|  |  |  |
| --- | --- | --- |
| **Politechnika Świętokrzyska w Kielcach Wydział Elektrotechniki, Automatyki i Informatyki Katedra Informatyki, Elektroniki i Elektrotechniki** | | |
| **Kierunek**  **Informatyka** | **Przedmiot**  **Modelowanie i analiza systemów informatycznych** | |
| **Temat ćwiczenia:**  **Modelowanie i analiza systemu informatycznego realizującego zamianę unitermu pionowej operacji**  **sekwencjowania unitermów na poziomą operację sekwencjowania unitermów** | **Wykonał:**  **Jakub Sztuka** |
| **Grupa dziekańska: 1ID25B** |

### 1. Wprowadzenie

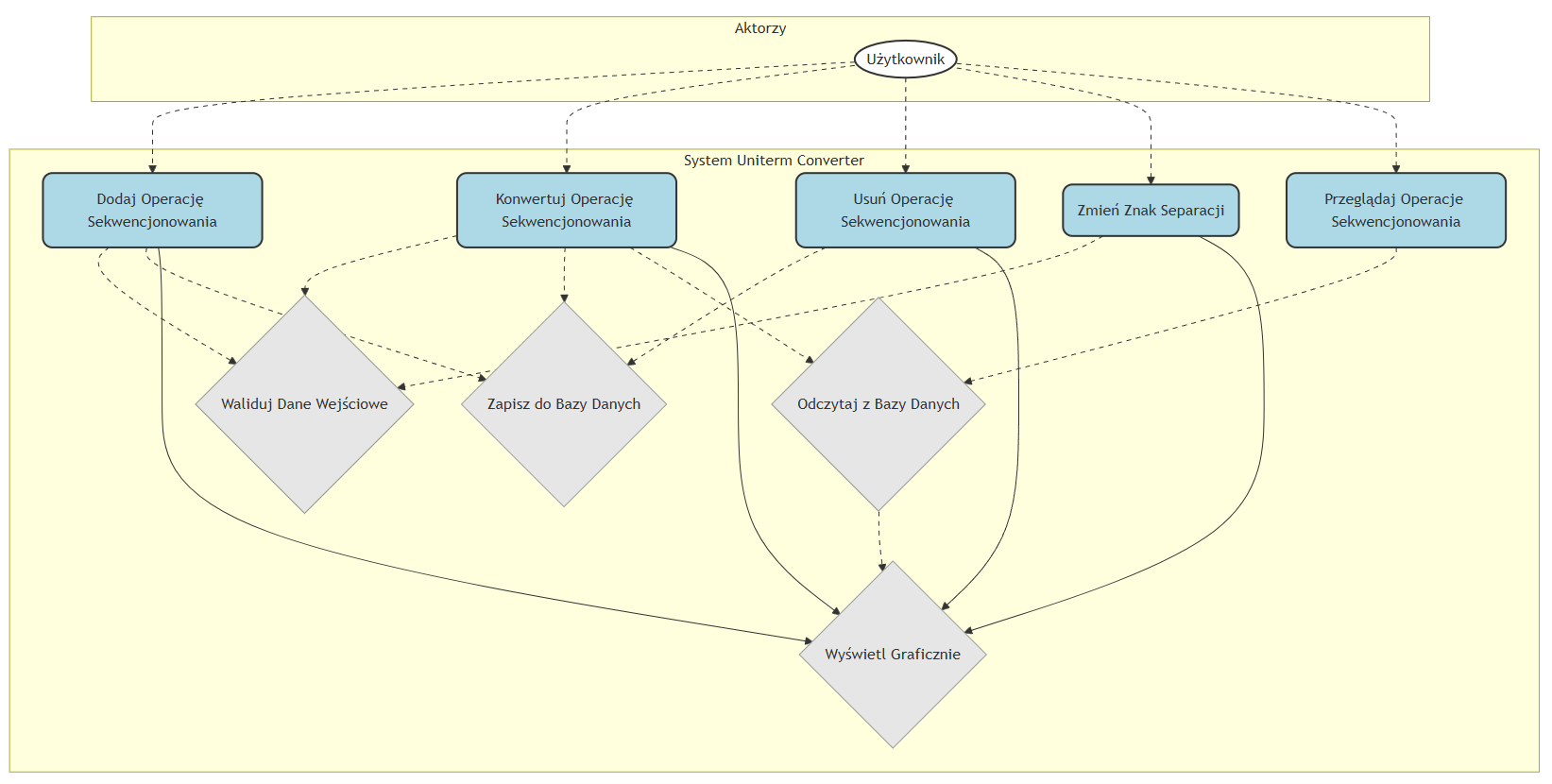
Projekt stanowi system informatyczny zrealizowany w postaci aplikacji desktopowej (GUI) z wykorzystaniem biblioteki Tkinter w języku Python. Głównym celem systemu jest modelowanie i wizualizacja procesu transformacji unitermów sekwencjonowania. Aplikacja umożliwia dodawanie bazowych operacji sekwencjonowania (pionowych i poziomych), ich graficzną reprezentację, a także przeprowadzanie konwersji, polegającej na zamianie wybranego parametru operacji pionowej na operację poziomą. Wszystkie operacje, w tym wyniki konwersji, są trwale przechowywane w bazie danych MySQL, co pozwala na ich przeglądanie, usuwanie oraz analizę. System wspiera dynamiczną zmianę znaku separacji, co pozwala na elastyczną wizualizację.

