

WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ CYBERNETYKI



Projekt z przedmiotu Programowanie w językach funkcyjnych

Temat pracy: Aplikacja okienkowa To do List

Autorka: **Ołowniuk Julia**

Grupa: **WCY21KA1S1**

Nr Albumu: **80968**

Warszawa 2024

Spis treści

I.	Wstęp	5
I.1.	Github	5
I.2	Ogólny opis aplikacji	5
II.	Opis implementacji/ technologii w funkcjach	6
II.1.	Dodawanie zadań	6
II.2.	Oznacz zadanie jako zrobione	7
II.3.	Oznacz jako niezrobione	7
II.4.	Edytuj nazwę zadania	8
II.5.	Usuń zadanie	9
II.6.	Wyszukaj zadanie	9
II.7.	Cofnij wyszukiwanie	10
II.8.	Sortuj zadania	11
II.9.	Nieposortowane zadania	12
II.10.	Zadania na dzisiaj	12
II.11.	Dodaj opis zadania	13
II.12.	Pokaż opis	13
II.13.	Zapisywanie do plików	14
II.14.	Wykresy do statystyk	14
II.15.	Importuj dane z csv	15
II.16.	Główny kod	16
III.	Problemy podczas tworzenia aplikacji	18
IV.	Funkcjonalności aplikacji	26
IV.1.	Dodawanie zadań	26
IV.2.	Oznacz zadanie jako zrobione	30
IV.3.	Oznacz jako niezrobione	31
IV.4.	Edytuj nazwę zadania	32
IV.5.	Usuń zadanie	33
IV.6.	Wyszukaj zadanie	34

IV.7 Cofnij wyszukiwanie.....	35
IV.8. Sortuj zadania.....	36
IV.9. Nieposortowane zadania.....	40
IV.10. Zadania na dzisiaj.....	40
IV.11. Dodaj opis zadania	41
IV.12. Pokaż opis	43
IV.13. Zapisywanie do plików	43
IV.14 Wykresy do statystyk	47
IV.15.Importuj dane z csv	51
V. Podsumowanie oraz wnioski.....	52
V.1. Podsumowanie.....	52
V.2 Wnioski.....	53
V.3. Plan na możliwy rozwój w przyszłości	54

I. Wstęp

I.1. Github

Repozytorium w którym były wykonywane commity:

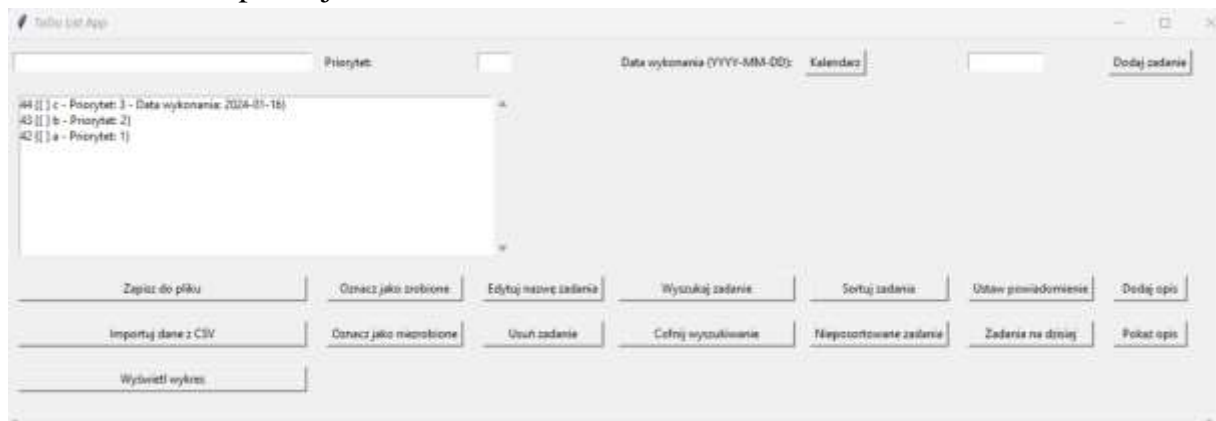
https://github.com/JuliaOlowniuk/PJF_Olowniuk

Od początku wykonywałam commity w branchu main, aczkolwiek w pewnym momencie przestała mi działać pewna funkcjonalność i musiałam przywrócić wcześniejszą wersję kilka commitów wstecz, więc powstał tak branch „cofnięcie” w którym działałam od momentu cofnięcia się aż do końca pracy nad aplikacją.

I.2 Ogólny opis aplikacji

ToDo List to aplikacja okienkowa napisana w języku Python, stworzona w celu ułatwienia organizacji codziennych zadań i planowania czasu. Pozwala użytkownikowi tworzyć listę zadań, określać priorytety, czy na przykład ustawiać terminy wykonania zadania.

Screen widoku aplikacji:



II. Opis implementacji/ technologii w funkcjach

II.1. Dodawanie zadań

Kod tej funkcji znajdujący się w pliku **add.py** to implementacja funkcji dodawania zadania do aplikacji do zarządzania listą zadań. Opis kodu:

1. Importowane Biblioteki:

- **sqlite3**: Do obsługi operacji na bazie danych SQLite.
- **tkinter**: Do tworzenia interfejsu graficznego.
- **simpdialog** i **messagebox** z **tkinter**: Do interakcji z użytkownikiem.
- **load_tasks** z modułu **load**: Funkcja do ładowania zadań z bazy danych.

2. Funkcja **add_task**:

- Parametry:
 - **conn**: Połączenie z bazą danych SQLite.
 - **task_entry_widget**: Widget wprowadzania tekstu dla treści zadania.
 - **task_listbox**: Listbox, w którym wyświetlane są zadania.
 - **priority_entry**: Widget wprowadzania tekstu dla priorytetu zadania.
 - **due_date_entry**: Widget wprowadzania tekstu dla daty wykonania zadania.
 - **dynamic_priority_value**: Opcjonalny dynamiczny priorytet.
 - **display_mode**: Tryb wyświetlania (1 dla domyślnego, 2 dla tygodniowego planera).
 - **weekday_combobox**: Combobox z dniem tygodnia (tylko w trybie tygodniowego planera).
- Funkcja pobiera treść zadania, priorytet i datę wykonania z widgetów wprowadzania tekstu.
- Sprawdza poprawność wprowadzonych danych, wyświetlając ostrzeżenia, jeśli coś jest nieprawidłowe.
- Próbuje skonwertować priorytet na liczbę całkowitą.
- Sprawdza bazę danych w celu istniejących zadań o tym samym priorytecie.
- Jeśli zadanie o podanym priorytecie już istnieje, prosi użytkownika o podanie nowego priorytetu.
- Dodaje nowe zadanie do bazy danych, uwzględniając dynamiczny priorytet, jeśli został dostarczony.

- Wyświetla zadanie w odpowiednim formacie w zależności od dostarczonych danych (treść, priorytet, data wykonania).
- Czyści widgety wprowadzania tekstu po dodaniu zadania.
- Wywołuje funkcję **load_tasks** do ponownego załadowania zadań do listboxa.
- Obsługuje błędy związane z bazą danych, wyświetlając komunikaty o błędach.

II.2. Oznacz zadanie jako zrobione

Kod tej funkcji znajdujący się w pliku **done.py** to implementacja funkcji zawiera funkcję **mark_done**, która odpowiada za oznaczanie zadań jako zakończone w aplikacji do zarządzania listą zadań. Opis kodu:

1. Importowane Biblioteki:

- **messagebox** z **tkinter**: Do wyświetlania komunikatów dla użytkownika.
- **load_tasks** z modułu **load**: Funkcja do ładowania zadań z bazy danych.

2. Funkcja mark_done:

- Parametry:
 - **conn**: Połączenie z bazą danych SQLite.
 - **task_listbox**: Listbox, w którym wyświetlane są zadania.
- Funkcja rozpoczyna od pobrania indeksów zaznaczonych zadań z **task_listbox**.
- Jeśli żadne zadanie nie jest zaznaczone, zostaje zgłoszony błąd i wyświetlony odpowiedni komunikat.
- Następnie dla każdego zaznaczonego zadania:
 - Pobierany jest identyfikator zadania (**task_id**).
 - Sprawdzane jest, czy zadanie już jest zakończone, jeśli tak, to zgłaszany jest błąd.
 - Jeśli zadanie nie jest zakończone, aktualizuje jego status na zakończone (ustawia na **True**) w bazie danych.
- Po zaktualizowaniu zadań, funkcja wywołuje **load_tasks** do ponownego załadowania zadań do listboxa.
- Obsługuje błędy, takie jak próba oznaczenia już zakończonego zadania, wyświetlając komunikat o błędzie.

II.3. Oznacz jako niezrobione

Kod tej funkcji znajdujący się w pliku **done.py** to implementacja funkcji zawiera funkcję **mark_undone**, która odpowiada za odznaczanie zadań jako niewykonane w aplikacji do zarządzania listą zadań. Opis kodu:

1. Importowane Biblioteki:

- **messagebox** z **tkinter**: Do wyświetlania komunikatów dla użytkownika.
- **load_tasks** z modułu **load**: Funkcja do ładowania zadań z bazy danych.

2. Funkcja **mark_undone**:

- Parametry:
 - **conn**: Połączenie z bazą danych SQLite.
 - **task_listbox**: Listbox, w którym wyświetlane są zadania.
- Funkcja rozpoczyna od pobrania indeksów zaznaczonych zadań z **task_listbox**.
- Jeśli żadne zadanie nie jest zaznaczone, zostaje zgłoszony błąd i wyświetlony odpowiedni komunikat.
- Następnie dla każdego zaznaczonego zadania:
 - Pobierany jest identyfikator zadania (**task_id**).
 - Sprawdzane jest, czy zadanie już jest oznaczone jako niezrobione, jeśli tak, to zgłaszany jest błąd.
 - Jeśli zadanie jest zakończone, aktualizuje jego status na niewykonane (ustawia na **False**) w bazie danych.
- Po zaktualizowaniu zadań, funkcja wywołuje **load_tasks** do ponownego załadowania zadań do listboxa.
- Obsługuje błędy, takie jak próba oznaczenia już niewykonanego zadania, wyświetlając komunikat o błędzie.

II.4. Edytuj nazwę zadania

Kod tej funkcji znajdujący się w pliku **editTask.py** to implementacja funkcji **edit_task_name**, która umożliwia edycję nazwy zadania w aplikacji do zarządzania listą zadań. Opis kodu:

1. Importowane Biblioteki:

- **tkinter**: Do tworzenia interfejsu graficznego.
- **simpledialog** i **messagebox** z **tkinter**: Do interakcji z użytkownikiem.
- **load_tasks** z modułu **load**: Funkcja do ładowania zadań z bazy danych.

2. Funkcja **edit_task_name**:

- Parametry:
 - **conn**: Połączenie z bazą danych SQLite.
 - **task_listbox**: Listbox, w którym wyświetlane są zadania.
- Funkcja rozpoczyna od sprawdzenia, czy jakiegokolwiek zadanie zostało zaznaczone do edycji.
- Jeśli żadne zadanie nie jest zaznaczone, wyświetla komunikat błędu.
- Pobiera identyfikator zaznaczonego zadania (**task_id**).
- Pobiera obecną nazwę zadania z bazy danych.
- Wykorzystuje **simpledialog.askstring** do uzyskania nowej nazwy zadania od użytkownika.

- Jeśli użytkownik podał nową nazwę i różni się od obecnej, aktualizuje nazwę zadania w bazie danych.
- Po aktualizacji zadania, ponownie ładuje zadania za pomocą funkcji **load_tasks**.
- Wyświetla stosowne komunikaty informacyjne, zależnie od wyniku edycji (sukces lub brak zmian).

II.5. Usuń zadanie

Kod tej funkcji znajdujący się w pliku **delete.py**, `delete_task`, która obsługuje usunięcie zadania z aplikacji do zarządzania listą zadań.. Opis kodu:

1. Importowane Biblioteki:

tkinter: Do tworzenia interfejsu graficznego.

messagebox z tkinter: Do wyświetlania komunikatów dla użytkownika.

load_tasks z modułu load: Funkcja do ładowania zadań z bazy danych.

2. Funkcja `delete_task`:

- Parametry:
- `conn`: Połączenie z bazą danych SQLite.
- **`task_listbox`**: Listbox, w którym wyświetlane są zadania.
- Funkcja rozpoczyna od sprawdzenia, czy jakiegokolwiek zadanie zostało zaznaczone do usunięcia.
- Jeśli żadne zadanie nie jest zaznaczone, wyświetla komunikat ostrzegawczy.
- Pobiera indeks zaznaczonego zadania oraz jego identyfikator (**`task_id`**) i priorytet (**`deleted_priority`**).
- Wyświetla okno dialogowe (**`messagebox.askyesno`**) w celu potwierdzenia chęci usunięcia zadania.
- Jeśli użytkownik potwierdzi, usuwa zadanie z bazy danych.
- Funkcja jest przzerwana tutaj, ale w pełnej wersji kodu mogłaby zawierać dodatkowe działania, takie jak aktualizacja priorytetów po usunięciu zadania.

II.6. Wyszukaj zadanie

W pliku **search.py** znajduje się kilka funkcji. Główną funkcją jest funkcja **`search_task`**, która obsługuje wyszukiwanie zadań w aplikacji do zarządzania listą zadań. Oto opis:

1. Użyte Technologie i Biblioteki:

- **Tkinter:** Biblioteka do tworzenia interfejsu graficznego w Pythonie.
- **datetime:** Moduł do obsługi operacji na datach i czasie.

2. Funkcja `search_task`:

- Parametry:

- **conn:** Połączenie z bazą danych SQLite.
- **task_listbox:** Listbox, w którym wyświetlane są zadania.
- Funkcja rozpoczyna od wyświetlenia okna dialogowego, w którym użytkownik wprowadza frazę do wyszukania (**simpledialog.askstring**).
- Jeśli fraza została podana, przekształca ją na małe litery i zapisuje oryginalne zadania przed rozpoczęciem wyszukiwania.
- Następnie próbuje znaleźć pasujące zadania za pomocą funkcji **find_matching_tasks**.
- Jeśli wyszukiwanie powiedzie się, wyniki są wyświetlane w **task_listbox** za pomocą funkcji **display_search_results**.
- W przypadku błędu, funkcja wywołuje **show_error_message** do wyświetlenia komunikatu o błędzie.

3. Funkcje Dodatkowe:

- **show_error_message:** Wyświetla okno dialogowe z komunikatem błędu.
- **load_original_tasks:** Pobiera oryginalne zadania przed wyszukiwaniem.
- **undo_search:** Cofa wyszukiwanie i przywraca oryginalne zadania.
- **find_matching_tasks:** Znajduje pasujące zadania na podstawie frazy wyszukiwania.
- **display_search_results:** Wyświetla wyniki wyszukiwania w **task_listbox**.
- **format_due_date:** Formatuje datę zgodnie z określonym wzorcem.

II.7. Cofnij wyszukiwanie

W pliku `search.py` z funkcji dodatkowych na szczególną uwagę zasługuje funkcja `undo_search` ma na celu cofnięcie wyników poprzedniego wyszukiwania i przywrócenie oryginalnej listy zadań przed wyszukiwaniem. Oto bardziej szczegółowy opis tej funkcji:

1. Parametry:

conn: Połączenie z bazą danych SQLite.

task_listbox: Listbox, w którym wyświetlane są zadania.

2. Zmienne Globalne:

oryginalne_zadania: Zmienna globalna przechowująca oryginalne zadania przed wyszukiwaniem.

poprzednie_wyszukiwanie_powiodlo_sie: Zmienna globalna informująca, czy poprzednie wyszukiwanie było udane.

3. Logika Funkcji:

- Sprawdzane jest, czy poprzednie wyszukiwanie powiodło się (**poprzednie_wyszukiwanie_powiodlo_sie**) oraz czy istnieją oryginalne zadania przed wyszukiwaniem (**oryginalne_zadania**).

- Jeśli tak, to funkcja wywołuje **display_search_results** w celu przywrócenia oryginalnych zadań do **task_listbox**, czyli listy zadań wyświetlanej w interfejsie użytkownika. Następnie zeruje **oryginalne_zadania** przy pomocy `oryginalne_zadania = []`, aby wyczyścić oryginalne zadania po cofnięciu wyszukiwania.
- Jeśli poprzednie wyszukiwanie nie powiodło się, funkcja wywołuje **show_error_message** z informacją o błędzie.
- Jeśli nie było poprzedniego wyszukiwania, również wywołuje **show_error_message**, ale z informacją, że nie było poprzedniego wyszukiwania.

II.8. Sortuj zadania

W pliku **sort.py** znajdują się funkcje, które pozwalają na sortowanie zadań w listboxie. Oto ich opis:

1. Funkcja **sort_tasks**:

Wyświetla okno dialogowe, w którym użytkownik może wybrać kryterium sortowania. Na podstawie wyboru użytkownika wywołuje jedną z funkcji sortujących.

2. Funkcje Sortujące:

- **sort_by_priority_asc** i **sort_by_priority_desc**:

Sortują zadania według priorytetu rosnąco/malejąco.

Dzieli zadania na te z datą wykonania i bez niej, sortując je odpowiednio.

