне</i>	9
8	<i>Wymagania danych</i>	10
8.1	<i>Baza danych</i>	10
8.2	<i>Kod aplikacji</i>	10
9	<i>Metody pracy, narzędzia i techniki</i>	12
9.1	<i>Metody pracy</i>	12
9.2	<i>Narzędzia</i>	13
9.3	<i>Techniki</i>	13
10	<i>Opisy metod</i>	14
10.1	<i>Główne klasy projektu</i>	14
10.2	<i>Baza danych</i>	14
10.3	<i>Logika aplikacji</i>	22
10.4	<i>Graficzny interfejs użytkownika</i>	24
10.5	<i>Metody projektu</i>	25
10.6	<i>Obiekty projektu</i>	25
10.7	<i>Struktury projektu</i>	25
10.8	<i>Algorytmy projektu</i>	25

Rozdział 1

Wprowadzenie do tematyki projektu

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

Rozdział 2

Zamierzony cel projektu

Celem projektu jest ułatwienie zarządzania budżetem domowym poprzez dostarczenie użytkownikowi narzędzia pozwalające na analizę wpływów i wydatków, wizualizację trendów oraz automatyczną kategoryzację wpływów i wydatków. Docelowymi odbiorcami aplikacji są użytkownicy domowi, a 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 pozwoli bardziej świadomie podejmować dalsze decyzje finansowe, planować budżet, łatwo identyfikować obszary które wymagają usprawnień czy ogólną obserwację trendów. 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 które o niewielkim lub średnim stopniu zaznajomienia z obsługą komputera. Podczas tworzenia graficznego interfejsu użytkownika cechą która będzie miała największy wpływ na kierunek prac będzie 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 obiektów lub utworzyć i edytować własne. Aplikacja będzie udostępniać predefiniowane wizualizacje, wliczając możliwość wizualizacji określonego przez użytkownika obiektu.

3.2 *Uwarunkowania*

Celem projektu jest dostarczenie minimalnego opłacalnego produktu [5], obecnie pozostałe funkcjonalności zostaną pominięte z różnych przyczyn jak ograniczony czas wdrożenia, zakres umiejętności technicznych autora czy fakt że jest to projekt w głównej mierze edukacyjny. Termin wdrożenia wyklucza bardziej zaawansowane funkcjonalności, jako że jest to projekt edukacyjny 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 zawarte w niej dane będą 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 towarów. Aplikacja nie będzie także udostępniać żadnego rodzaju interfejsu programistycznego (API). W momencie zakończenia projektu wszystkie dane użytkownika przechowywane będą w pojedynczym miejscu, w przyszłości może jednak zająć potrzeba rozdzielenia danych w aplikacji od konfiguracji użytkownika. Interfejs aplikacji będzie statyczny bez możliwości zmiany przez użytkownika.

Rozdział 4

Założone ograniczenia i możliwości ewaluacji projektu

W aplikacji utworzony zostanie panel administracyjny prezentujący użytkownikowi dane statystyczne obrazujące ilość, zakres i jakość danych a także sugerujące kolejny krok ich usprawnienia. 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 skompilowanej wersji spakowanej, jeśli zajdzie taka potrzeba i nastąpi możliwość stworzony zostanie także instalator.

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 i nie zostaną wdrożone w początkowej fazie projektu.

Rozdział 5

Plan pracy

Prace nad projektem prowadzone będą w formie listy zadań do zrealizowania którym przypisane zostaną priorytety metodą MoSCoW [4] lub Matrycy Eisenhowera. 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 powiązań encji
3. Wybór technologii
 - Wspierane systemy i wersje
 - Wybór języka
 - Biblioteki interfejsu użytkownika
 - Sposób przechowywania danych
 - Instalator, aktualizacja i utrzymanie
4. Wstępne wdrożenie
 - Utworzenie struktur bazy danych
 - Wypełnienie danymi testowymi
 - Podstawowe triggery i widoki
 - Projekt interfejsu użytkownika
 - Szkielet interfejsu użytkownika
 - Połączenie interfejsu z bazą danych

-
- Podstawowa wizualizacja
 - Iteracyjne uzupełnienie interfejsu i bazy o dodatkowe funkcje
 - Usprawnienia