### 2. Opis kodu

Kod źródłowy projektu został podzielony na kilka modułów, co zapewnia jego czytelność i modułowość:

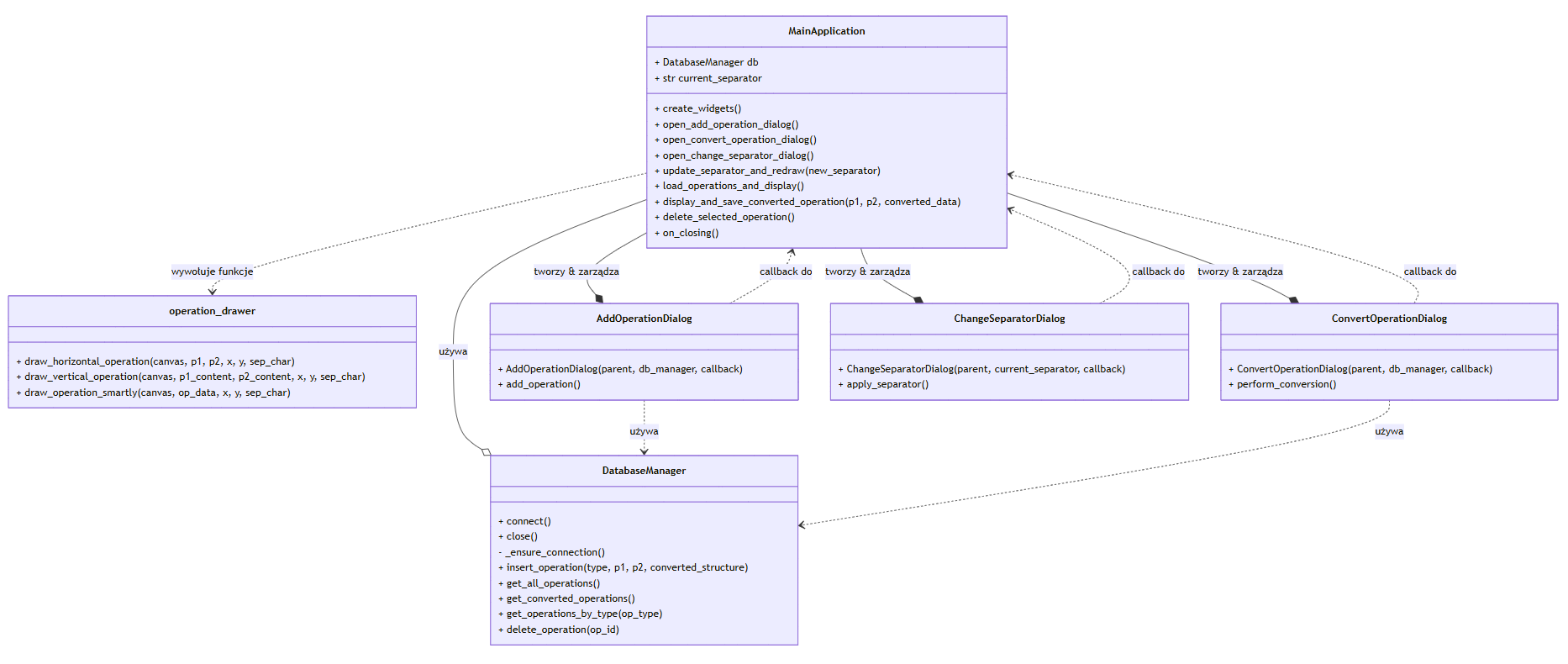
* **main.py**: Główny plik aplikacji, odpowiedzialny za inicjalizację interfejsu użytkownika (GUI), zarządzanie przepływem danych między komponentami, obsługę zdarzeń użytkownika oraz koordynację operacji na danych i ich wizualizację.
  + **Klasa MainApplication**: Centralna klasa aplikacji Tkinter.
    - \_\_init\_\_(): Inicjalizuje główne okno, ustawia jego rozmiar (geometry), konfiguruje style ttk.Style dla widgetów GUI (przyciski, etykiety, tabela) nadając im estetyczny wygląd, inicjalizuje połączenie z DatabaseManager oraz ustala domyślny znak separacji (self.current\_separator).
    - create\_widgets(): Definiuje i rozmieszcza wszystkie elementy interfejsu głównego okna: ramki przycisków, przyciski akcji (dodawanie, konwersja, usuwanie, zmiana separatora), etykiety, tabelę ttk.Treeview do wyświetlania operacji oraz płótno tk.Canvas do graficznej reprezentacji, wraz z paskami przewijania.
    - open\_add\_operation\_dialog(): Wywołuje okno dialogowe AddOperationDialog w celu dodania nowej operacji.
    - open\_convert\_operation\_dialog(): Wywołuje okno dialogowe ConvertOperationDialog w celu przeprowadzenia konwersji.
    - open\_change\_separator\_dialog(): Wywołuje okno dialogowe ChangeSeparatorDialog w celu zmiany znaku separacji.
    - update\_separator\_and\_redraw(): Metoda wywoływana po zmianie znaku separacji; aktualizuje self.current\_separator i odświeża cały widok.
    - clear\_canvas(): Czyści płótno rysunkowe i resetuje jego region przewijania.
    - load\_operations\_and\_display(): Kluczowa metoda do odświeżania GUI. Pobiera wszystkie operacje (w tym konwersje) z bazy danych, czyści Treeview i płótno, a następnie iteruje po pobranych operacjach, wstawiając je do tabeli i rysując graficznie na Canvasie jedną pod drugą. Dynamicznie dostosowuje scrollregion płótna.
    - display\_and\_save\_converted\_operation(): Metoda wywoływana po udanej konwersji. Zapisuje złożoną strukturę skonwertowanej operacji do bazy danych, a następnie odświeża cały widok aplikacji.
    - delete\_selected\_operation(): Obsługuje usuwanie wybranej operacji z listy i bazy danych, z potwierdzeniem użytkownika.
    - on\_closing(): Zapewnia zamknięcie połączenia z bazą danych przy zamykaniu aplikacji.
* **database\_manager.py**: Moduł odpowiedzialny za wszelkie interakcje z bazą danych MySQL.
  + **Klasa DatabaseManager**: Upraszcza zarządzanie połączeniem i operacjami CRUD.
    - \_\_init\_\_() i connect(): Ustanawia połączenie z bazą danych przy starcie.
    - close(): Zamyka aktywne połączenie.
    - \_ensure\_connection(): Kluczowa metoda zapewniająca, że połączenie z bazą danych jest zawsze aktywne; próbuje je nawiązać ponownie, jeśli jest zerwane.