- Wywołują funkcję **display_sorted_tasks** do wyświetlenia posortowanych zadań.
- **sort_by_due_date_asc** i **sort_by_due_date_desc**:
- Sortują zadania według daty wykonania rosnąco/malejąco.
- Wykorzystują **datetime(due_date)** do sortowania według daty.
- Wywołują funkcję **display_sorted_tasks** do wyświetlenia posortowanych zadań.

- **Funkcja **display_sorted_tasks**:**

Wyświetla posortowane zadania w **task_listbox**.

Dla każdego zadania tworzy tekst opisujący zadanie (tekst zgodny z formatem) i dodaje je do **task_listbox**.

- **Funkcja **format_due_date**:**

Formatuje datę zgodnie z określonym wzorcem ("%d-%m-%Y").

W przypadku problemów z formatowaniem zwraca oryginalną datę.

- **Funkcja **display_unsorted_tasks**:**

Wyświetla wszystkie zadania w oryginalnej kolejności.

Pobiera zadania z bazy danych i wywołuje funkcję **display_sorted_tasks** do ich wyświetlenia

II.9. Nieposortowane zadania

Funkcja **display_unsorted_tasks** pełni rolę w interfejsie użytkownika, prezentując nieposortowane zadania w widżecie **task_listbox**. Znajduje się w pliku **sort.py**. Opis kodu:

1. Utworzenie kursora:

- Funkcja inicjalizuje kursor, umożliwiając operacje na bazie danych SQLite.

2. Pobranie zadań z bazy danych:

- Przy użyciu kursora realizowane jest zapytanie SQL w celu pobrania wszystkich zadań z tabeli **tasks**.

3. Wyczyszczenie listboxa:

- W celu uniknięcia duplikacji, funkcja usuwa wszelkie istniejące wpisy z **task_listbox** przed dodaniem nowych zadań.

4. Iteracja przez zadania i dodanie do listboxa:

- Przy użyciu pętli **for** funkcja przegląda każde zadanie pobrane z bazy danych.
- Dla każdego zadania tworzony jest tekstowy opis, zawierający informacje o treści zadania, priorytecie oraz, opcjonalnie, o dacie wykonania, jeśli jest dostępna.

5. Dodanie zadania do listboxa:

- Utworzony tekstowy opis zadania jest wstawiany do widżetu **task_listbox** za pomocą metody **insert**.

Zastosowane technologie i biblioteki:

- **SQLite:** Wykorzystane do obsługi bazy danych SQLite.
- **Tkinter:** Stosowane do kreowania interfejsu użytkownika i zarządzania elementami graficznymi, np. **Listbox**.

II.10. Zadania na dzisiaj

Funkcja **show_tasks_for_today** odpowiada za wyświetlenie informacji o zadaniach zaplanowanych na dzisiaj. Wykorzystuje moduł **datetime** do pobrania aktualnej daty. Następnie korzysta z funkcji **get_tasks_for_date** w celu uzyskania listy zadań zaplanowanych na dzisiaj.

Po pobraniu zadań, funkcja sprawdza, czy istnieją jakiekolwiek zadania na dzisiaj. Jeśli nie, wyświetla okno informacyjne z komunikatem o braku zaplanowanych zadań. W przeciwnym razie, tworzy tekstowy opis zadań na dzisiaj i prezentuje go w formie okna informacyjnego. Znajduje się w pliku **notification.py**

Zastosowane technologie i biblioteki:

- **datetime:** Wykorzystywane do operacji związanych z datami.
- **tkinter.messagebox:** Umożliwia wyświetlanie prostych okien informacyjnych w interfejsie graficznym.

II.11. Dodaj opis zadania

Funkcja **add_description** jest odpowiedzialna za dodawanie opisu do zadań. Znajduje się w pliku **description.py**

Dodawanie Opisu (add_description):

- Funkcja umożliwia dodanie opisu do wybranego zadania.
- Najpierw sprawdza, czy jakiegokolwiek zadanie zostało wybrane na liście.
- Następnie pobiera identyfikator wybranego zadania i bieżący opis tego zadania.
- Korzystając z **simplifiedialog.askstring**, pozwala użytkownikowi wprowadzić nowy opis.
- Jeżeli nowy opis istnieje i różni się od bieżącego, aktualizuje opis w bazie danych.
- Informuje użytkownika o sukcesie lub braku zmiany w opisie.

Zastosowane technologie i biblioteki:

- **tkinter:** Biblioteka do tworzenia interfejsu graficznego w Pythonie.
- **tkinter.simplifiedialog:** Umożliwia proste interakcje z użytkownikiem, takie jak pobieranie wartości wejściowych.
- **tkinter.messagebox:** Służy do wyświetlania okien informacyjnych w interfejsie graficznym.

II.12. Pokaż opis

Funkcja **show_description** jest odpowiedzialna za dodawanie opisu do zadań. Znajduje się w pliku **description.py**

Wyświetlanie Opisu (show_description):

- Funkcja prezentuje aktualny opis wybranego zadania.
- Sprawdza, czy jakiegokolwiek zadanie zostało wybrane na liście.
- Pobiera identyfikator wybranego zadania i bieżący opis tego zadania z bazy danych.
- Jeżeli opis istnieje, wyświetla go w oknie informacyjnym. W przeciwnym razie, informuje o braku opisu.

Zastosowane technologie i biblioteki:

- **tkinter:** Biblioteka do tworzenia interfejsu graficznego w Pythonie.
- **tkinter.simplifiedialog:** Umożliwia proste interakcje z użytkownikiem, takie jak pobieranie wartości wejściowych.

- **tkinter.messagebox:** Służy do wyświetlania okien informacyjnych w interfejsie graficznym.

II.13. Zapisywanie do plików

Funkcja **save_tasks_to_file** odpowiada za zapisywanie zadań do pliku w różnych formatach (txt, xlsx, docx). Znajdują się w pliku **saveTaskToFile.py**

Zapis do Pliku (save_tasks_to_file):

- Funkcja umożliwia zapisanie zadań do pliku w wybranym formacie (txt, xlsx, docx).
- Na początku pobiera wszystkie zadania z bazy danych, uporządkowane według priorytetu.
- Następnie w zależności od wybranego formatu, tworzy plik i zapisuje w nim zadania.
- W przypadku pliku tekstowego (txt), tworzy napisy z danymi zadań i zapisuje je w pliku.
- Dla plików xlsx i docx korzysta z bibliotek pandas, openpyxl i python-docx, aby utworzyć odpowiednie dokumenty.
- Na końcu wyświetla komunikat informujący o sukcesie lub ostrzeżenie o nieobsługiwanym formacie pliku.

Zastosowane technologie i biblioteki:

- **sqlite3:** Moduł do obsługi bazy danych SQLite w Pythonie.
- **pandas:** Biblioteka do manipulacji danymi, wykorzystywana do konwersji danych do formatu DataFrame.
- **docx (python-docx):** Biblioteka do tworzenia i edycji plików DOCX (Microsoft Word).
- **openpyxl:** Biblioteka do obsługi plików XLSX (Microsoft Excel).
- **tkinter.messagebox:** Moduł z tkinter do wyświetlania okienek komunikatów.

II.14. Wykresy do statystyk

Funkcja **display_charts** jest odpowiedzialna za wyświetlanie wykresów na podstawie danych o zadaniach. Znajduje się w pliku **chart.py**.

1. Wyświetlanie Wykresów (display_charts):

- Funkcja pozwala użytkownikowi na wybór rodzaju wykresu (kołowy lub słupkowy).
- Pobiera dane o zadaniach z listboxa przy użyciu funkcji **refresh_task_listbox**.
- Przetwarza dane, a następnie tworzy i wyświetla wybrany rodzaj wykresu.

2. Rodzaje Wykresów:

- **Wykres Kołowy:**
 - Przykład: Zlicza zadania wykonane i niewykonane.
 - Tworzy dane w postaci słownika, a następnie tworzy DataFrame.
 - Rysuje wykres kołowy na podstawie danych i wyświetla go w oknie.
 - **Wykres Słupkowy:**
 - Przykład: Zlicza zadania z datą i bez daty.
 - Tworzy dane w postaci słownika, a następnie tworzy DataFrame.
 - Rysuje wykres słupkowy na podstawie danych i wyświetla go w oddzielnym oknie.
3. **Zastosowane Technologie i Biblioteki:**
- **pandas:** Biblioteka do manipulacji danymi, wykorzystywana do przetwarzania danych do formatu DataFrame.
 - **tkinter:** Biblioteka do tworzenia interfejsu graficznego, używana do okienek dialogowych.
 - **matplotlib.pyplot:** Biblioteka do tworzenia wykresów i wizualizacji danych.
 - **load.refresh_task_listbox:** Funkcja z pliku **load**, używana do odświeżania listy zadań w listboxie.
4. **Obsługa Błędów:**
- Funkcja obsługuje błędy podczas generowania i wyświetlania wykresów.
 - Wyświetla komunikaty błędów w przypadku problemów z generowaniem wykresów.

Uwagi:

- Funkcja korzysta z okna topowego (**tk.Toplevel()**) do wyświetlenia wykresu słupkowego.
- Zastosowano funkcję **plt.show(block=False)**, aby nie blokować interfejsu użytkownika podczas wyświetlania wykresów.

II.15. Importuj dane z csv

Funkcja **import_from_csv** umożliwia importowanie danych z pliku CSV do bazy danych SQLite, a następnie odświeża listę zadań w interfejsie graficznym. Znajduje się w pliku **importCSV.py**

1. **Importowanie Zadań z Pliku CSV (import_from_csv):**
 - Umożliwia użytkownikowi wybór pliku CSV do importu.
 - Otwiera wybrany plik CSV, pomijając nagłówek, a następnie przetwarza i zapisuje dane do bazy danych SQLite.
2. **Odczytywanie i Przetwarzanie Danych CSV:**

- Otwiera plik CSV i używa czytnika CSV do odczytania danych.
 - Przetwarza każdy wiersz, dzieląc wartości za pomocą przecinka i usuwając cudzysłowy.
 - Sprawdza, czy wiersz zawiera pięć wartości, a następnie konwertuje je do odpowiednich typów (np., bool, int, datetime).
3. **Zapisywanie do Bazy Danych SQLite:**
- Korzysta z połączenia SQLite i kursora, aby wykonywać operacje na bazie danych.
 - Wstawia dane do tabeli **tasks**, sprawdzając poprawność formatu i konwertując wartości.
4. **Obsługa Błędów:**
- Funkcja obsługuje błędy SQLite i ogólne błędy podczas importowania danych.
 - Wyświetla komunikaty błędów w przypadku problemów podczas importu.
5. **Odświeżanie Listy Zadań:**
- Po zakończeniu importu, używa funkcji **load_tasks** z modułu **load** do ponownego załadowania zadań z bazy danych i odświeżenia listy w interfejsie graficznym.
6. **Uwagi:**
- Obsługuje błędy związane z formatem danych w pliku CSV, błędami SQLite oraz nieprawidłowym formatem daty.
 - Informuje użytkownika o sukcesie lub problemach związanych z importem danych.

II.16. Główny kod

Klasa **ToDoListApp** jest centralną częścią aplikacji ToDo List. Poniżej znajduje się opis tej klasy :

1. **Inicjalizacja Aplikacji (initialize):**
 - Po uruchomieniu aplikacji, tworzy główne okno i nawiązuje połączenie z bazą danych SQLite (**todolist.db**).
 - Tworzy interfejs graficzny, który obejmuje ramkę, paski przewijania, pole tekstowe, przyciski i inne elementy.
2. **Dostosowanie Rozmiaru Okna i Czcionki (adjust_widgets i adjust_font_size):**
 - Odpowiada za dostosowanie rozmiaru okna oraz czcionki w zależności od zmiany rozmiaru głównego okna.

- Przykładowo, elementy takie jak przyciski, etykiety i pole tekstowe są dostosowywane proporcjonalnie.
3. **Tworzenie Elementów Interfejsu (create_widgets):**
 - Tworzy wszystkie elementy interfejsu, takie jak pole tekstowe, przyciski, etykiety, lista zadań itp.
 - Przypisuje odpowiednie funkcje do przycisków, np. dodawanie zadania, usuwanie zadania, wyświetlanie wykresu itp.
 4. **Obsługa Przycisków i Funkcji (add_task_with_dynamic_priority, import_data_from_csv, delete_task, itp.):**
 - Zawiera funkcje obsługujące interakcje użytkownika, np. dodawanie zadania z dynamicznym priorytetem, importowanie danych z pliku CSV, usuwanie zadania itp.
 5. **Obsługa Kalendarza (open_calendar):**
 - Otwiera nowe okno dialogowe z kalendarzem (wykorzystując bibliotekę tkcalendar), pozwalając użytkownikowi wybrać datę.
 6. **Wczytywanie Zadań (load_tasks):**
 - Wywołuje funkcję `load_tasks` z modułu `load`, aby wczytać zadania z bazy danych i wyświetlić je w liście zadań na interfejsie graficznym.
 7. **Zapisywanie Zadań do Pliku (save_tasks_to_file):**
 - Otwiera okno dialogowe do wyboru pliku i umożliwia użytkownikowi zapisanie zadań do pliku w różnych formatach (txt, xlsx, docx).
 8. **Uruchomienie Aplikacji (root.mainloop()):**
 - Inicjalizuje główną pętlę zdarzeń aplikacji Tkinter, umożliwiając interakcję użytkownika z interfejsem.
 9. **Ustawienia Okna Głównego (root.geometry, root.resizable):**
 - Ustala domyślny rozmiar okna głównego i określa, czy można zmieniać jego rozmiar.
 10. **Obsługa Bazy Danych (conn):**
 - Inicjalizuje połączenie z bazą danych SQLite i tworzy niezbędne tabele za pomocą funkcji `create_tables` z modułu `todolist_db`.
 11. **Obsługa Responsywności i Paska Przewijania (on_frame_configure):**
 - Zapewnia responsywność dla elementów interfejsu oraz obsługuje pasek przewijania w przypadku, gdy lista zadań jest dłuższa niż dostępna przestrzeń.
 12. **Ostatnia Opcja Wstępnego Konfigurowania Aplikacji (if tk.TkVersion >= 8.6):**
 - Sprawdza wersję Tkinter i inicjalizuje główne okno (`tk.Tk()` lub `tk.Tk(className="ToDo List App")`).
 13. **Zamknięcie Aplikacji (root.mainloop()):**

- Uruchamia główną pętlę zdarzeń, co pozwala na interakcję z użytkownikiem i obsługę zdarzeń w aplikacji.

III. Problemy podczas tworzenia aplikacji

III.1.1 Problem z bazą danych/ przypisywaniem zadań do użytkownika

Dodawanie zadań do listboxa opierało się na dodawaniu zadań do wbudowanej bazy danych, do tabeli tasks z kolumnami umieszczonymi na poniższym screenie:

Tables		
tasks		
id	INTEGER	"id" INTEGER
task	TEXT	"task" TEXT NOT NULL
done	BOOLEAN	"done" BOOLEAN NOT NULL
priority	INTEGER	"priority" INTEGER DEFAULT 0
due_date	TEXT	"due_date" TEXT
note	TEXT	"note" TEXT

Jednakże, gdy dodano tabelę users do tej bazy danych

users		
id	INTEGER	"id" INTEGER
username	TEXT	"username" TEXT NOT NULL
password_hash	TEXT	"password_hash" TEXT NOT NULL

Pojawił się problem z połączeniem obu tabel i mimo to, iż w kodzie w tworzeniu tabeli dodano klucz obcy drugiej, na której podstawie table miały się łączyć, to połączenie nie zadziałało.

```
FOREIGN KEY (user_id) REFERENCES users (id)
```

Przez to, niestety dla użytkowników nie był możliwy fakt przypisania do ich ID zadań z listboxa.

Na poniższym screenie widzimy jak zalogowany użytkownik iga, chce dodać zadanie do listbox i widnieje problem, że nie istnieje kolumna user_id, mimo, że przy tworzeniu tabeli została dodana kolumna z id użytkownika.

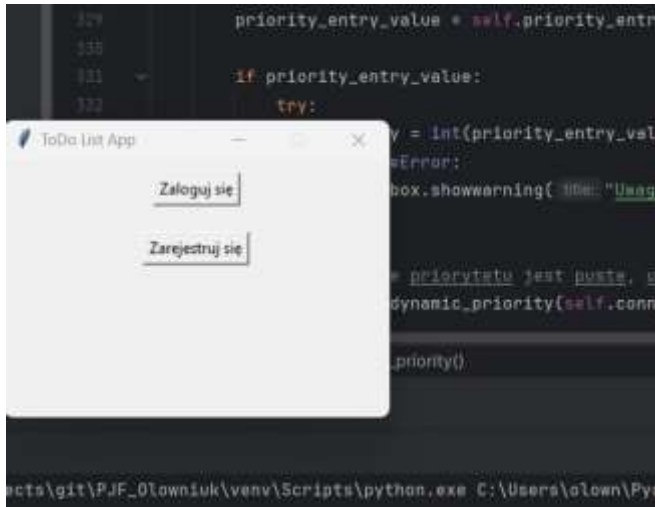
```
Task: iga, Priority: 1, Due Date: 2024-01-16
Executing SQL: INSERT INTO tasks (user_id, task_name, done, priority, due_date) VALUES (?, ?, ?, ?, ?) (None, 'iga', False, 1, '2024-01-16')
Błąd SQLite przy dodawaniu zadania do bazy danych: table tasks has no column named user_id
```

Tabele, pomimo próby „Drop the table”, czyli usunięcia i ponownego zaimplementowania niestety nie aktualizowały swojej zawartości, przez co zrezygnowano z logowania, które działało.