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 które 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 bilans zysków i strat
Definiowanie produktów	M	-	Definiowanie produktów
Definiowanie przychodów	M	-	Definiowanie przychodów
Definiowanie typów produktów	M	-	Definiowanie typów produktów
Definiowanie typów przychodów	M	-	Definiowanie typów przychodów
Panel konfiguracyjny	S	-	Osobny panel konfiguracyjny
Rejestr zdarzeń	S	-	Logi z działania aplikacji
Dostęp zdalny	C	-	Dostęp do zdalnych baz danych
Import danych	C	-	Import danych w standardowym formacie
Walidacja danych	C	-	Potwierdzenie jakości danych
Eksport danych	C	-	Eksport danych do standardowego formatu
Instalator	C	-	Prosty instalator aplikacji
Trendy	W	-	Predykcja trendów wydatków i wpływów
Porady	W	-	Porady dla użytkownika dotyczące usprawnień budżetu
Wiele użytkowników	W	-	Wsparcie dla wielu użytkowników jednocześnie
Personalizacja interfejsu	W	-	Personalizacja interfejsu użytkownika
Aktualizacje	W	-	Automatyczne sprawdzanie wersji i aktualizacja

Tabela 6.1: Wymagania funkcjonalne

Rozdział 7

Wymagania нефunkcjonalne

7.1 Sprzętowe wymagania нефunkcjonalne

Pamięć 50MB dowolnego typu, pamięć RAM 2GB, klawiatura, mysz komputerowa, dowolny monitor, opcjonalne połączenie z siecią internet.,

7.2 Systemowe wymagania нефunkcjonalne

System Operacyjny Windows 10, Python3.0, SQLite.

7.3 Organizacyjne wymagania нефunkcjonalne

Aplikacja wspierać będzie działanie z wyłącznie jednym użytkownikiem jednocześnie, każdy z użytkowników będzie korzystał z własnej instancji bazy danych w aplikacji która będzie przechowywana w dowolnej dogodnej określonej przez użytkownika pamięci lokalnej, a także zdalnej jeśli wdrożona zostanie funkcjonalność dostępu zdalnego. Aplikacja wstępnie nie będzie wymagała stałego dostępu do sieci, jednak w przyszłości rozwój jej funkcjonalności 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

Sekcja będzie uzupełniana wraz z rozwojem projektu, w trakcie modelowania i wdrażania tak, aby odzwierciedlała faktyczny stan aplikacji.

Użytkownik będzie wprowadzał dane do aplikacji ręcznie lub za pomocą interfejsu importującego dane w formacie CSV (Comma Separated Values) [7]. Dane wprowadzone przez użytkownika trafią do w bazie danych do zbioru tymczasowego z którego po walidacji potwierdzone prawidłowe dane trafią do zbioru docelowego. Dane które nie przejdą walidacji pomyślnie pozostaną w zbiorze tymczasowym gdzie użytkownik będzie mógł je poprawić, uzupełnić lub usunąć.

8.1 *Baza danych*

Aplikacja wymaga sposobu na trwałe składowanie danych wprowadzonych przez użytkownika. W tym celu dla każdego użytkownika utworzona zostanie dedykowana lokalna bazy danych SQLite [8]. Technologia ta została wybrana ze względu na niewielkie wymagania przestrzeni pamięci trwałej jakie zajmuje oraz w miarę kompletne wsparcie standardowego języka zapytań SQL.

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 10.2

8.2 *Kod aplikacji*

Python jest językiem elastycznym, dzięki czemu dane można przyjmować w dowolnej formie bez definiowania typu, w efekcie klasy będą projektowane w trakcie tworzenia programu.

Aplikacja powinna udostępniać użytkownikowi intuicyjny sposób wizualizacji danych poprzez interfejs użytkownika. Z uwagi na edukacyjny cel projektu aplikacja zostanie zaimplementowana w języku Python [9]. Do implementacji

interfejsu użytkownika posłuży biblioteka PySimpleGUI [10] która pozwala stworzyć prosty i responsywny niezależny od platformy interfejs o ograniczonych możliwościach. Jest to adapter (tak zwany wrapper) który łączy kilka popularnych bibliotek udostępnia spójny interfejs co pozwala budować interfejs za pomocą matrycy złożonych z udostępnianych jako obiekty widżetów.