    - insert\_operation(op\_type, param1, param2, converted\_structure=None): Dodaje nową operację do tabeli operations. Opcjonalny parametr converted\_structure (słownik) jest serializowany do formatu JSON i zapisywany w bazie.
    - get\_all\_operations(): Pobiera wszystkie operacje z bazy danych, deserializując converted\_structure z JSON z powrotem do słownika Pythona.
    - get\_operation\_by\_id(op\_id): Pobiera konkretną operację po jej ID, również deserializując dane konwersji.
    - get\_operations\_by\_type(op\_type): Pobiera operacje danego typu (pionowa lub pozioma), ale tylko te, które nie są wynikiem konwersji (tzn. converted\_structure jest NULL).
    - delete\_operation(op\_id): Usuwa operację z bazy danych na podstawie jej ID.
* **operation\_drawer.py**: Moduł zawierający funkcje do graficznego renderowania operacji sekwencjonowania na płótnie Tkinter (Canvas).
  + draw\_horizontal\_operation(canvas, param1, param2, start\_x, start\_y, separator\_char): Rysuje operację poziomą w formacie (param1 ; param2) z łukowym nawiasem nad i używając dynamicznie wybranego znaku separacji.
  + draw\_vertical\_operation(canvas, param1\_content, param2\_content, start\_x, start\_y, separator\_char): Rysuje operację pionową w formacie param1 ; param2 z dużym okrągłym nawiasem po lewej stronie, używając dynamicznie wybranego znaku separacji. Może rekursywnie rysować zagnieżdżone operacje poziome jako parametry.
  + draw\_operation\_smartly(canvas, op\_data, start\_x, start\_y, separator\_char): Funkcja pomocnicza, która na podstawie op\_data['type'] (pionowa/pozioma) wywołuje odpowiednią funkcję rysującą. Jest w stanie obsłużyć zagnieżdżone struktury operacji.
* **add\_operation\_dialog.py**: Moduł implementujący okno dialogowe do dodawania nowych, bazowych operacji sekwencjonowania.
  + **Klasa AddOperationDialog**: Okno tk.Toplevel, które prosi użytkownika o wprowadzenie dwóch parametrów oraz wybór typu operacji (pionowa/pozioma). Po pomyślnym dodaniu wywołuje callback w głównym oknie.
* **convert\_operation\_dialog.py**: Moduł implementujący okno dialogowe do przeprowadzania konwersji unitermów.
  + **Klasa ConvertOperationDialog**: Okno tk.Toplevel, które pozwala użytkownikowi wybrać operację pionową i poziomą, a także zdecydować, który parametr operacji pionowej ma zostać zastąpiony operacją poziomą. Po konwersji przekazuje wynik do main.py w celu zapisu.
* **change\_separator\_dialog.py**: Moduł implementujący okno dialogowe do zmiany znaku separacji używanego w graficznej reprezentacji operacji.
  + **Klasa ChangeSeparatorDialog**: Okno tk.Toplevel, które pozwala użytkownikowi wprowadzić pojedynczy znak, który będzie używany jako separator w wizualizacjach.