III.1.2 Zrezygnowano z logowania/ rejestrowania, pomimo jego działania

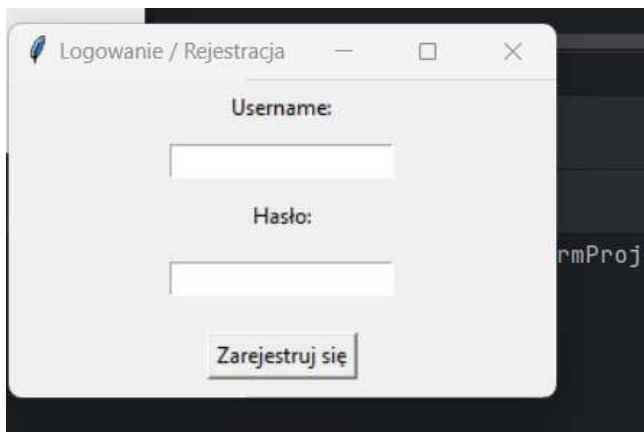
Poniżej przedstawiam jak działało logowanie w aplikacji, ale niestety ze względu na punkt III.1.2 z niego zrezygnowano, ze względu na to, iż zadania do zalogowanych użytkowników nie przypisywały się. Problem wynikający z bazy danych.

Po uruchomieniu aplikacji wyświetlało się okienko:

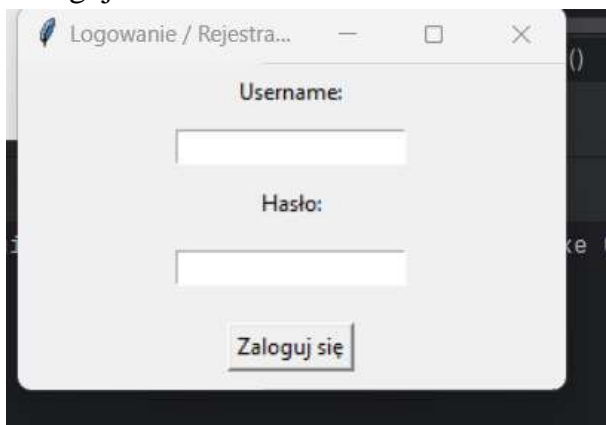


Następnie można było wybrać opcje zaloguj się lub zarejestruj się:


Zarejestruj:



Zaloguj:



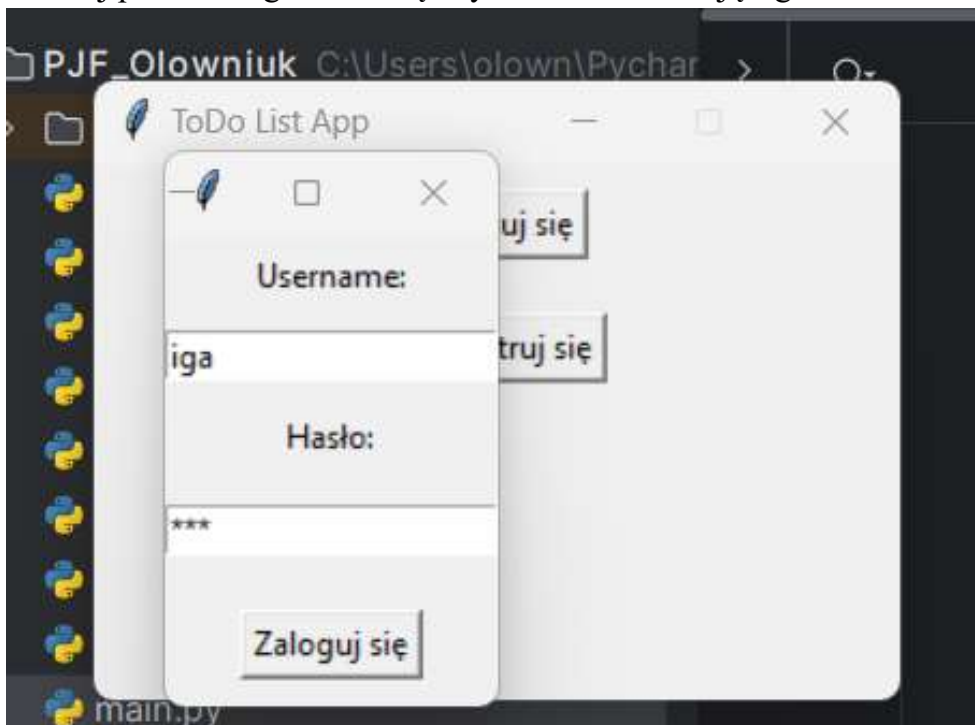
Spójrzmy na zawartość bazy danych przed zarejestrowaniem nowego użytkownika:



id	userna...	passwo...
1	aga	123
2	jula	123
3	kuba	123
4	iga	BLOB
5	igus	BLOB
6	katarzynka	BLOB

Niektóre hasła są widoczne- ponieważ dodano tych użytkowników do tabeli users przed dodaniem funkcji hashującej hasła w bazie danych.

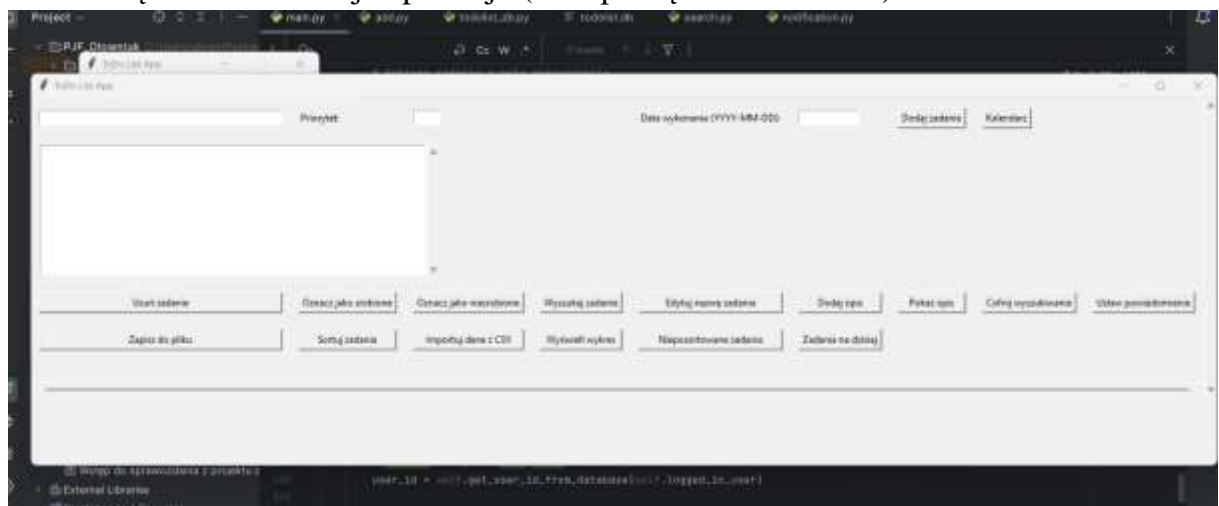
Poniżej próba zalogowania się użytkownika widniejącego w bazie danych:



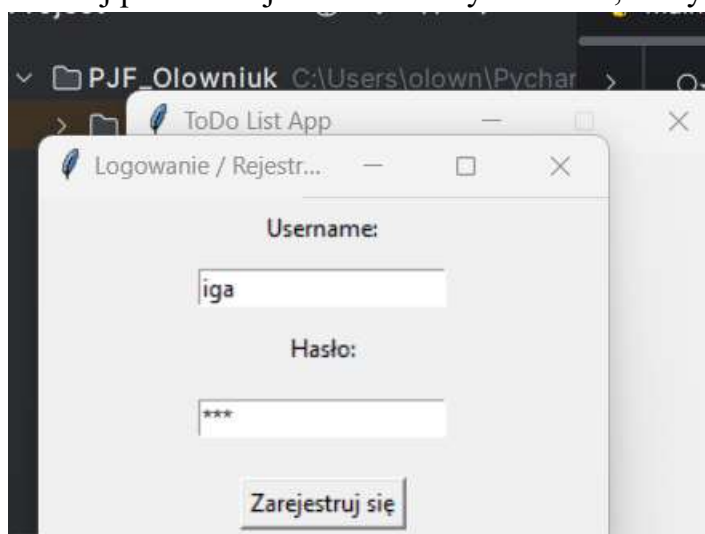
Użytkownik w bazie- logowanie udane:



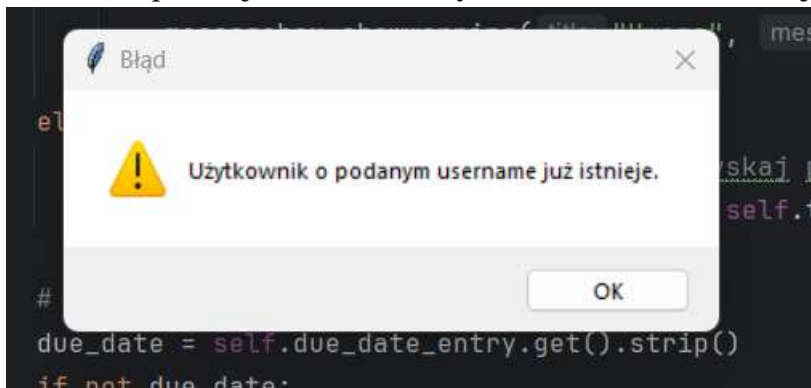
Po kliknięciu OK widnieje aplikacja (nieuporządkowane GUI):



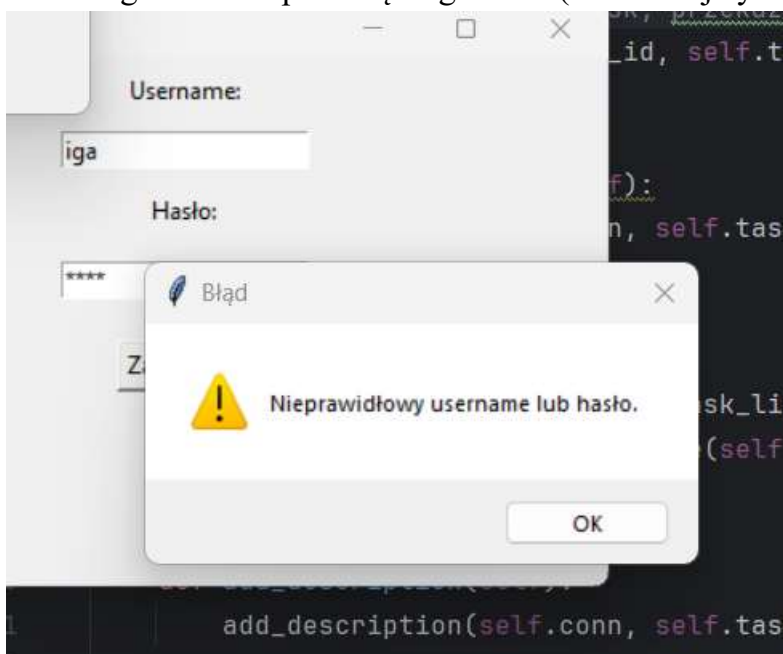
Poniżej próba zarejestrowania użytkownika, który widnieje w bazie:



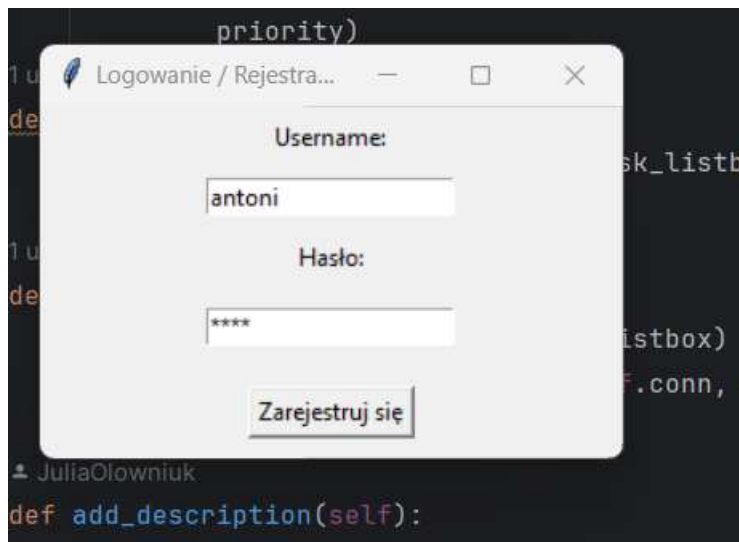
Jak widać poniżej, nie można użytkownika dodać, bo znajduje się już w bazie:



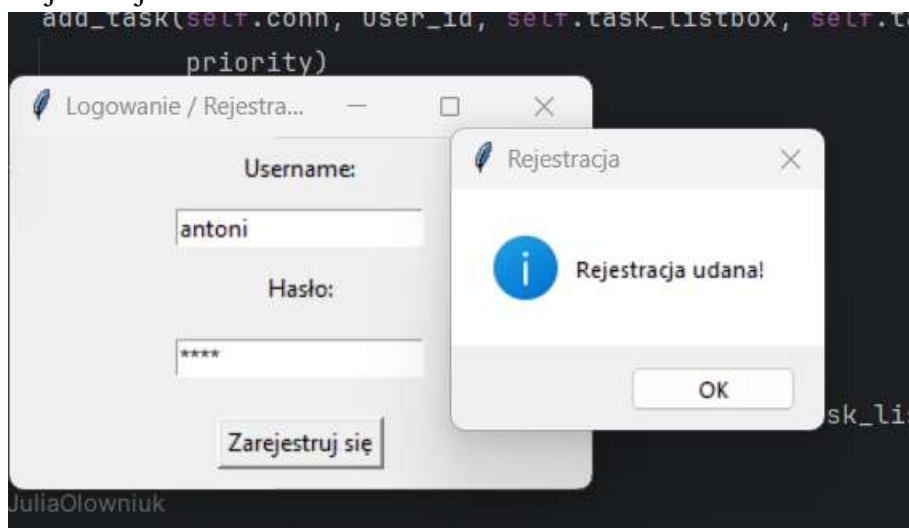
Próba logowania za pomocą złego hasła (wcześniej było 3 znakowe teraz 4-ro znakowe)



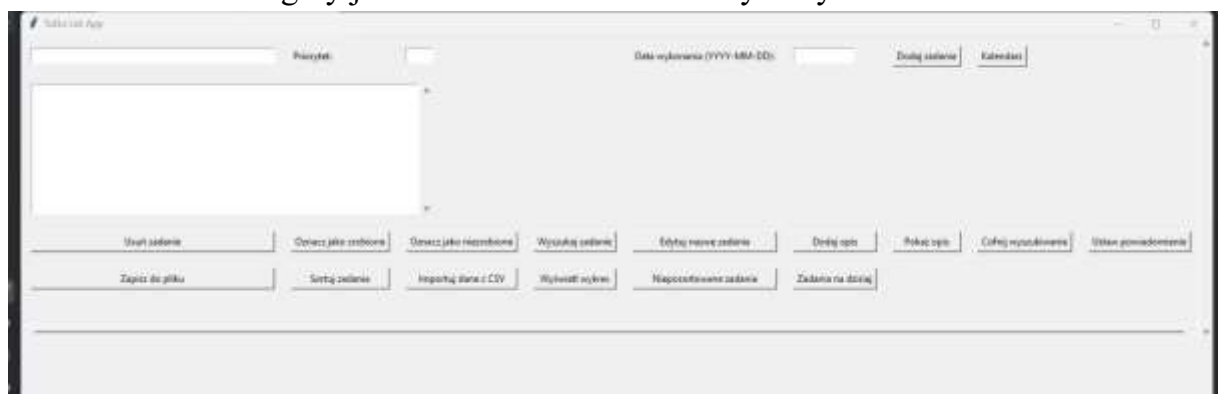
Zarejestrowanie nowego użytkownika i sprawdzenie czy został dodany do bazy:



Rejestracja udana:



Tak i oto antoni mógłby już dodawać zadania do bazy danych:



Sprawdzając czy widnieje w tabeli po wciśnięciu Refresh:

id	username	password
1	aga	123
2	jula	123
3	kuba	123
4	iga	BLOB
5	igus	BLOB
6	katarzynka	BLOB
7	antoni	BLOB

Wniosek: Gdyby baza danych działała poprawnie, czyli dwie tabele byłyby poprawnie ze sobą połączone, można byłoby prowadzić listy zadań dla użytkowników. Aby aplikacja działała poprawnie i można byłoby pokazać funkcjonalności cofnięto do poprzedniej wersji, bez logowania.