Poza interfejsem użytkownika aplikacja może umożliwiać prostą konfigurację podstawowych opcji jak rozmiar okna niestandardową lokalizację bazy danych czy wygląd interfejsu. 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 [11]. Rozwiązanie to jest na tyle elastyczne 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 tekstu do pliku.

W języku Python [9] domyślnie wszystkie zmienne są typu Any co pozwala definiować typy zależnie od potrzeb w trakcie tworzenia programu. Wszystkie wczytywane z plików wsadowych dane domyślnie przyjmują typ string lub pojedyncze są traktowane jako pojedyncze bajty [12]. Zatem 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.

W samym kodzie aplikacji podstawowymi 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 wraz z kodem opisane zostaną w sekcjach 10.3 oraz 10.4

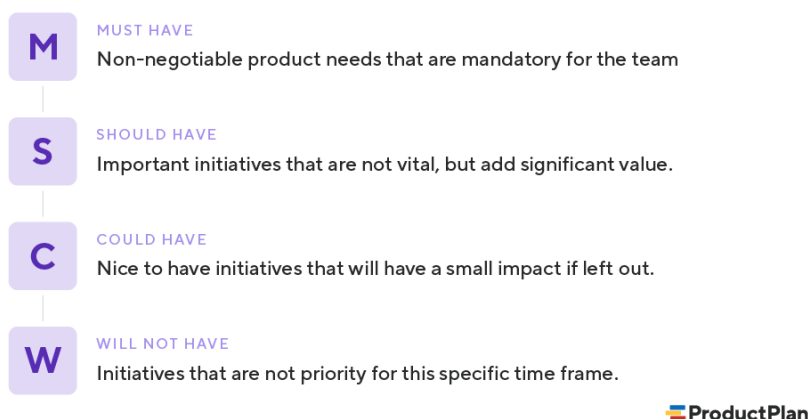
Rozdział 9

Metody pracy, narzędzia i techniki

9.1 *Metody pracy*

Aby dostarczyć minimalny opłacalny produkt [5] aplikacja będzie rozwijana poprzez wdrażanie wymaganych funkcjonalności w kolejności wynikającej z ich priorytetów. W projekcie będzie wykorzystywana priorytetyzacja MoSCoW [4] która polega na określeniu priorytetu za pomocą jednej z kategorii:

Rysunek 9.1: MoSCoW



W fazie projektu zostaną wdrożone wszystkie funkcjonalności wymagane, natomiast wszelkie pozostałe kategorie zostaną wdrożone w miarę możliwości. Plan uwzględni także cykliczne przeglądy priorytetów aby lepiej dopasować aplikację do potrzeb użytkowników i kierunku rozwoju projektu. Zadania rozpisane zostaną w metodologii kanban. W każdej iteracji kod będzie dodatkowo refaktoryzowany i upraszczany, a dane w projekcie abstrahowane do dedykowanych klas jeśli zajdzie taka potrzeba lub uprości to logikę aplikacji i zwiększy czytelność.

9.2 *Narzędzia*

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

Kategoryzacja MoSCoW dla poszczególnych funkcjonalności wykonywana będzie na zadaniach zarejestrowanych w tablicy kanban, z wykorzystaniem serwisu Trello [13]. Model encji w aplikacji zostanie przygotowany w aplikacji StarUML [14]. Do stworzenia bazy SQLite [8] posłuży aplikacja DB Browser for SQLite [17]. Aplikacja Visual Studio Code [16] posłuży do pisania kodu w Python [9] oraz dokumentacji w LaTeX [15].

9.3 *Techniki*

lorem ipsum

Rozdział 10

Opisy metod

10.1 *Główne klasy projektu*

Projekt składa się z warstw bazy danych oraz graficznego interfejsu aplikacji. Warstwa bazy przechowuje dane wprowadzone przez użytkownika, na główne klasy projektu składają się tabele oraz widoki. Warstwa graficznego interfejsu użytkownika odpowiedzialna jest za interakcję z użytkownikiem oraz interakcję użytkownika z bazą - prezentację danych przechowywanych w bazie oraz wizualizację danych.