### 3. Diagram przypadków użycia



Użytkownik systemu może dodawać nowe bazowe operacje sekwencjonowania (pionowe lub poziome). Może również inicjować proces konwersji, wybierając istniejącą operację pionową i zastępując jeden z jej parametrów inną operacją poziomą. Wszystkie wprowadzone i skonwertowane operacje są automatycznie wyświetlane na liście i graficznie. Użytkownik ma możliwość usunięcia wybranej operacji z listy oraz zmiany znaku separacji używanego w graficznej reprezentacji. Każde wprowadzenie danych oraz operacja jest powiązana z walidacją, zapewniającą poprawność działania.

### 4. Diagram klas



* **MainApplication (z main.py)**: Główna klasa aplikacji, zarządzająca interfejsem użytkownika, stanem aplikacji oraz koordynująca przepływ danych i operacje na danych. Agreguje instancje DatabaseManager i wywołuje okna dialogowe.
* **DatabaseManager (z database\_manager.py)**: Klasa odpowiedzialna za wszelkie interakcje z bazą danych MySQL (połączenie, tworzenie tabel, zapis, odczyt, usuwanie). Działa jako warstwa dostępu do danych.
* **operation\_drawer (moduł z operation\_drawer.py)**: Zestaw funkcji pomocniczych odpowiedzialnych za renderowanie graficznej reprezentacji operacji sekwencjonowania na płótnie Tkinter. Funkcje te przyjmują dane operacji i rysują odpowiednie elementy wizualne.
* **AddOperationDialog (z add\_operation\_dialog.py)**: Klasa reprezentująca okno dialogowe do dodawania nowych operacji. Dziedziczy z tk.Toplevel.
* **ConvertOperationDialog (z convert\_operation\_dialog.py)**: Klasa reprezentująca okno dialogowe do przeprowadzania konwersji operacji. Dziedziczy z tk.Toplevel.
* **ChangeSeparatorDialog (z change\_separator\_dialog.py)**: Klasa reprezentująca okno dialogowe do zmiany znaku separacji. Dziedziczy z tk.Toplevel.