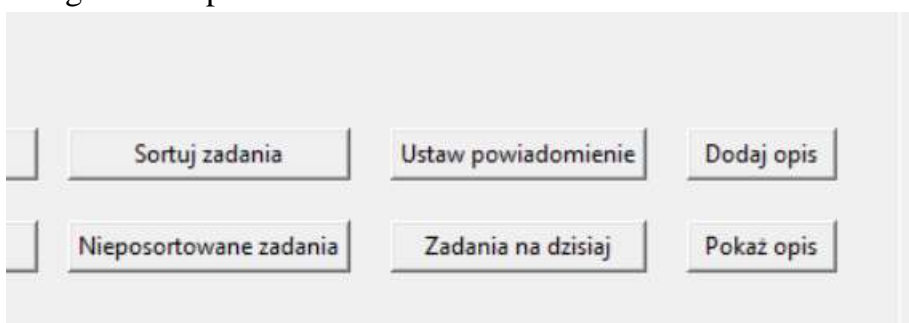
III.1.3 Zrezygnowanie z przełączania trybu na weekly planner

Zrezygnowano z przełączania trybu na weekly planner z listboxa z podobnego względu, iż zadania nie chciały dodawać się do dni, był błąd bazy danych. Co za tym idzie zrezygnowano z opcji które miały być wykorzystywane w weekly planerze: auto- pomocnik, kopiowanie z innych dni i autouzupełnianie innych dni.

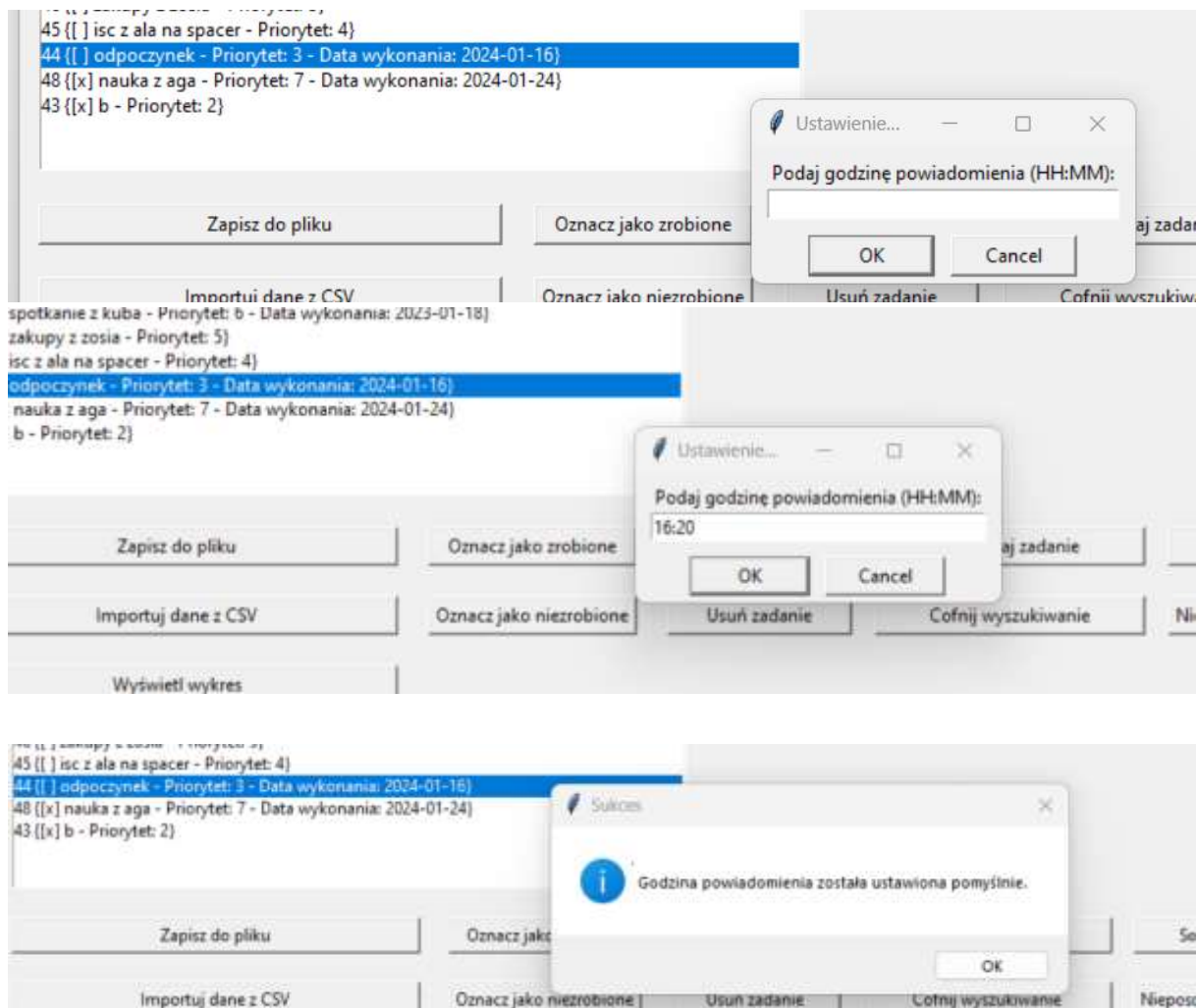
III.1.4 Niedziałające powiadomienia push na pulpit

Niestety nie mogłam włączyć powiadomień systemowych dla aplikacji PyCharm w ustawieniach. PyCharm nie widnieje w aplikacjach, w których można włączyć powiadomienia u mnie na komputerze. W aplikacji włączono notifications, Dodano funkcję notifications i widget umieszczony poniżej który po wybraniu zadania i kliknięciu na niego- ukazuje się okienko w które należy wpisać godzinę, o której powiadomienie ma przyjść.

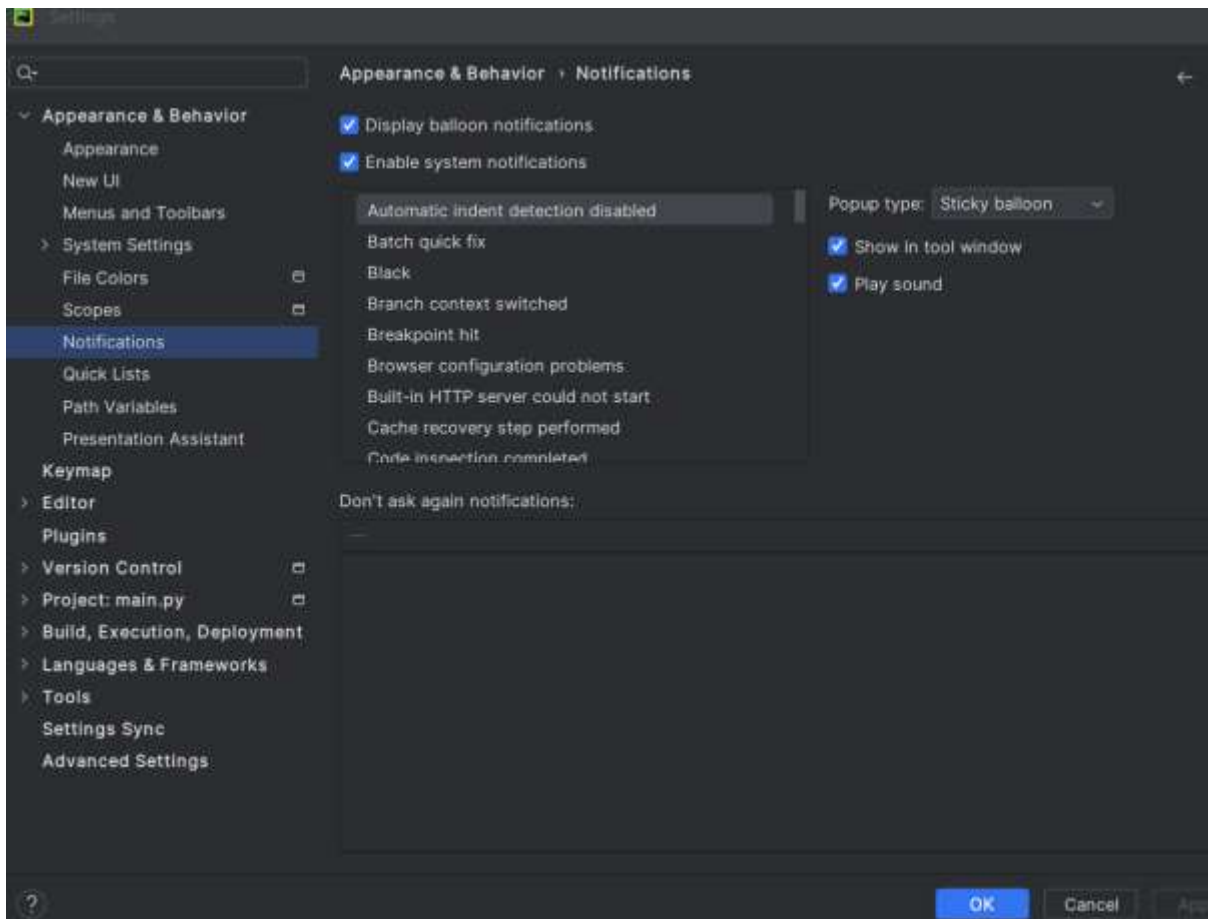
Widget ustaw powiadomienie:



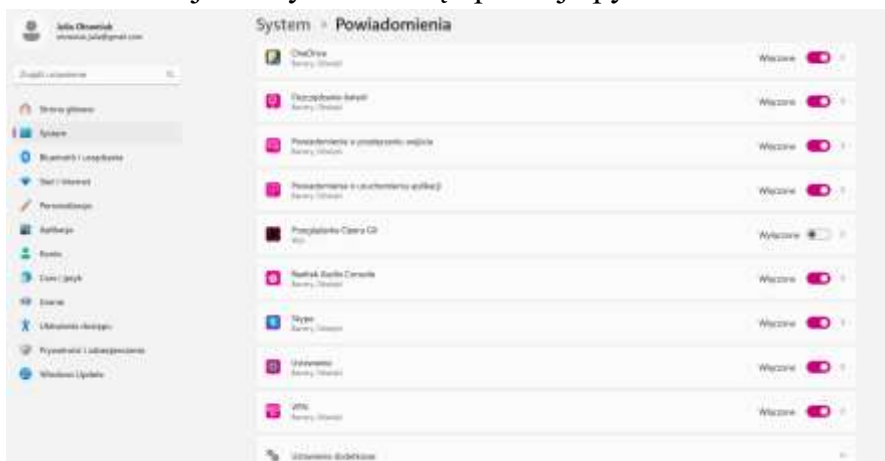
Po zaznaczeniu zadania i kliknięciu w przycisk:



Mimo takiego komunikatu, powiadomienie nie pojawia się.
Powiadomienia włączone w aplikacji:



Jednakże tutaj nie wyświetla się aplikacja pycharm:



Myślę, że to może być powodem tego, że powiadomienia się nie pojawiają

IV. Funkcjonalności aplikacji

W funkcjonalności aplikacji opiszę jakie ma funkcje przedstawię oraz przedstawię obsługę błędów.

IV.1. Dodawanie zadań

IV.1.1 Dodano zadanie wpisując jego treść nadając mu priorytet, bez daty, ponieważ nie jest konieczna do tego aby zadanie zostało dodane

The screenshot shows the 'Todo List App' interface. At the top, there are input fields for 'Task description', 'Priority' (set to 4), and 'Due date' (with a calendar icon). A 'Dodaj zadanie' button is on the right. Below the inputs is a list of tasks: '45 [] | nic z ala na spacer - Priorytet: 4', '44 [] | c - Priorytet: 3 - Data wykonania: 2024-01-16', '43 [] | b - Priorytet: 2', and '42 [] | a - Priorytet: 1'. At the bottom, there are several buttons: 'Zapisz do pliku', 'Oznacz jako wykonane', 'Edytuj nazwę zadania', 'Wyszukaj zadanie', 'Sortuj zadania', 'Ustaw powiadomienie', 'Dodaj opis', 'Importuj dane z CSV', 'Oznacz jako nieukończony', 'Usuń zadanie', 'Cofnij wyszukiwanie', 'Nieposortowane zadania', 'Zadania na dzisiaj', 'Pokaż opis', and 'Wyświetl wykres'.

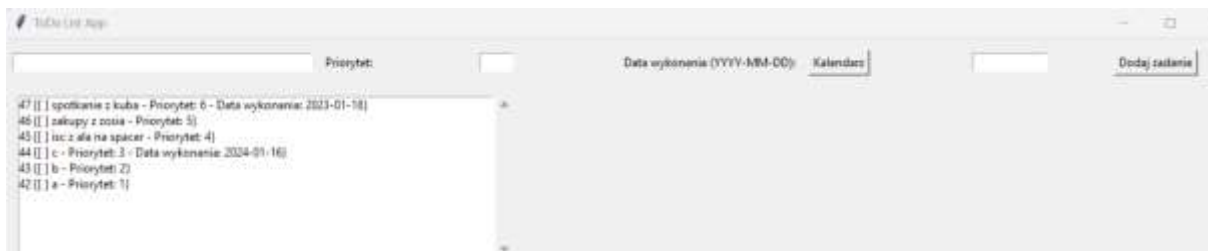
IV.1.2 Dodano zadanie wpisując jego treść nie dodając ani daty ani priorytetu, priorytet automatycznie zostanie dodany po dodaniu zadania do listy. Priorytet który będzie dodany, to taki który jest kolejny dostępny.

This screenshot is identical to the previous one, showing the 'Todo List App' with the same task list and interface elements.

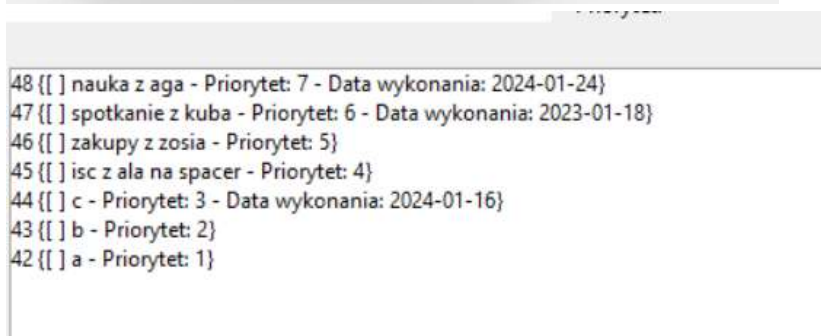
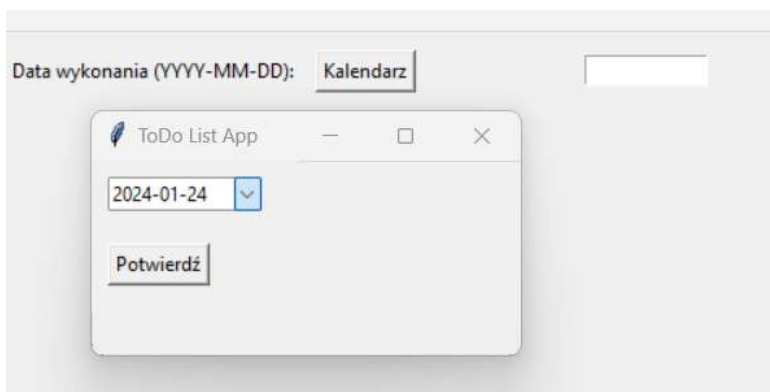
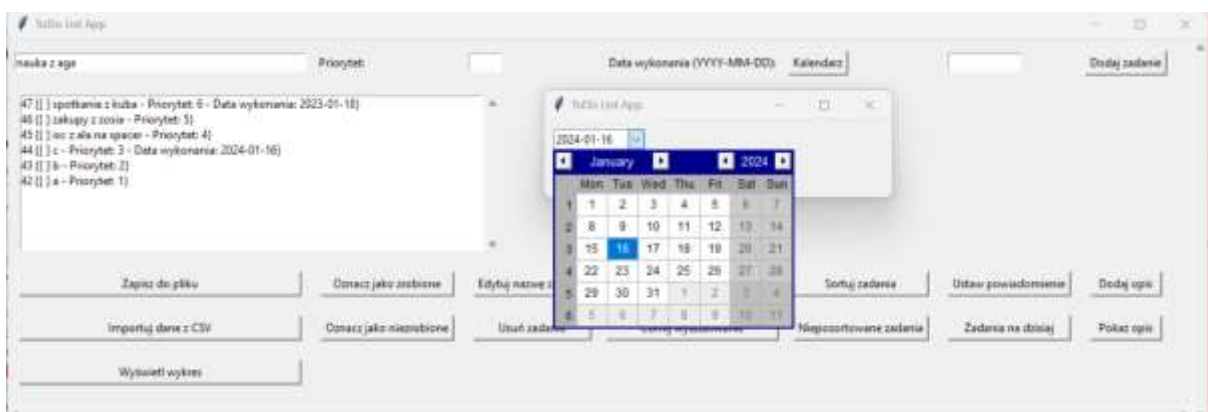
This screenshot is identical to the previous ones, showing the 'Todo List App' with the same task list and interface elements.

IV.1.3 Dodano zadanie wpisując jego treść nie dodając, priorytet automatycznie zostanie dodany po dodaniu zadania do listy. Priorytet który będzie dodany, to taki który jest kolejny dostępny. Data została dodana ręcznie wedle określonego formatu.