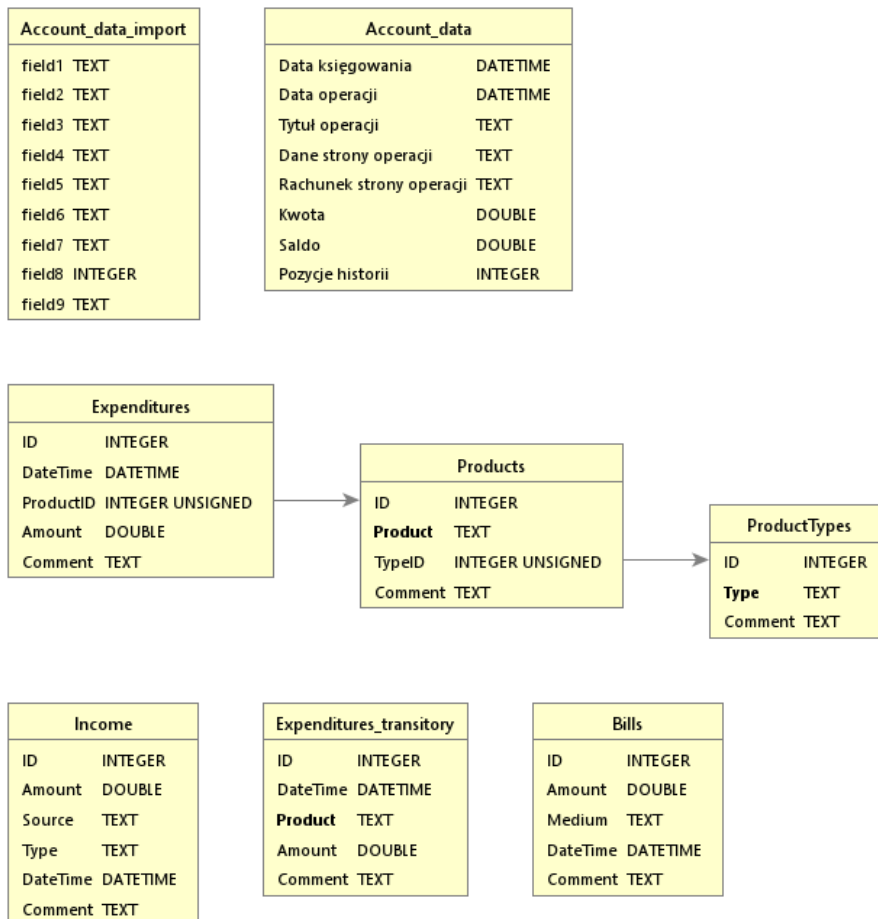
10.2 *Baza danych*

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 [6]
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.

```
01 | CREATE TABLE [Income] (  
02 |     [ID]          INTEGER PRIMARY KEY AUTOINCREMENT,  
03 |     [Amount]     DOUBLE,  
04 |     [Source]     TEXT,  
05 |     [Type]       TEXT,  
06 |     [DateTime]   DATETIME,  
07 |     [Comment]    TEXT DEFAULT NULL  
08 | );
```

Listing 10.4: Tabela Income

Tabela Income zawiera wpływy wprowadzone przez użytkownika.

```
01 | CREATE TABLE [Expenditures] (  
02 |     [ID]          INTEGER,  
03 |     [DateTime]   DATETIME,  
04 |     [ProductID]  INTEGER UNSIGNED,  
05 |     [Amount]     DOUBLE,  
06 |     [Comment]    TEXT DEFAULT NULL,  
07 |     PRIMARY KEY([ID] AUTOINCREMENT),  
08 |     FOREIGN KEY([ProductID]) REFERENCES [Products]([ID]  
09 | )  
09 | );
```

Listing 10.5: Tabela Expenditures

Tabela Expenditures zawiera wydatki wprowadzone przez użytkownika.

```

01 | CREATE TABLE [Expenditures_transitory] (
02 |     [ID] INTEGER,
03 |     [DateTime] DATETIME,
04 |     [ProductID] INTEGER UNSIGNED,
05 |     [Amount] DOUBLE,
06 |     [Comment] TEXT DEFAULT NULL,
07 |     PRIMARY KEY([ID] AUTOINCREMENT),
08 |     FOREIGN KEY([ProductID]) REFERENCES [Products]([ID]
09 | );

```

Listing 10.6: Tabela Expenditures_transitory

Tabela Expenditures_transitory tymczasowo przechowuje wydatki wprowadzone przez użytkownika. Dane które które się w niej znajdują są sprawdzane pod względem poprawności, po czym poprawne dane są przenoszone do tabeli Expenditures i usuwane z Expenditures_transitory. Na dłużej pozostają w niej tylko dane które użytkownik musi poprawić.

```

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 zdefiniowane produkty z tabeli Products i typy z tabeli ProductTypes.