### 5. Diagram aktywności

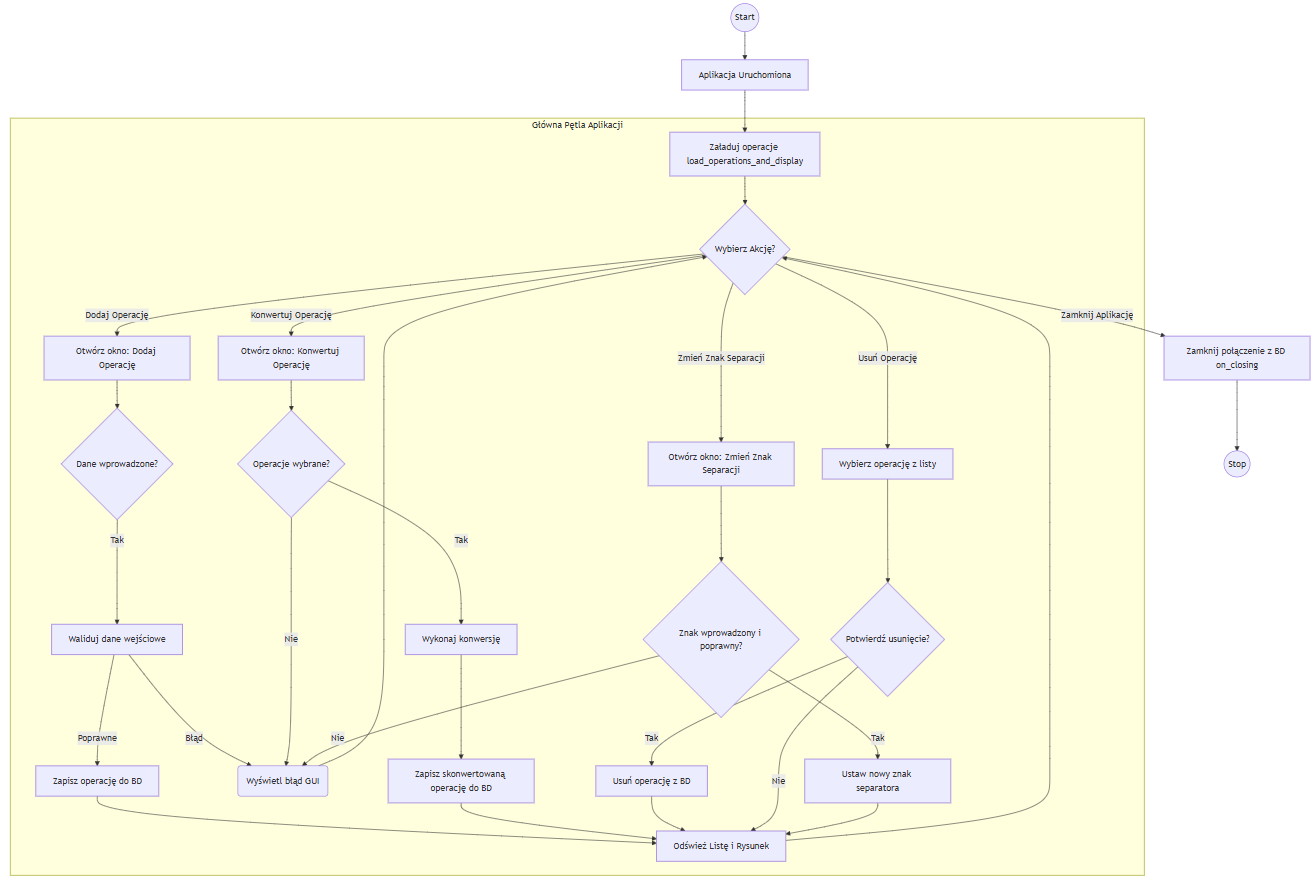
****

Diagram przedstawia kroki podejmowane przez użytkownika i system od momentu inicjalizacji aplikacji i załadowania danych, poprzez dodawanie nowych operacji, aż po ich konwersję i usuwanie. Pokazuje rozgałęzienia w zależności od akcji użytkownika i poprawności danych.