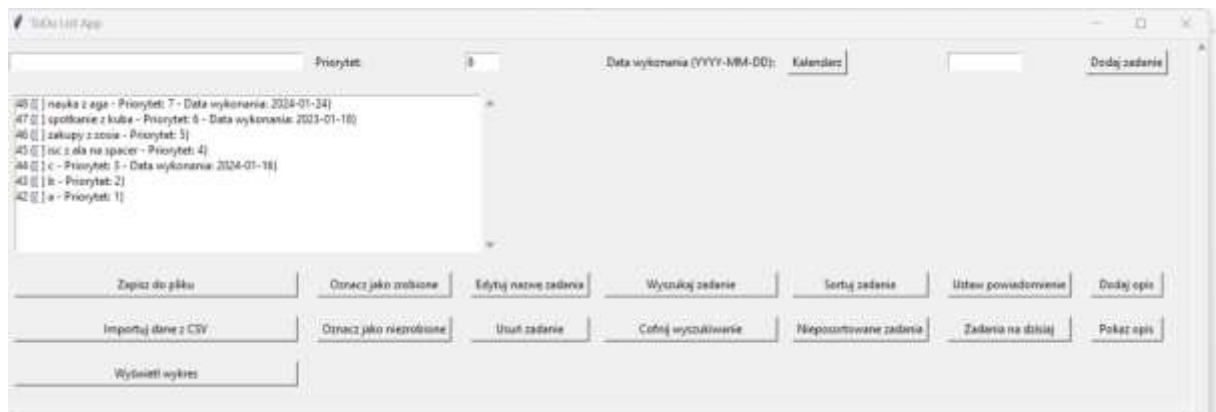
The screenshot shows the 'Todo List App' interface. The 'Task description' is 'spotkanie z kuba', 'Priority' is empty, and 'Due date' is manually set to '2023-01-18'. The 'Dodaj zadanie' button is on the right. Below the inputs is a list of tasks: '46 [] | spotkanie z kuba - Priorytet: 3', '45 [] | nic z ala na spacer - Priorytet: 4', '44 [] | c - Priorytet: 3 - Data wykonania: 2024-01-16', '43 [] | b - Priorytet: 2', and '42 [] | a - Priorytet: 1'. The bottom buttons are the same as in the previous screenshots.



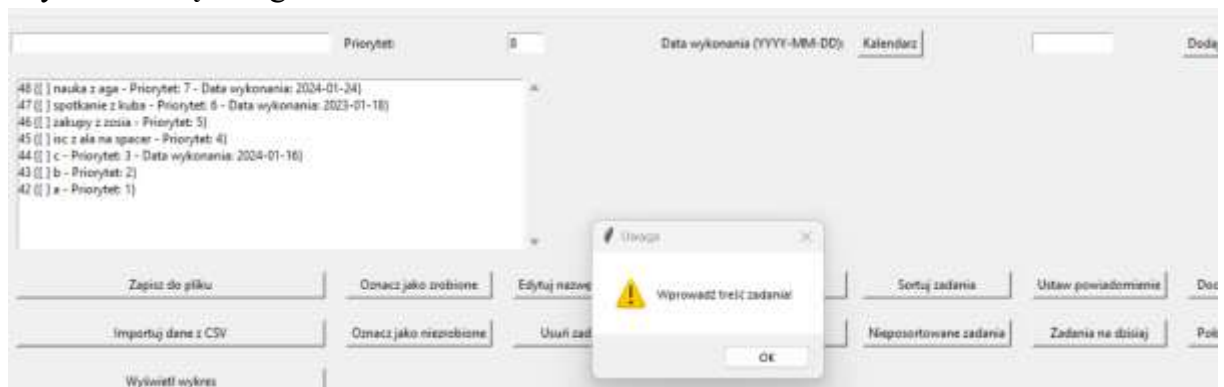
IV.1.4 Dodano zadanie wpisując jego treść nie dodając, priorytet automatycznie zostanie dodany po dodaniu zadania do listy. Priorytet który będzie dodany, to taki który jest kolejny dostępny. Data została wybrana z kalendarza (zaimplementowano datepicker)



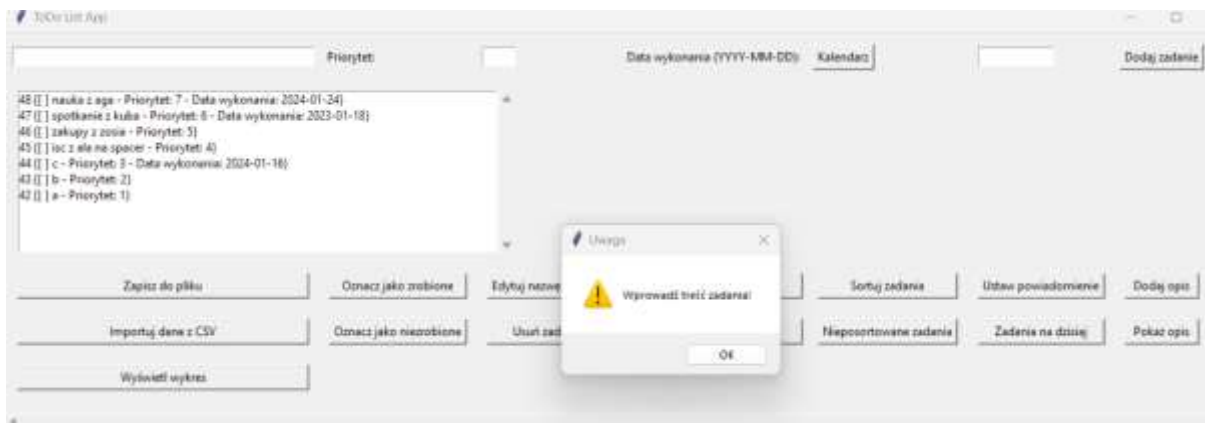
IV.1.5 Sytuacja w której ktoś będzie chciał dodać sam priorytet



Wyświetla się uwaga:

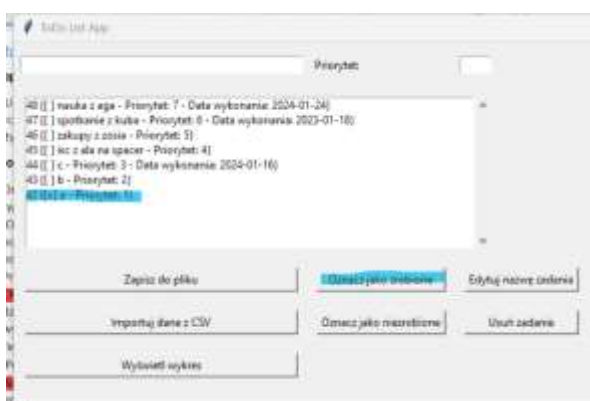


IV.1.6 Sytuacja, gdy ktoś po prostu chce dodać puste pola- wyświetla się uwaga, aby podać treść zadania (taka uwaga wyświetla się ponieważ można dodać tylko treść zadania, a priorytet zostanie dodany automatycznie)

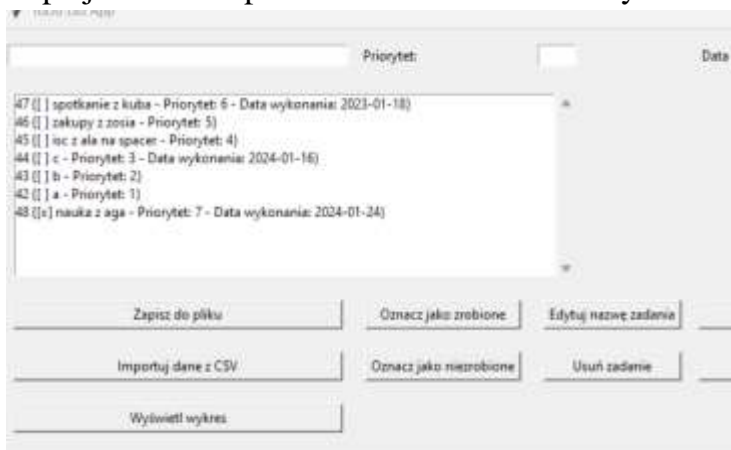


IV.2. Oznacz zadanie jako zrobione

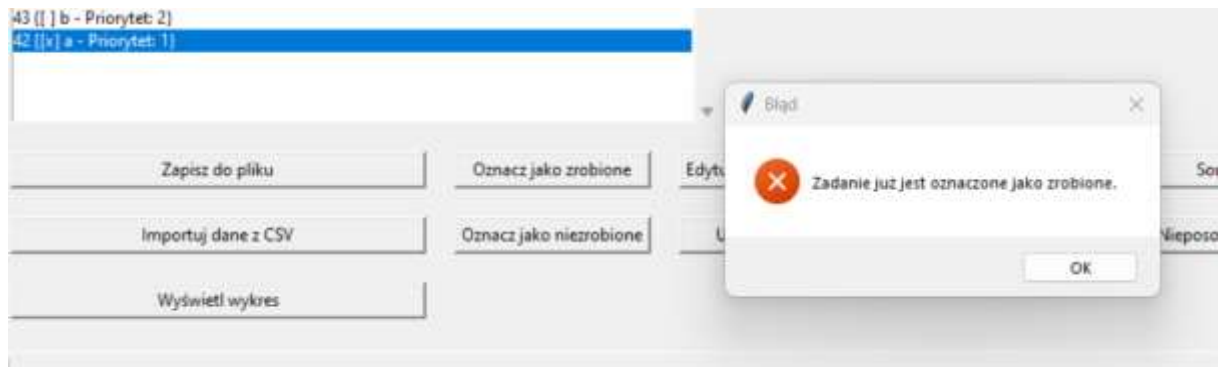
IV.2.1 Gdy chcemy oznaczyć zadanie jako zrobione, najpierw wybieramy zadanie następnie przyciskamy przycisk „Oznacz jako zrobione”, a zadanie „spada” na koniec listy.



Lepiej widoczne przeniesienie na koniec listy

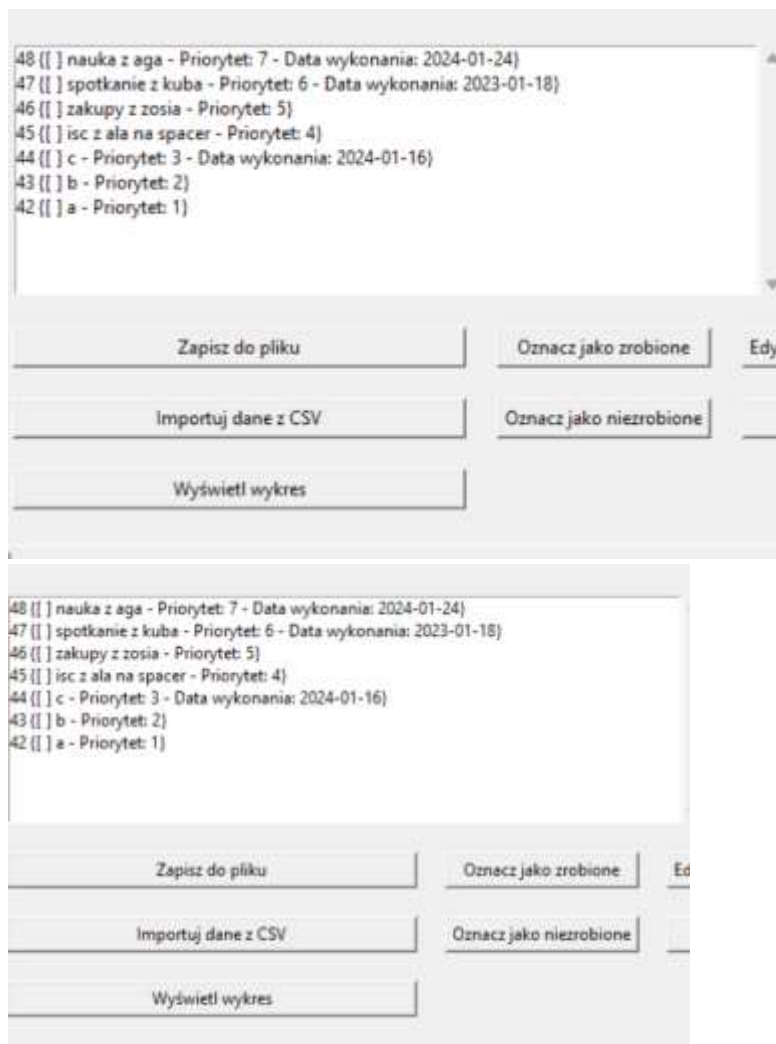


IV.2.2 Obsługa błędu oznaczenia jako zrobione, gdy chcę oznaczyć zadanie oznaczone jako zrobione, wyświetla się błąd

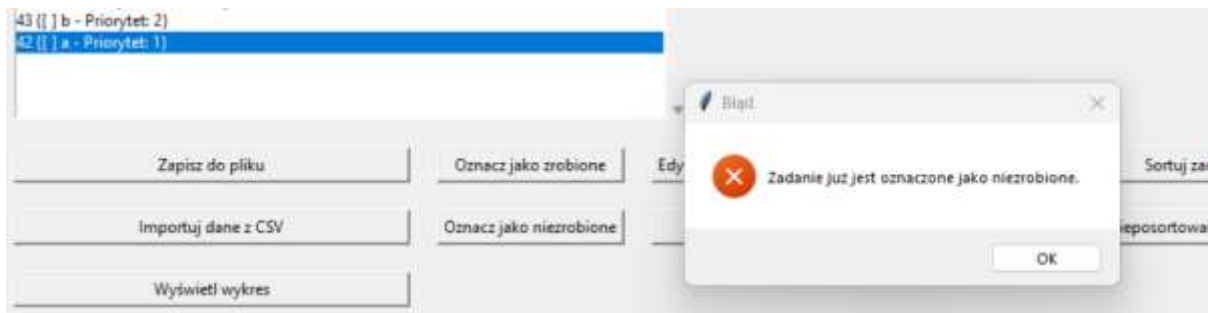


IV.3. Oznacz jako niezrobione

IV.3.1 Gdy oznaczam jako niezrobione zadanie wcześniej zaznaczone jako zrobione, wraca na swoje miejsce.



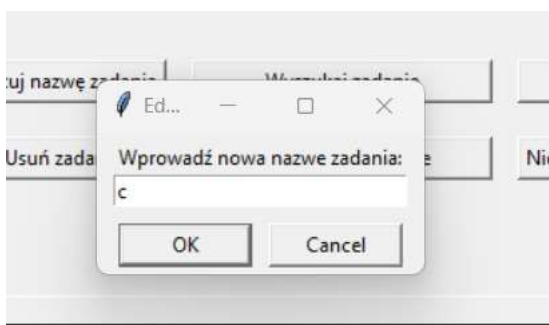
IV.3.2 Obsługa błędu oznaczenia jako niezrobione, gdy chcę oznaczyć zadanie oznaczone jako niezrobione, wyświetla się błąd



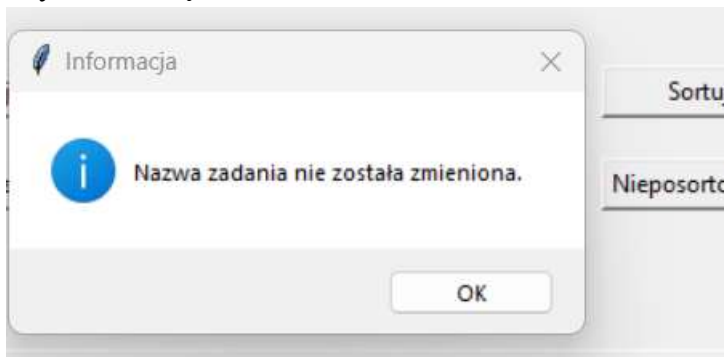
IV.4. Edytuj nazwę zadania

Tutaj listbox na screenach może się nieco różnić od innych, ze względu na to, iż część tego podpunktu była poprawiana.