```

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

```

Listing 10.8: Widok TypeSummary

Widok TypeSummary podsumowujący dla użytkownika dane o typach produktów.

```

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

```

Listing 10.9: Widok ProductSummary

Widok ProductSummary podsumowujący dla użytkownika statystyki produktów.

```
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 10.10: Widok MonthlyExpenditures

Widok MonthlyExpenditures podsumowujący dla użytkownika dane o miesięcznych wydatkach.

```
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 10.11: Widok MonthlyBills

Widok MonthlyBills podsumowujący dla użytkownika dane o rachunkach bieżących w rozrachunku miesięcznym zawartych w tabeli Bills.

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

Listing 10.12: Widok MonthlyIncome

Widok MonthlyIncome podsumowujący dla użytkownika dane o przychodach w ujęciu miesięcznym.

```

01 | CREATE VIEW [MonthlyBalance] 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])
09 |     as [DateTime],
10 |     [Amount]
11 |     FROM [Income]
12 |     UNION SELECT
13 |         [DateTime],
14 |         -([Amount])
15 |     FROM [Expenditures_Enriched]
16 |     UNION SELECT
17 |         [DateTime],
18 |         -([Amount])
19 |     FROM [Bills])
20 | GROUP BY [Month]
21 | ORDER BY [Month] DESC;
```

Listing 10.13: Widok MonthlyBalance

Widok MonthlyBalance podsumowujący dla użytkownika bilans miesięczny wydatków i wpływów 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 10.14: Widok Monthly_common_products

Widok Monthly_common_products podsumowujący dla użytkownika dane o najczęściej kupowanych produktach danego miesiąca. Uwzględni wyłącznie produkty które zakupiono 4 razy - liczba ta została wybrana 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 10.15: Widok Monthly_Expenditures_by_Type

Widok Monthly_Expenditures_by_Type podsumowujący dla użytkownika dane o typach produktów 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 10.16: Widok Temp_check

Widok Temp_check służy użytkownikowi do weryfikacji poprawności wprowadzonych danych.

```

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 10.17: Widok Products_to_fix

Widok Products_to_fix zawiera dane wprowadzone przez użytkownika które zostały zaakceptowane jednak wymagają poprawy. Rozpoznawane są po specjalnych wartościach w polu PRODUCT lub oznaczeniu [TODO] w komentarzu.

10.3 Logika aplikacji

W toku prac stopniowo zmienne w formie kolekcji słowników wykorzystywane w projekcie zamieniano w klasy które spajają dane. Wyłoniły się one w odpowiedzi na potrzeby w wyniku refaktoryzacji i tworzenia abstrakcji upraszczających