### 6. Diagram sekwencji

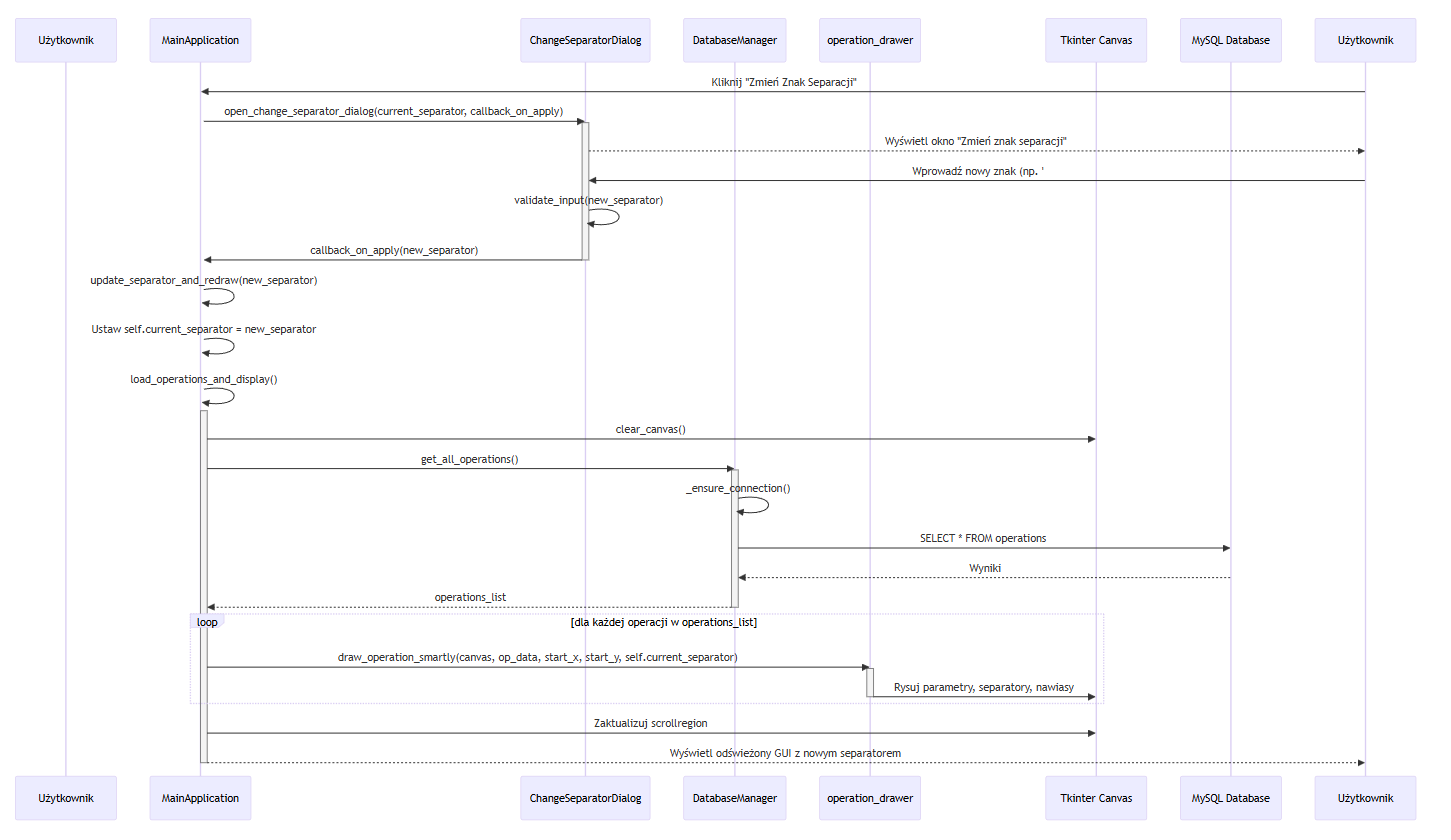
****

Diagram ilustruje sekwencję zdarzeń i wywołań funkcji, gdy użytkownik zmienia znak separacji w aplikacji. Pokazuje, jak komponenty aplikacji współpracują ze sobą w celu przetworzenia żądania i odświeżenia widoku.

### 7. Diagram warstw

Architektura aplikacji jest podzielona na logiczne warstwy, co ułatwia jej zrozumienie, utrzymanie i potencjalną rozbudowę.

### 

### 8. Diagram komponentów

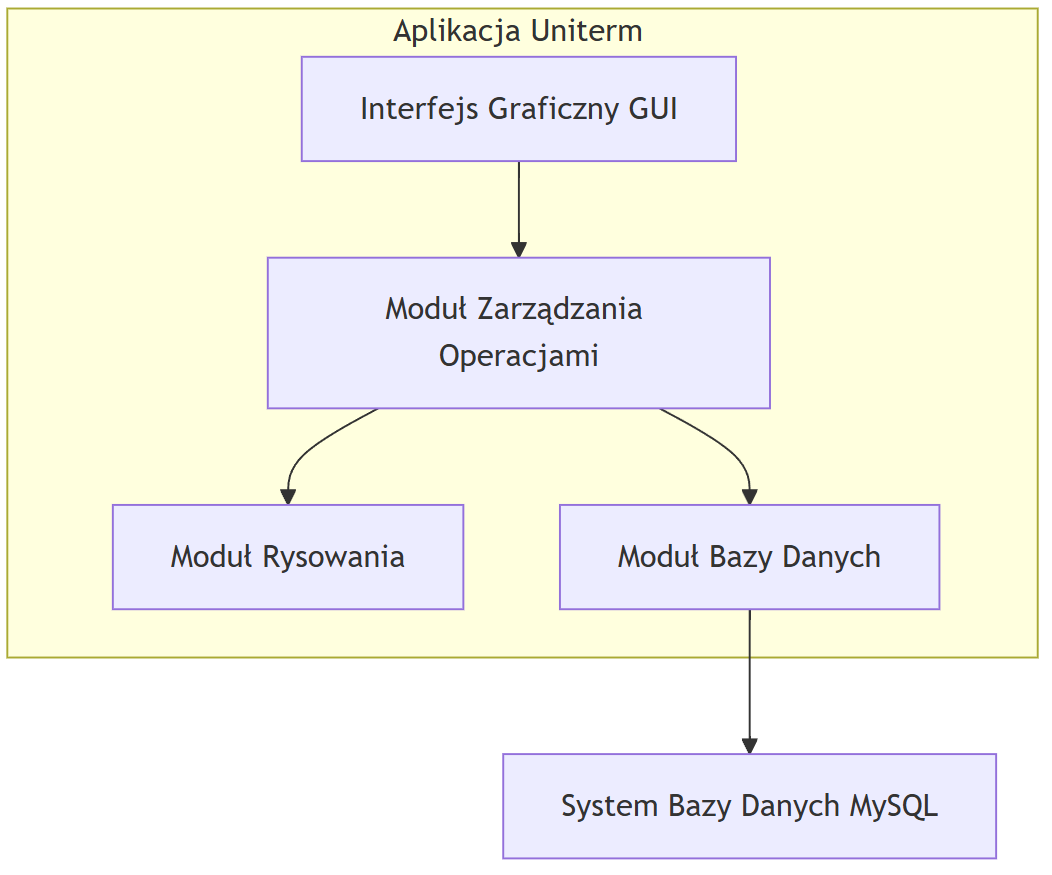


Diagram przedstawia kluczowe komponenty oprogramowania i ich relacje, ilustrując, jak są zorganizowane i współpracują ze sobą.