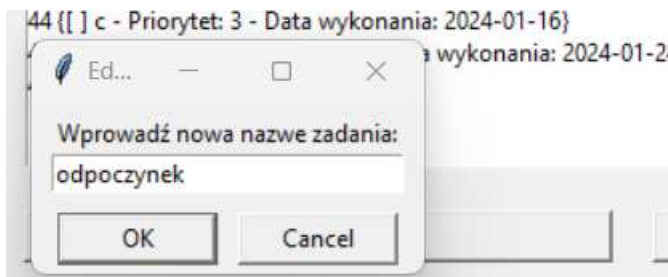
IV.4.1 Gdy wybrano zadanie o nazwie c i nie zmieniono



Wyświetlił się taki komunikat, że nazwa nie została zmieniona



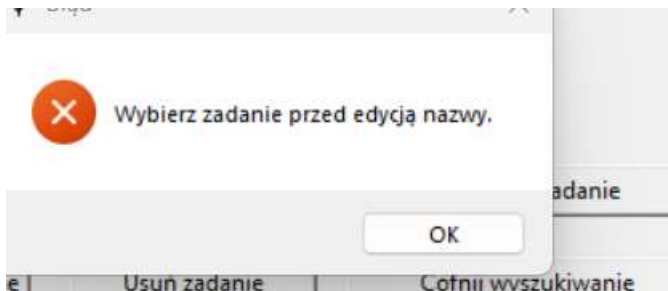
IV.4.2 Gdy wybrano zadanie o nazwie c i wprowadzono nową nazwę



Widoczna jest informacja, że nazwa zadania została pomyślnie zaktualizowana:

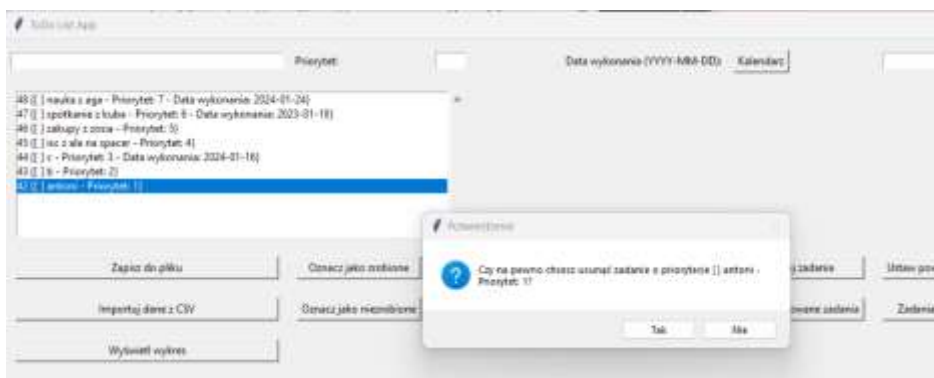


IV.4.3. Komunikat błędu, gdy zadanie nie zostało wybrane:

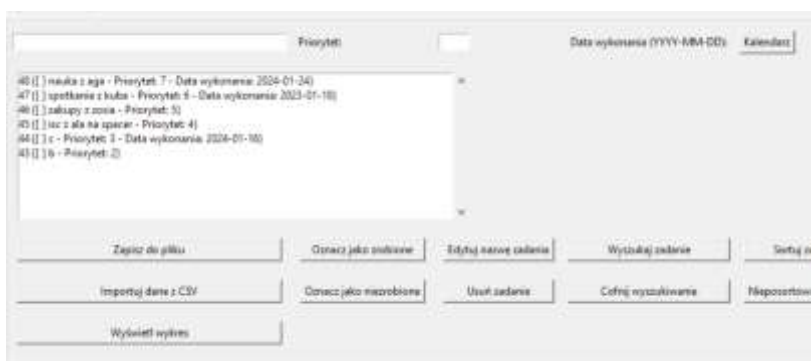


IV.5. Usuń zadanie

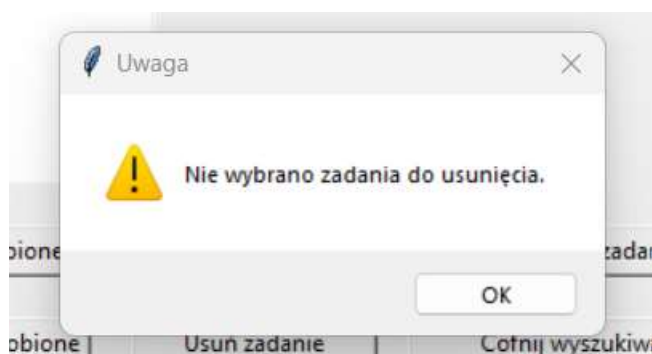
IV.5.1. Wybrano zadanie, kliknięto w przycisk. Przed usunięciem pojawi się komunikat:



Na liście nie widnieje już to zadanie:

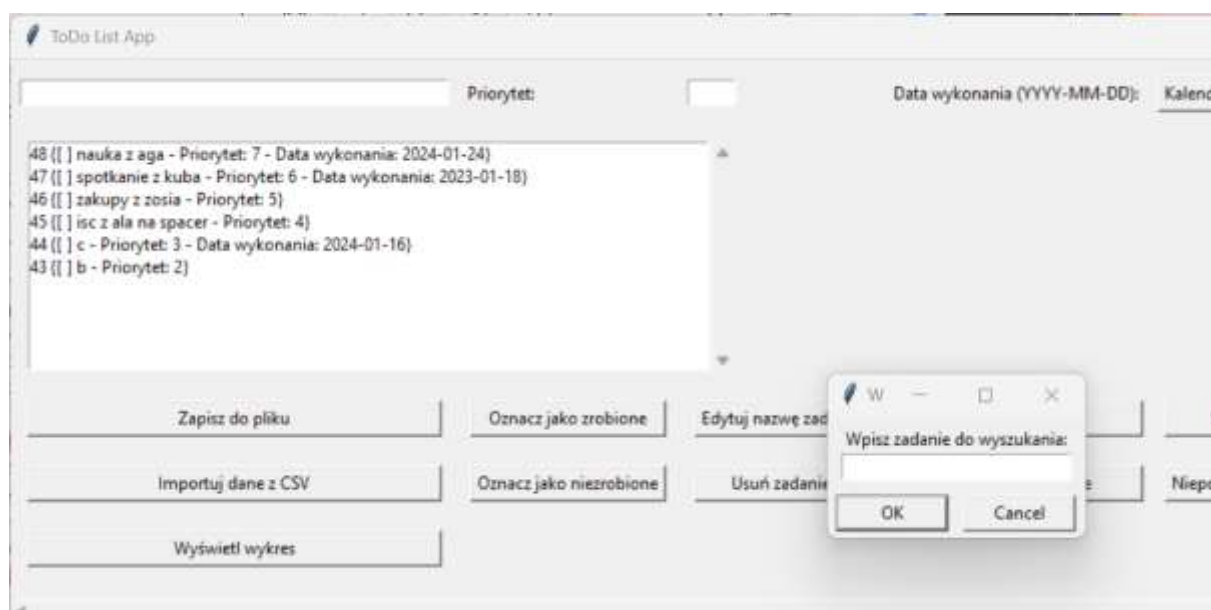


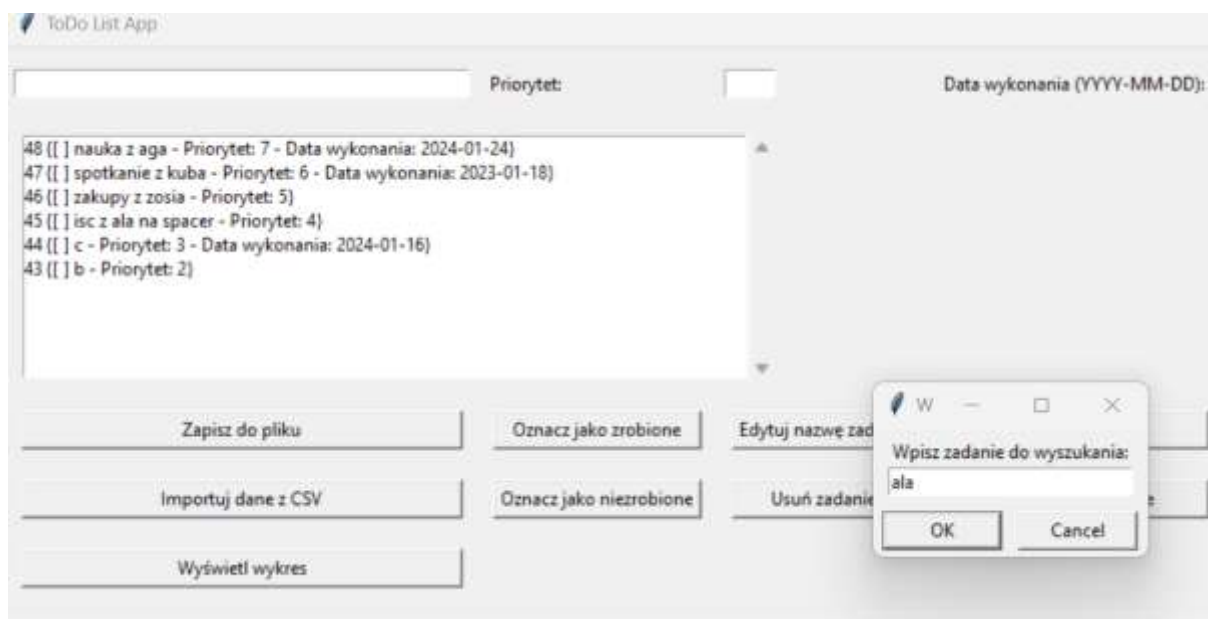
IV.5.2 Obsługa błędu, gdy nie wybierze się zadania a kliknie się przycisk usuń zadanie



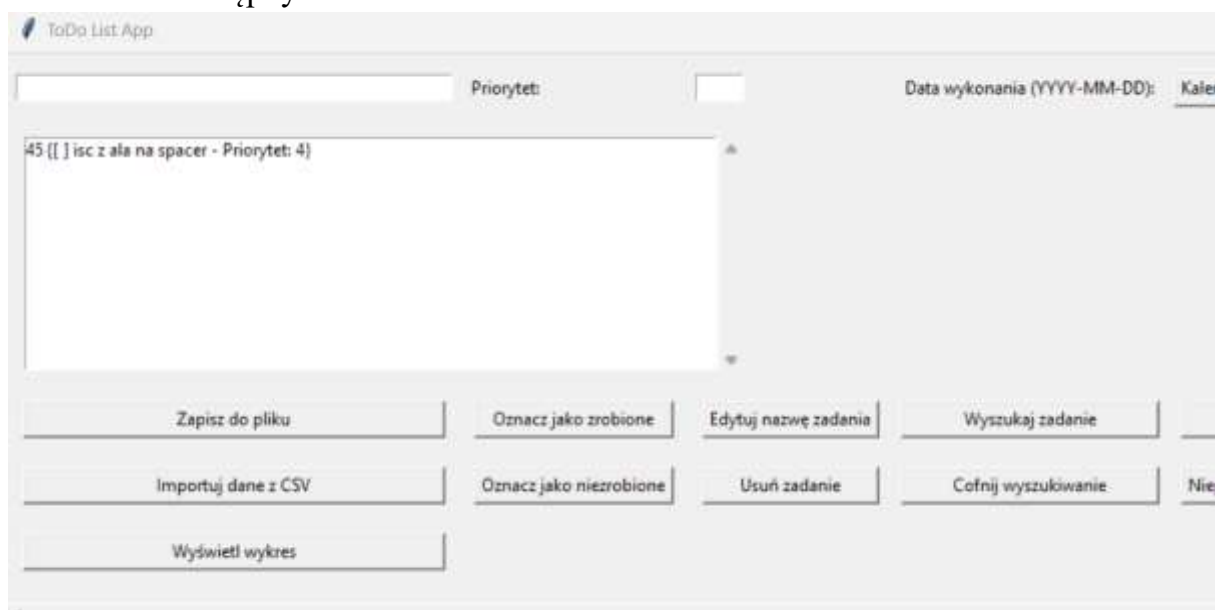
IV.6. Wyszukaj zadanie

IV.6.1 Po naciśnięciu przycisku wyszukaj zadanie, pojawia się okienko, w które można wpisać treść zadania do wyszukania:





Po kliknięciu „OK” wyświetli się zadanie zawierające wpisaną treść do wyszukiwania w treściach dostępnych zadań.

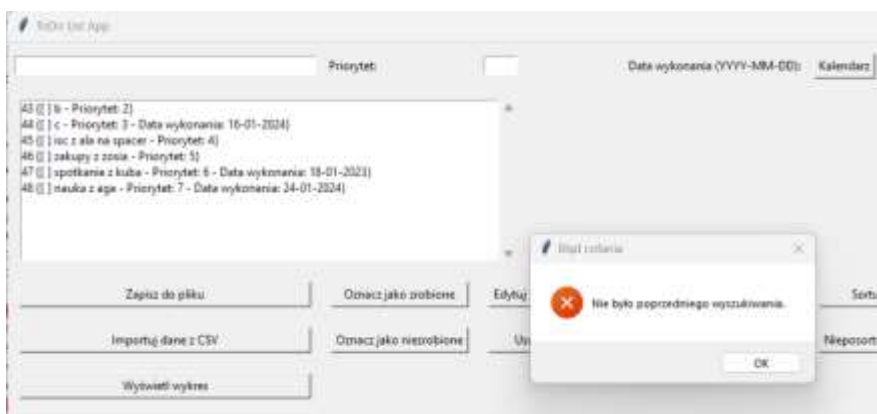


IV.7 Cofnij wyszukiwanie

IV.7.1 Po wyszukiwaniu chcemy wrócić do zawartości poprzedniej listy, wykonano to tak, że dodatkowo sortuje to po ID zadań, dla porządku.

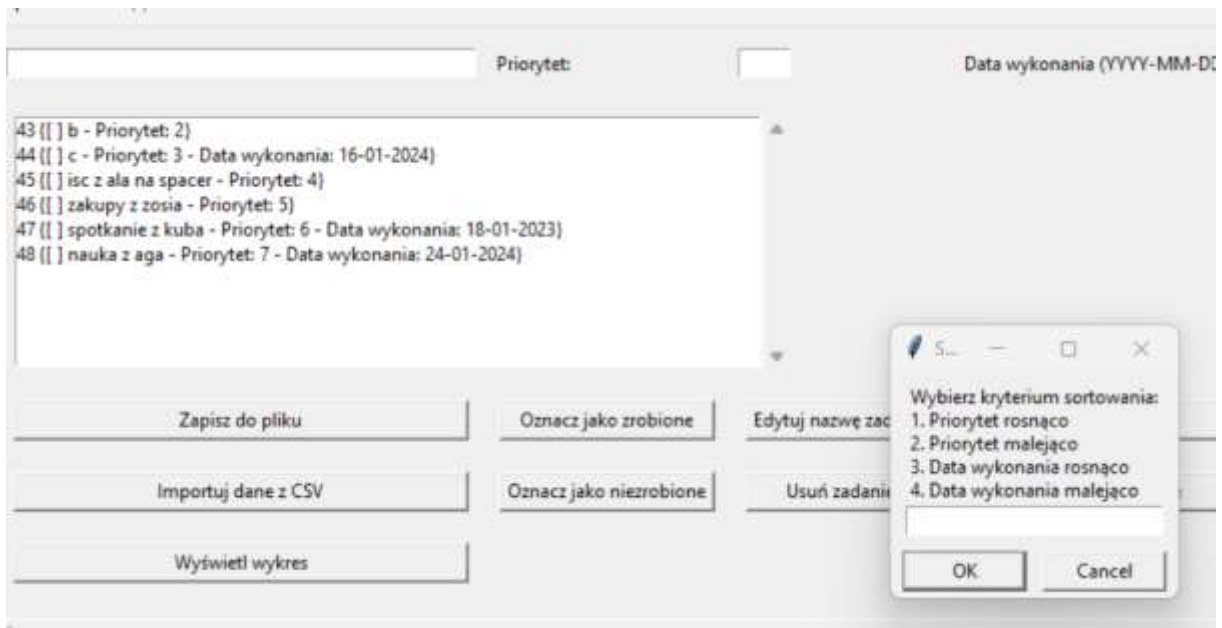


IV.7.2 Obsługa błędu dla cofania. Gdy nie było poprzedniego wyszukiwania, a kliknie się cofnij, wyświetli się taki komunikat. Tak samo gdy wyszukiwanie się nie powiodło.

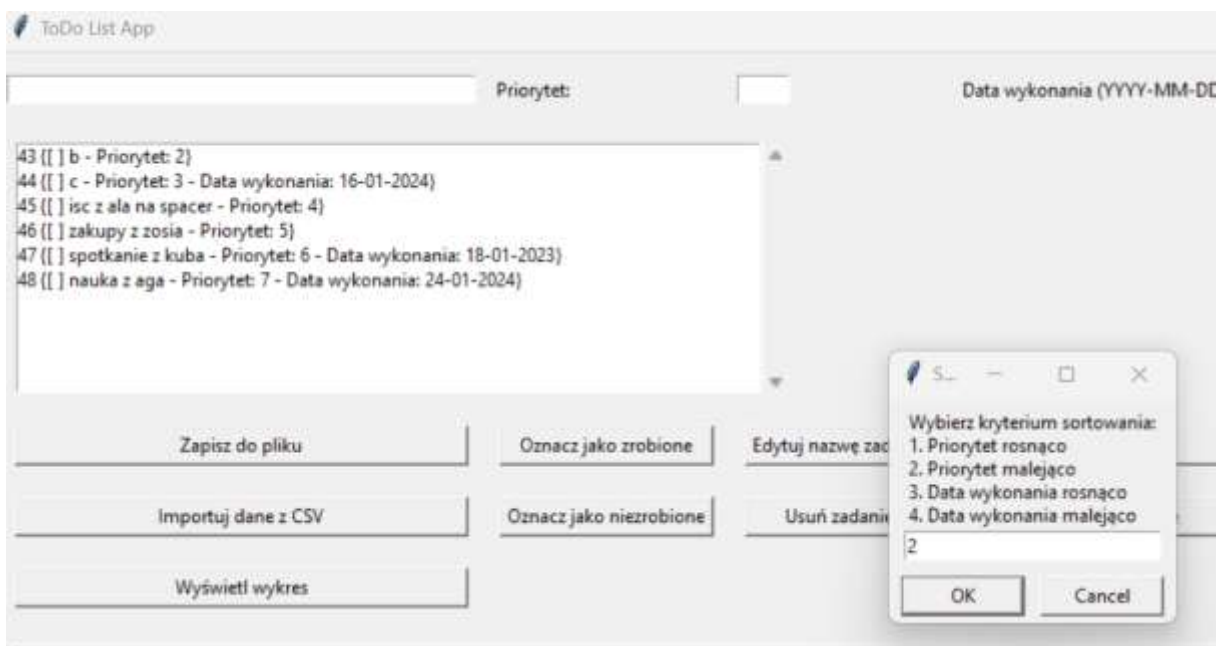


IV.8. Sortuj zadania

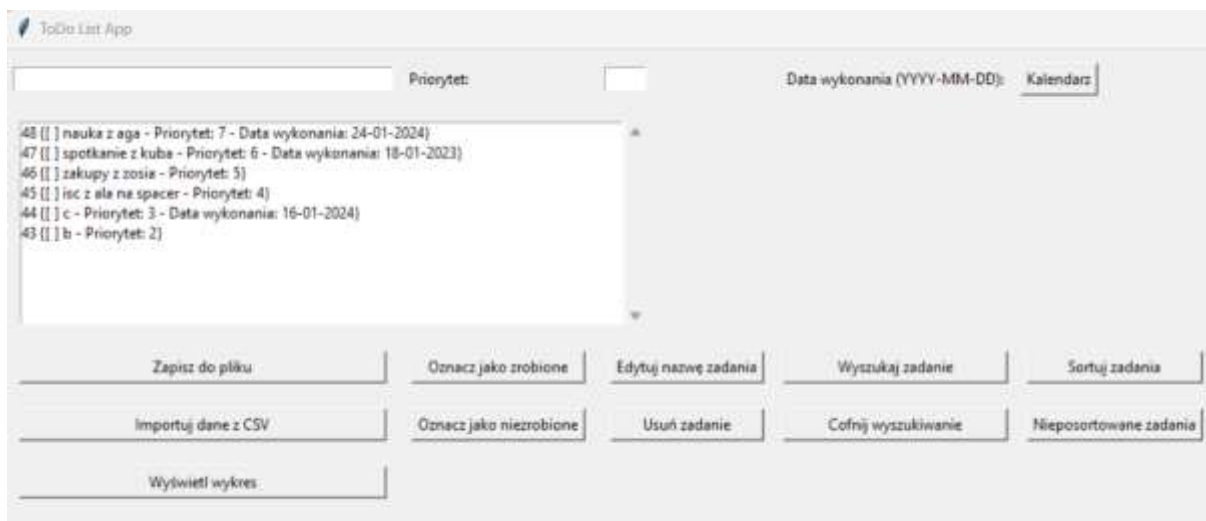
IV.8.1 Po naciśnięciu przycisku pojawi się okienko z wyborem opcji:



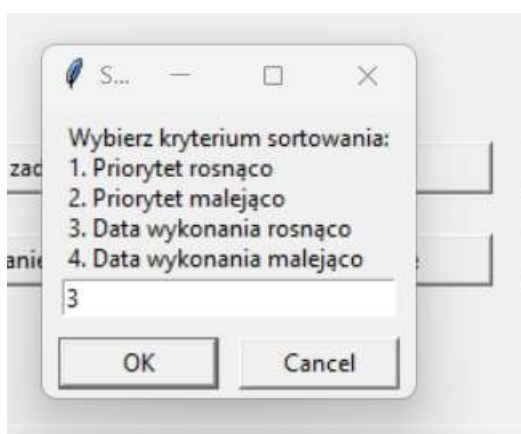
IV.8.2 Sortowanie zadań wedle: Priorytet malejąco



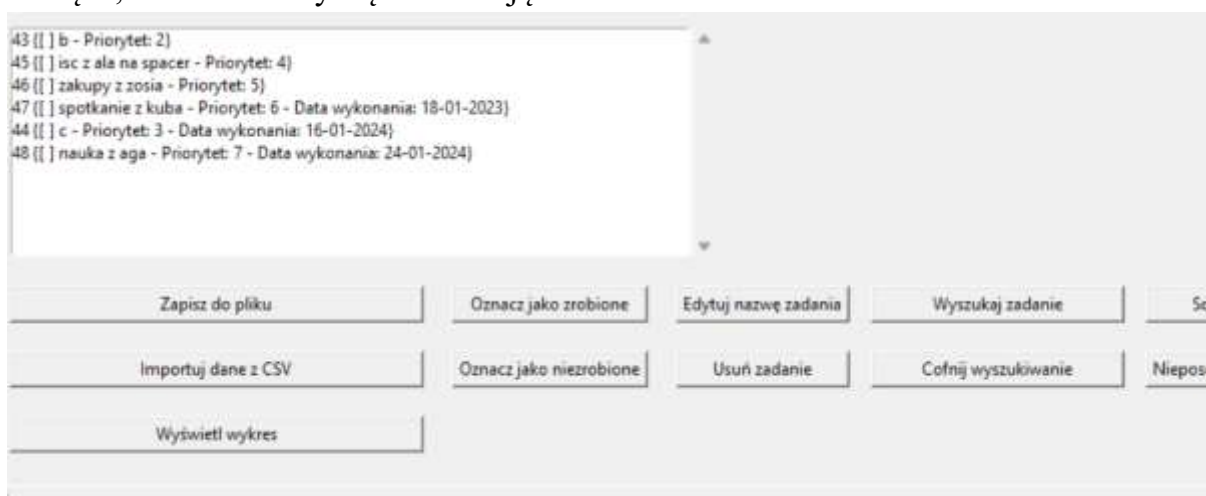
Zadania posortowane malejąco wedle priorytetów:



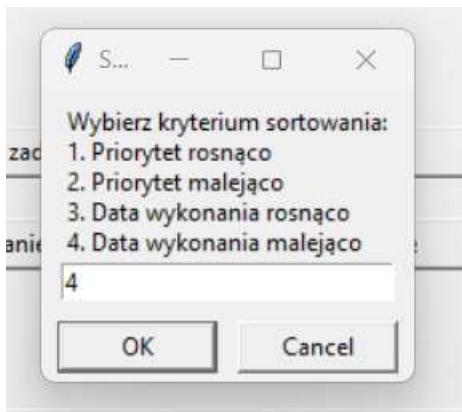
IV.8.3 Sortowanie zadań wedle: Data wykonania rosnąco



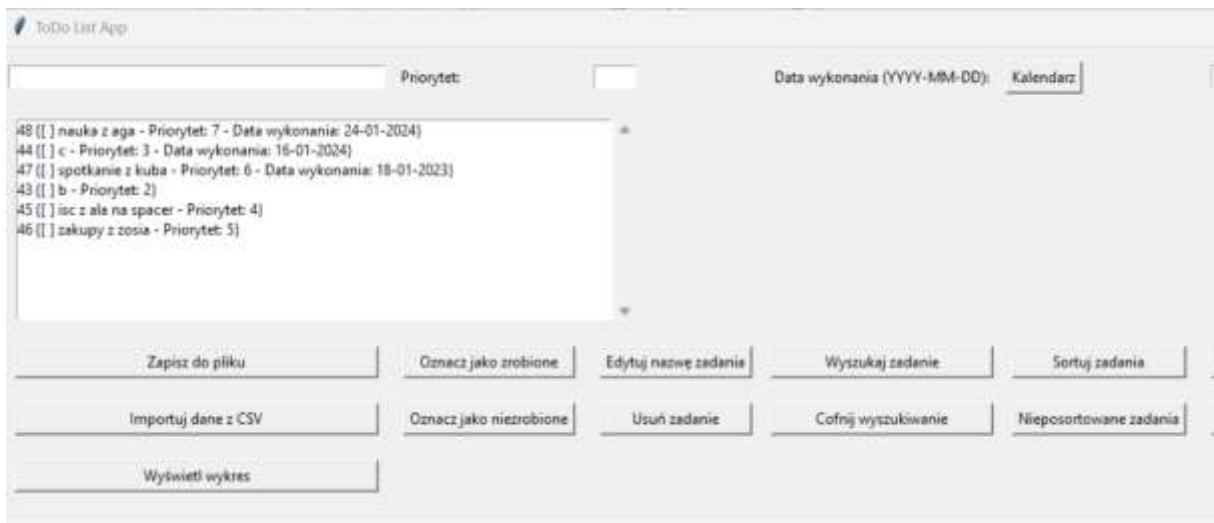
Priorytety są na początku listy, nie są brane pod uwagę w sortowaniu, ale są ułożone w rosnąco, ale wedle daty się nie sortują



IV.8.4 Sortowanie zadań wedle: Data wykonania malejąco

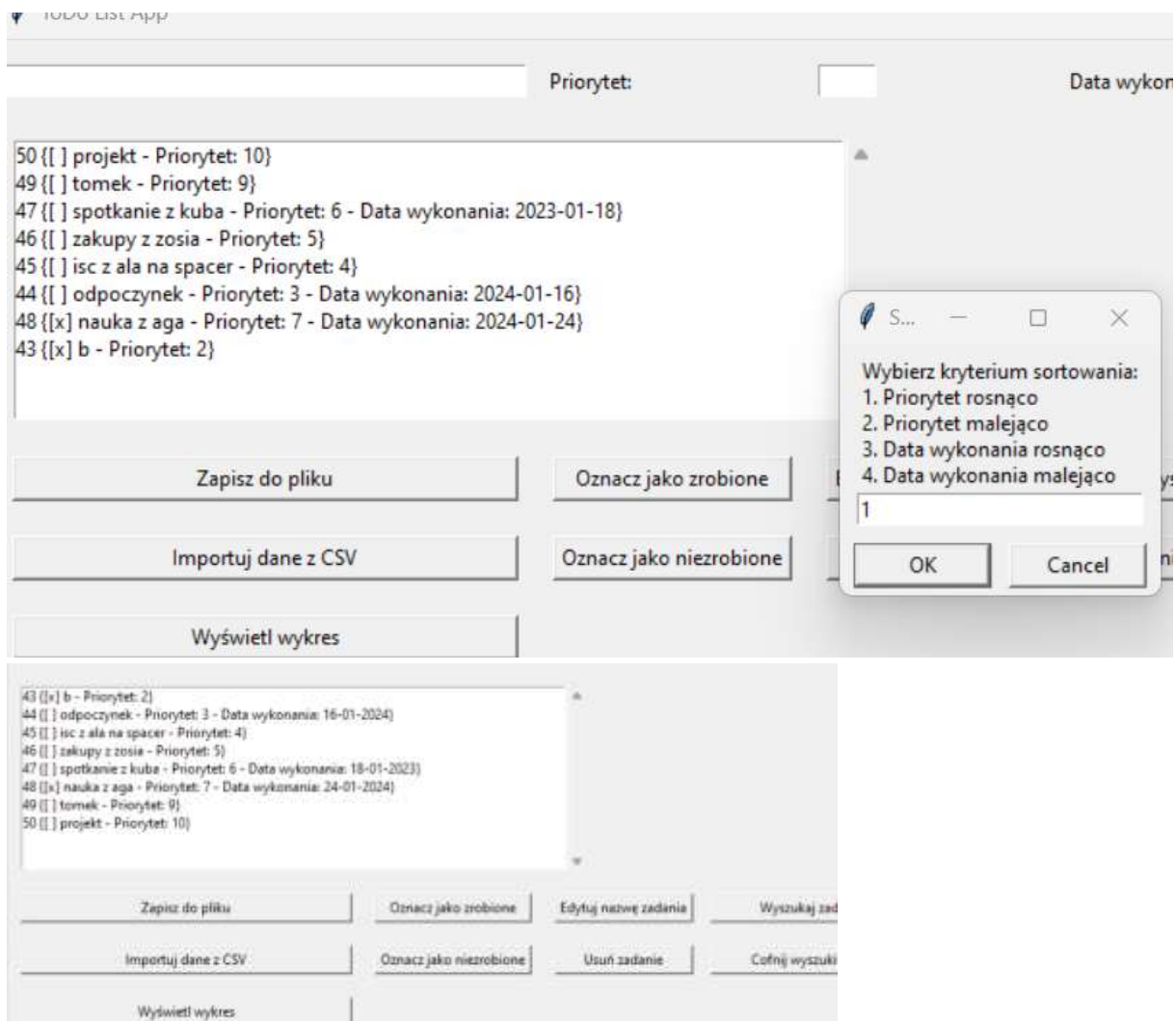


Priorytety są na końcu listy, nie są brane pod uwagę w sortowaniu, ale są ułożone w rosnąco, ale wedle daty się nie sortują



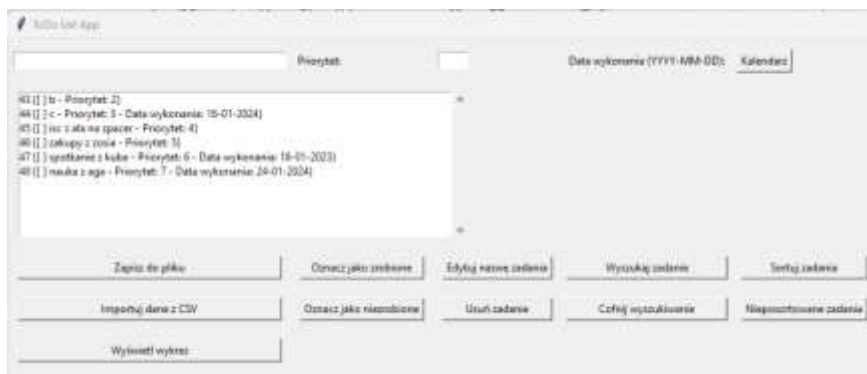
IV.8.5 Sortowanie zadań wedle: Priorytet rosnąco

Zadania na liście nie są takie same jak w poprzednich podpunktach, ponieważ ten podpunkt dodano do sprawozdania.



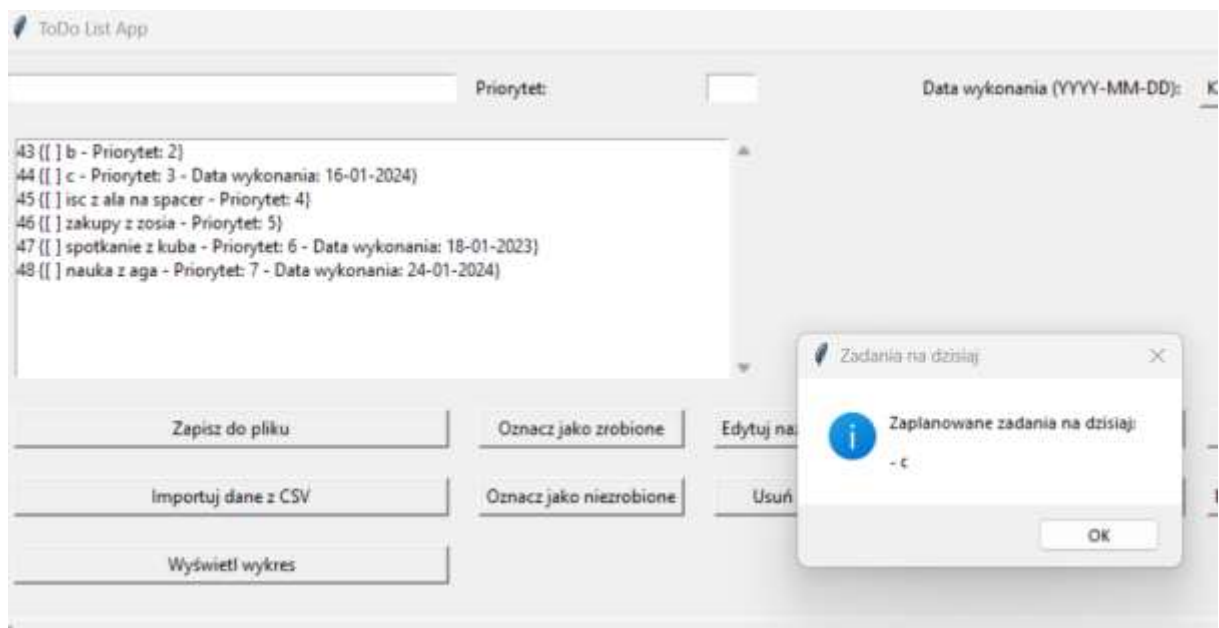
IV.9. Nieposortowane zadania

IV.9.1 Gdy lista zadań wygląda inaczej niż przed sortowaniem, lista po prostu wraca do stanu sprzed sortowania.