interfejs funkcji. Klasy projektowano tak, by były w miarę możliwości oczywiste i zrozumiałe, co mapoprawia czytelność i łatwość zrozumienia 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 posiada 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 definiuje dane do wizualizacji. Atrybut caption przyjmuje wartości typu string wyświetlane jako nagłówki 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
16 |               self.ID,
17 |               self.field,
18 |               self.newvalue,
19 |               self.oldvalue)
20 |
```

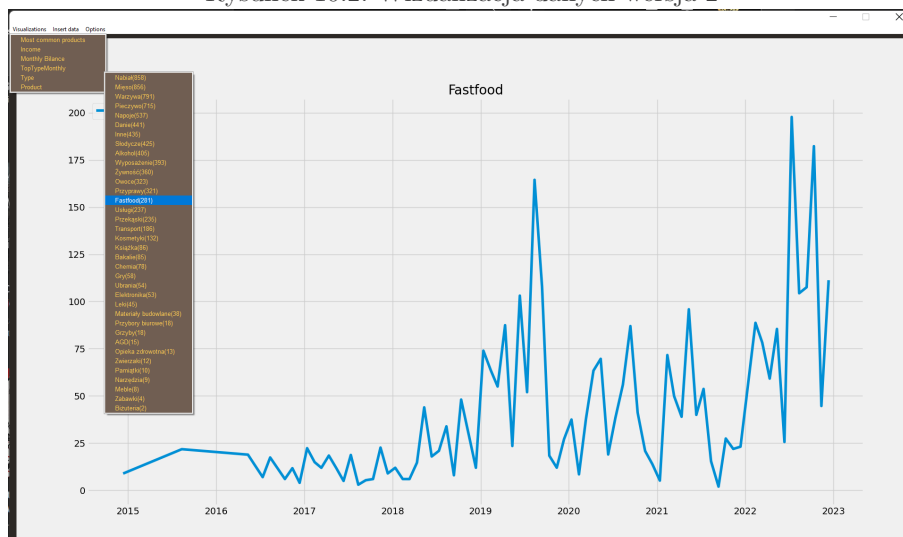
Listing 10.21: Klasa CellEdition

Obiekty klasy CellEdition służy do przechowywania danych rekordu bazy danych edytowanych przez użytkownika poprzez interfejs. Każdy obiekt przechowuje w polach odpowiednio:

10.4 *Graficzny interfejs użytkownika*

Graficzny interfejs użytkownika (GUI, Graphical User interface) aplikacji utworzono z wykorzystaniem biblioteki PySimpleGUI [10]. Dzięki temu interfejs definiowany jest w postaci kolekcji jak listy, lub listy list, obiektów klas zawartych w bibliotece.

Rysunek 10.2: Wizualizacja danych wersja 2



10.5 Metody projektu

lorem ipsum

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-ludnosc-i/sytuacja-gospodarstw-domowych-w-2021-r-w-swietle-badania-budzetow-gospodarstw-domowych,3,21.html>
- [3] Opcje24, Budżetowanie <https://www.opcje24h.pl/budzetowanie-przewodnik-planowanie-budzetu/>
- [4] Product Plan, MOSCOW Prioritization <https://www.productplan.com/glossary/moscow-prioritization/>
- [5] Wikipedia, Minimal Viable Product https://en.wikipedia.org/wiki/Minimum_viable_product
- [6] NASA.gov, A summary of the international standard date and time notation <https://fits.gsfc.nasa.gov/iso-time.html>
- [7] Y. Shafranovich, SolidMatrix Technologies, Inc., Common Format and MIME Type for Comma-Separated Values (CSV) Files <https://www.rfc-editor.org/rfc/rfc4180>
- [8] sqlite.org, SQLite <https://www.sqlite.org/index.html>
- [9] python.org, Python <https://www.python.org/>
- [10] pysimplegui.org, PySimpleGUI Python GUIs for Humans <https://www.pysimplegui.org/en/latest/>
- [11] json.org, Introducing JSON <https://www.json.org/json-en.html>
- [12] pythonspot.com, Python tutorials, How to Read a File in Python <https://pythonspot.com/read-file/>
- [13] Atlassian, Trello.com <https://trello.com/>
- [14] MKLabs Co.,Ltd, StarUML <https://staruml.io/>
- [15] The LaTeX Project <https://www.latex-project.org/>
- [16] Microsoft, Visual Studio Code <https://code.visualstudio.com/>
- [17] Public domain, DB Browser for SQLite <https://sqlitebrowser.org/>

Spis rysunków

9.1	MoSCoW	12
10.1	Klasy warstwy bazy danych - tabele	15
10.2	Wizualizacja danych wersja 2	25

Spis tabel

6.1	Wymagania funkcjonalne	8
10.1	Konwencja nazewnictwa bazy danych	14

Listings

10.1	Tabela ProductTypes	15
10.2	Tabela Products	16
10.3	Tabela Bills	16
10.4	Tabela Income	16
10.5	Tabela Expenditures	16
10.6	Tabela Expenditures_transitory	17
10.7	Widok Expenditures_Enriched	17
10.8	Widok TypeSummary	18
10.9	Widok ProductSummary	19
10.10	Widok MonthlyExpenditures	20
10.11	Widok MonthlyBills	20
10.12	Widok MonthlyIncome	20
10.13	Widok MonthlyBalance	21
10.14	Widok Monthly_common_products	21
10.15	Widok Monthly_Expenditures_by_Type	22
10.16	Widok Temp_check	22
10.17	Widok Products_to_fix	22
10.18	Klasa Database	23
10.19	Klasa ChartSelect	23
10.20	Klasa Chart	24
10.21	Klasa CellEdition	24