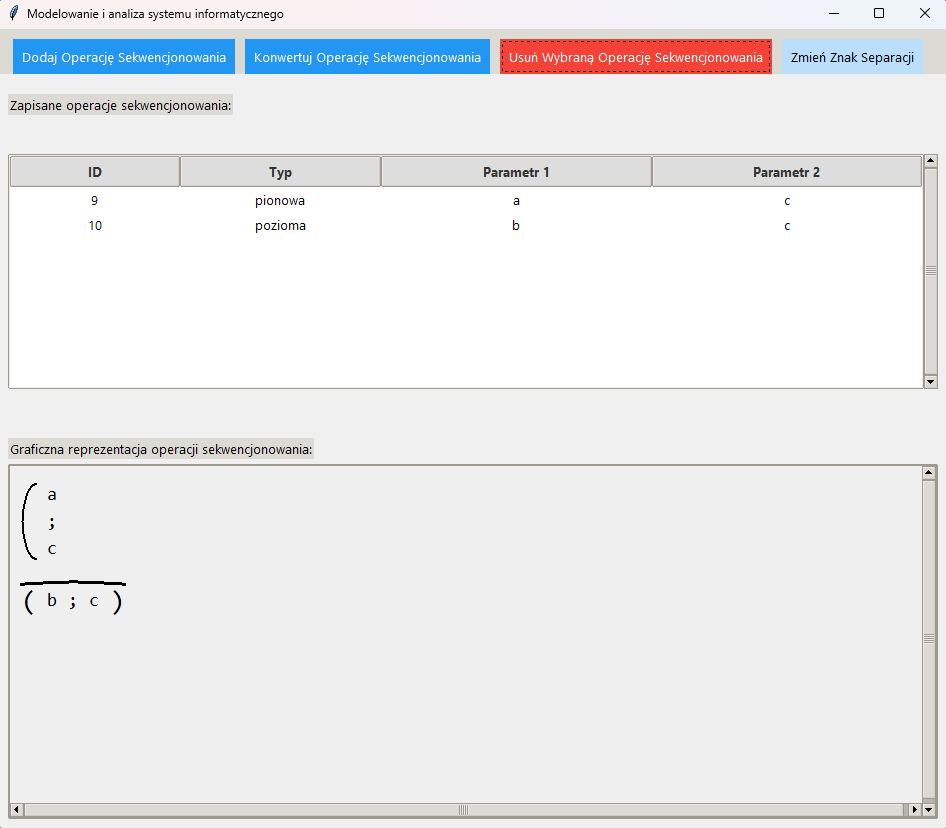
### 9. Struktura bazy danych

**Opis tabeli operations:**  
Tabela operations jest centralnym miejscem przechowywania wszystkich danych o operacjach sekwencjonowania, zarówno bazowych (pionowych i poziomych), jak i tych powstałych w wyniku konwersji.

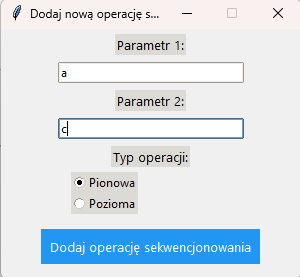
* **id**:
  + **Typ**: INT
  + **Właściwości**: AUTO\_INCREMENT, PRIMARY KEY
  + **Opis**: Unikalny identyfikator każdej operacji (zarówno bazowej, jak i skonwertowanej).
* **type**:
  + **Typ**: VARCHAR(20)
  + **Właściwości**: NOT NULL
  + **Opis**: Określa typ bazowej operacji ('pionowa' lub 'pozioma'). Dla skonwertowanych operacji zawsze będzie to 'pionowa', ponieważ konwersja dotyczy zamiany parametru w operacji pionowej.
* **param1**:
  + **Typ**: VARCHAR(255)
  + **Właściwości**: NOT NULL
  + **Opis**: Przechowuje pierwszy parametr unitermu. Dla operacji skonwertowanych, będzie to oryginalny pierwszy parametr operacji pionowej (niezależnie od tego, czy został zastąpiony).
* **param2**:
  + **Typ**: VARCHAR(255)
  + **Właściwości**: NOT NULL
  + **Opis**: Przechowuje drugi parametr unitermu. Dla operacji skonwertowanych, będzie to oryginalny drugi parametr operacji pionowej (niezależnie od tego, czy został zastąpiony).
* **converted\_structure**:
  + **Typ**: TEXT
  + **Właściwości**: NULL (może być puste)
  + **Opis**: Kolumna przechowująca strukturę skonwertowanej operacji w formacie JSON (tekstowym). Jest wypełniana tylko dla rekordów, które reprezentują wyniki konwersji. Pozwala to na przechowywanie zagnieżdżonych reprezentacji operacji. Dla operacji bazowych (niekonwertowanych) wartość w tej kolumnie będzie NULL.

### 10. Działanie programu

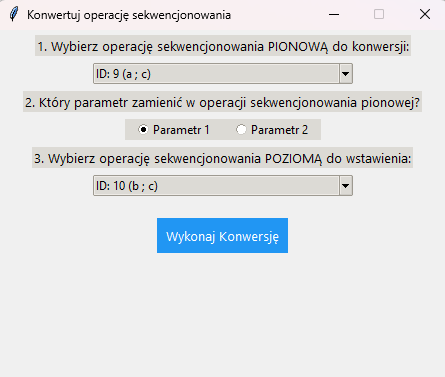
**Wygląd aplikacji po uruchomieniu z listą bazowych operacji**

****

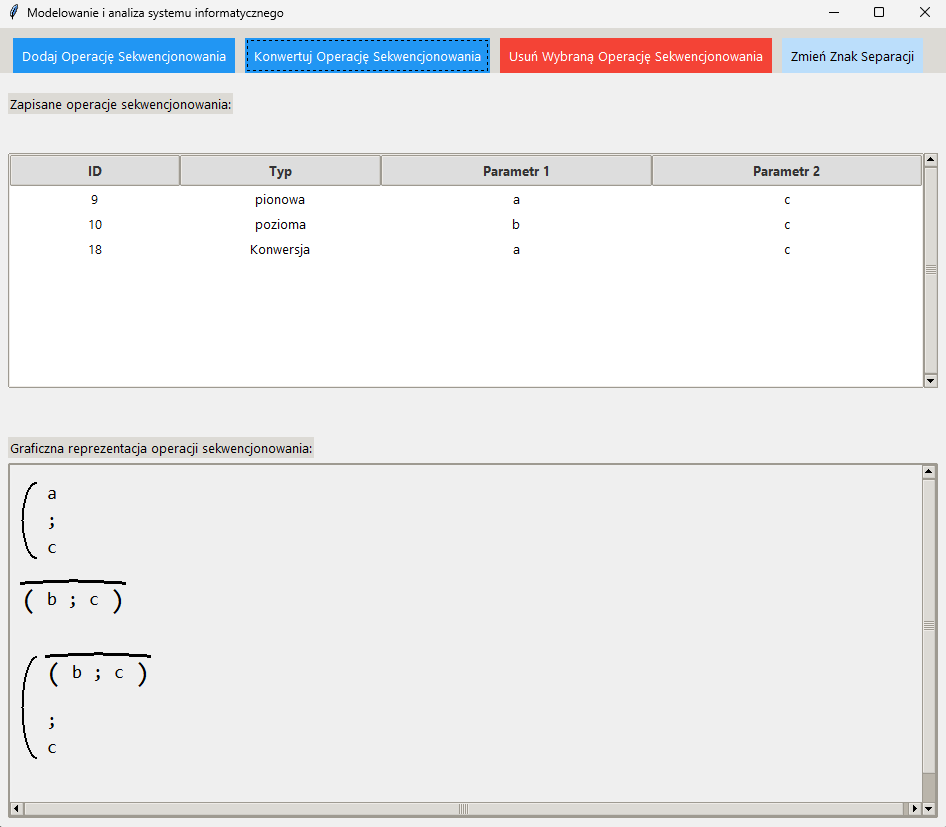
**Okno "Dodaj nową operację sekwencjonowania"**



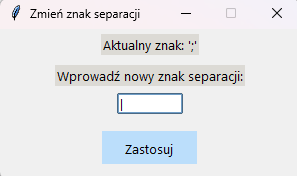
**Okno "Konwertuj operację sekwencjonowania"**



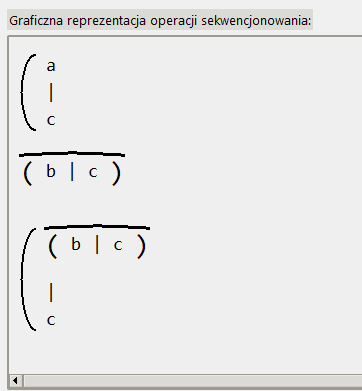
**Aplikacja po przeprowadzeniu konwersji**

****

**Okno "Zmień znak separacji"**



**Wygląd operacji po zmianie znaku**

****

### 11. Wnioski

Zaprojektowany i zaimplementowany system stanowi kompleksowe narzędzie do modelowania i analizy procesu zamiany unitermu pionowej operacji sekwencjonowania unitermów na poziomą operację sekwencjonowania unitermów. Aplikacja skutecznie realizuje postawione przed nią zadania, umożliwiając interaktywne definiowanie operacji sekwencjonowania, przeprowadzanie transformacji (konwersji), a także ich dynamiczną wizualizację.

Kluczowe cechy programu to:

* **Intuicyjny interfejs GUI** zrealizowany w Tkinter, zapewniający łatwość użytkowania.
* **Trwałe przechowywanie danych** (operacji bazowych i wyników konwersji) w bazie danych MySQL, co zapewnia ich bezpieczeństwo i dostępność.
* **Graficzna reprezentacja unitermów**, w tym możliwość wizualizacji złożonych, zagnieżdżonych struktur po konwersji.
* **Elastyczność wizualizacji** dzięki funkcji zmiany znaku separacji.
* **Mechanizmy obsługi błędów i walidacji**, które zwiększają stabilność i niezawodność aplikacji, zapobiegając jej awariom w przypadku nieprawidłowych danych wejściowych czy problemów z bazą danych.

Architektura aplikacji, oparta na podziale na logiczne moduły (GUI, logika biznesowa, dostęp do danych, rysowanie), ułatwia zrozumienie kodu, jego utrzymanie oraz potencjalną rozbudowę o nowe funkcjonalności w przyszłości. System stanowi solidną podstawę do dalszych badań i eksperymentów w dziedzinie analizy i transformacji unitermów sekwencjonowania.