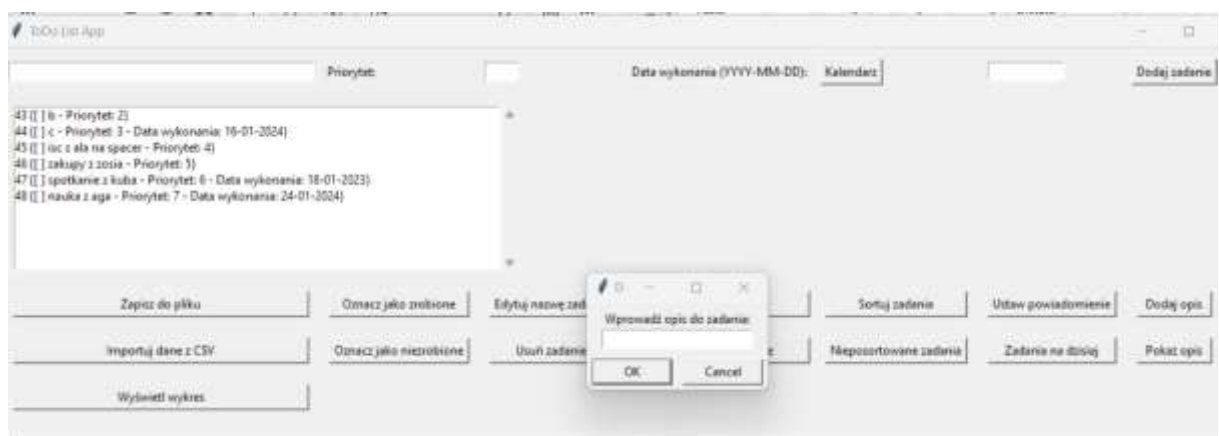
IV.10. Zadania na dzisiaj

IV.10.1 Pojawia się zaplanowana lista zadań na dziś

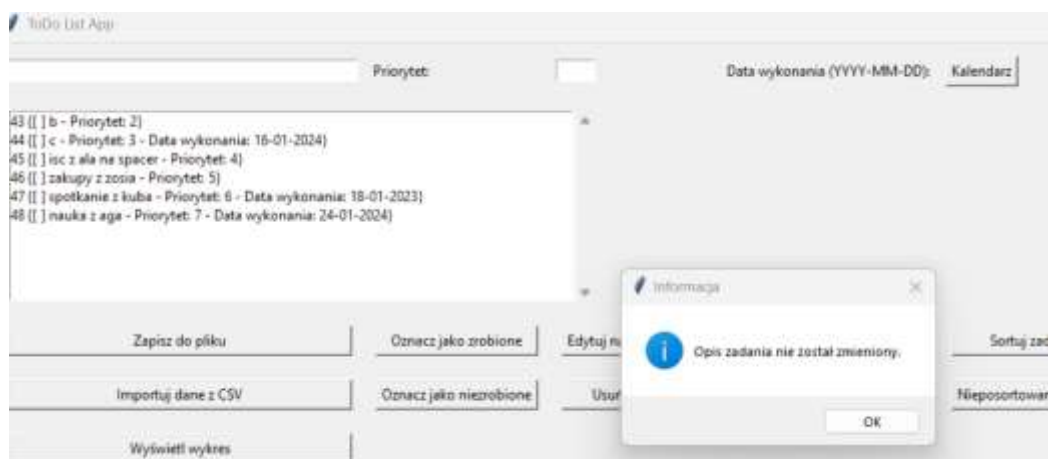


IV.11. Dodaj opis zadania

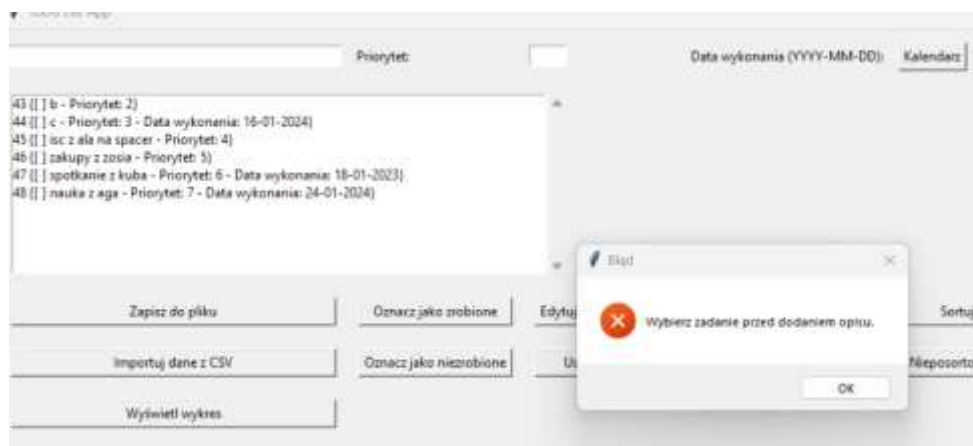
IV.11.1 Po naciśnięciu przycisku „dodaj opis” wyświetla się okienko, gdzie można wprowadzić opis.



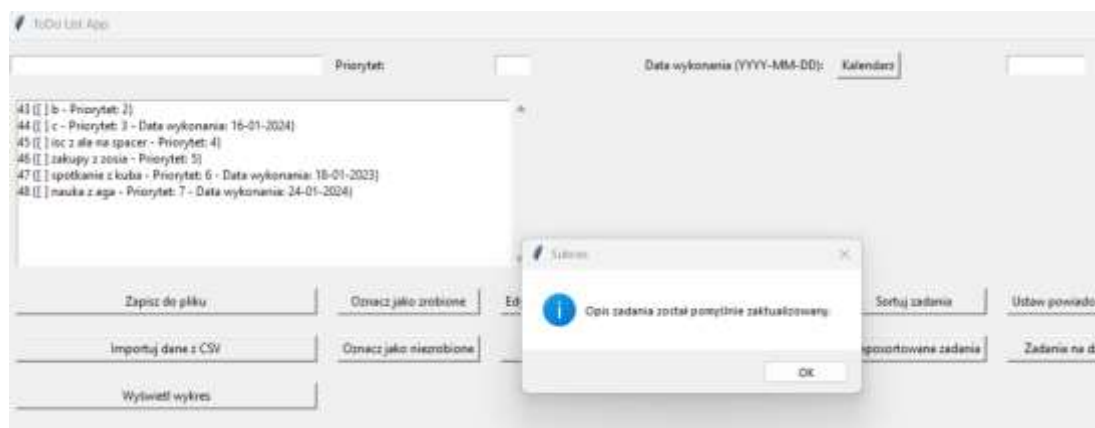
IV.11.2 Gdy jest już opis w zadaniu i kliknie się po prostu „OK” wyświetli się komunikat:



IV.11.3 Gdy się nie wybierze zadania tylko wciśnie się przycisk, pojawi się komunikat:

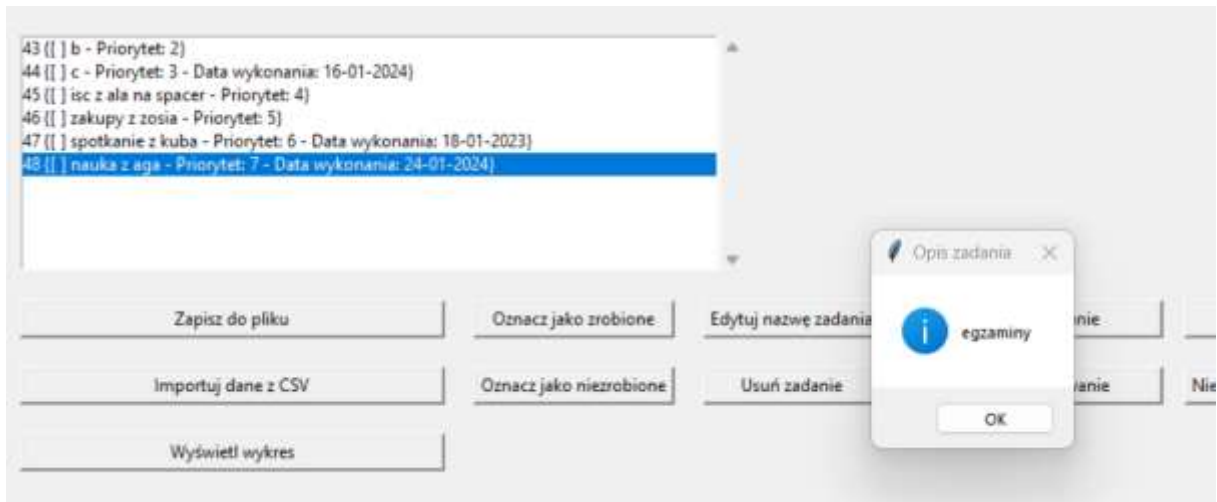


IV.11.4 Gdy się doda zadanie, pojawi się taki komunikat. Na listboxie nie pojawiają się opisy zadań, dopiero w funkcji pokaż opis można taki opis zobaczyć, zamysł był taki, aby lista była czytelniejsza

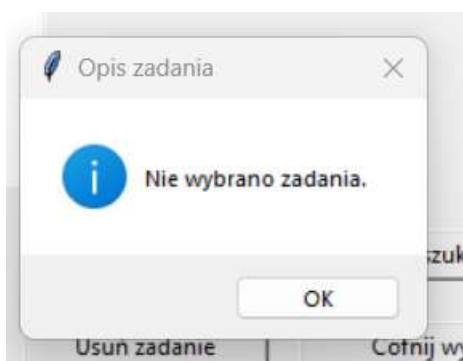


IV.12. Pokaż opis

IV.12.1 Po wybraniu zadania i kliknięciu w przycisk **pokaż opis**, wyświetla się opis:

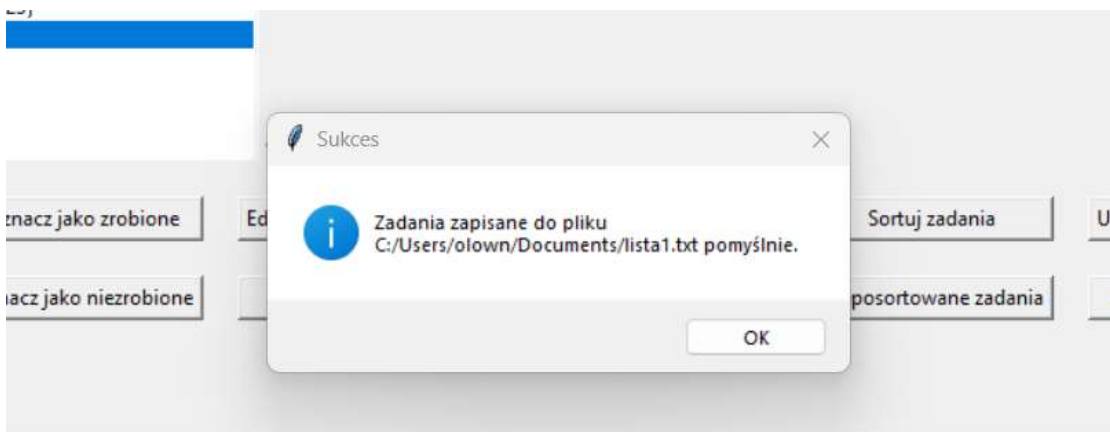


IV.12.2 Gdy nie wybierzemy zadania i klikniemy w „pokaż opis” pojawi się komunikat:

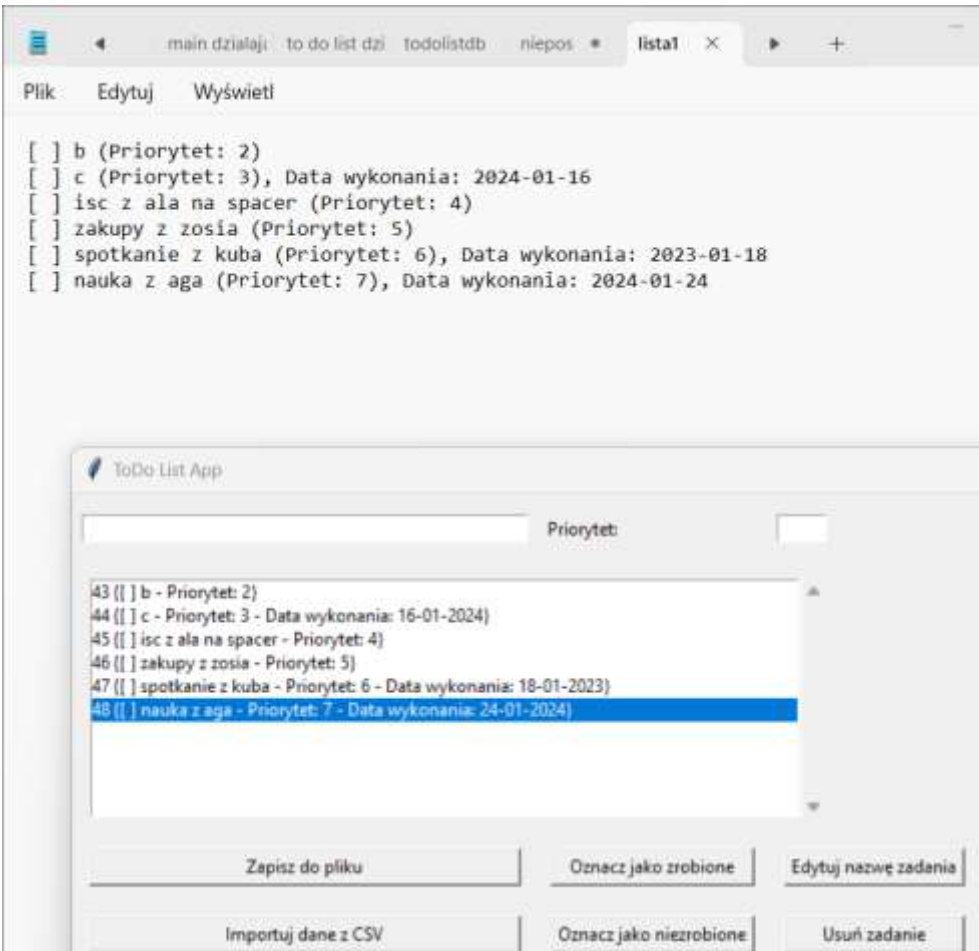


IV.13. Zapisywanie do plików

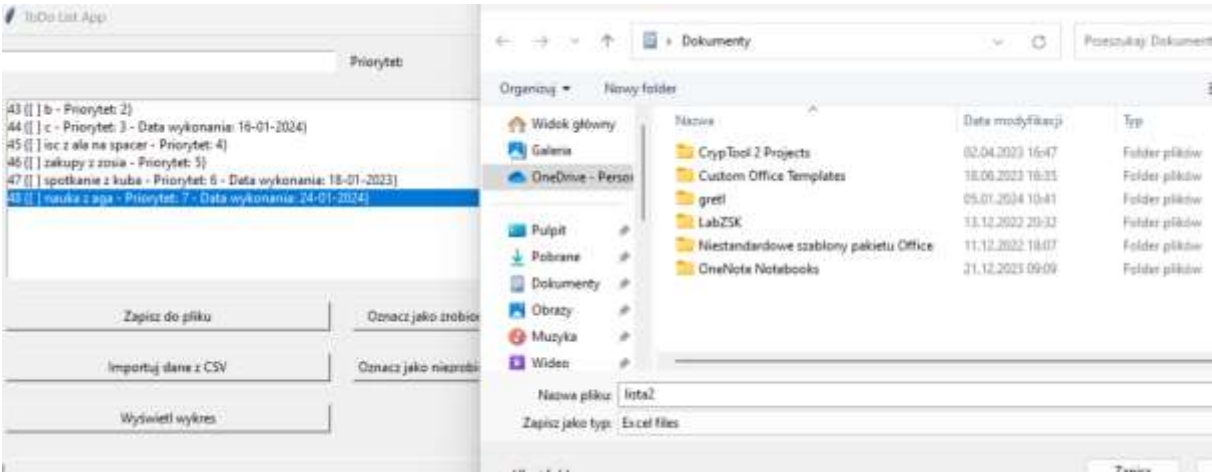
IV.13.1 Zapisywanie do formatu .txt:



IV.13.2 Wgląd do tego co znajduje się w pliku po zapisaniu:



IV.13.3 Zapis do pliku w formacie plików excel

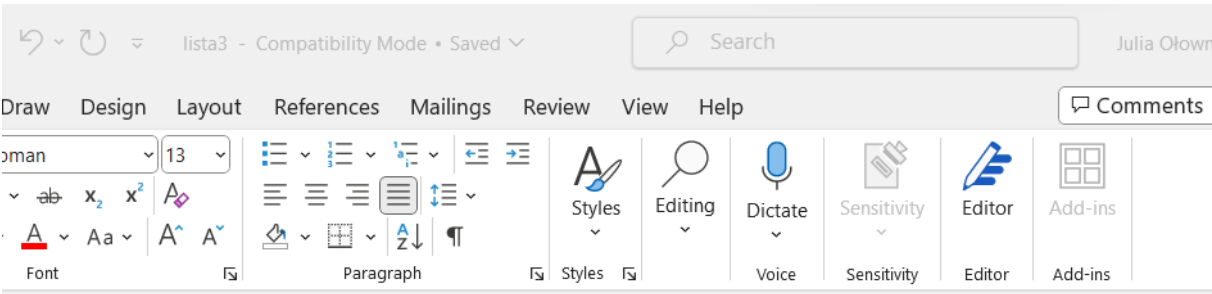
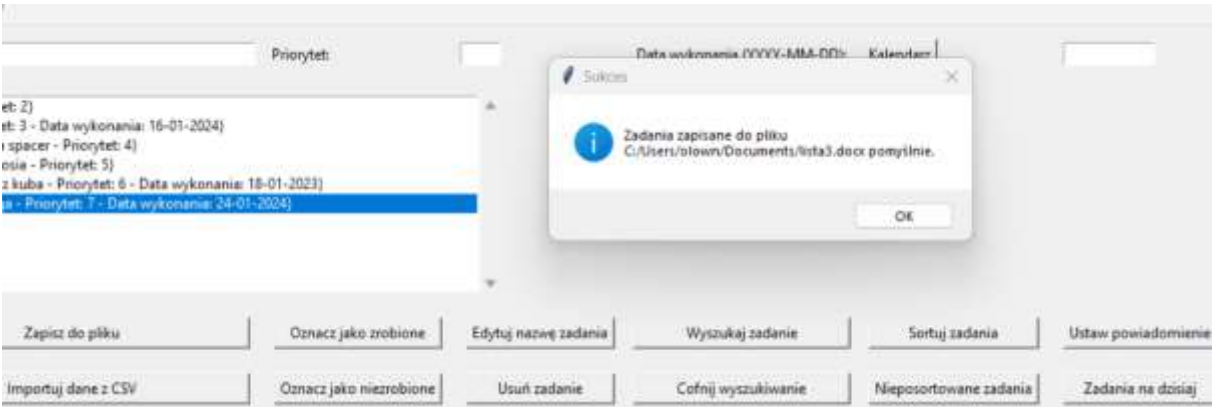


IV.13.4. Pomyślny zapis do pliku w formacie excel i wgląd do pliku jak został zapisany:

[illegible]

IV.13.5 Zapis do pliku w formacie plików word

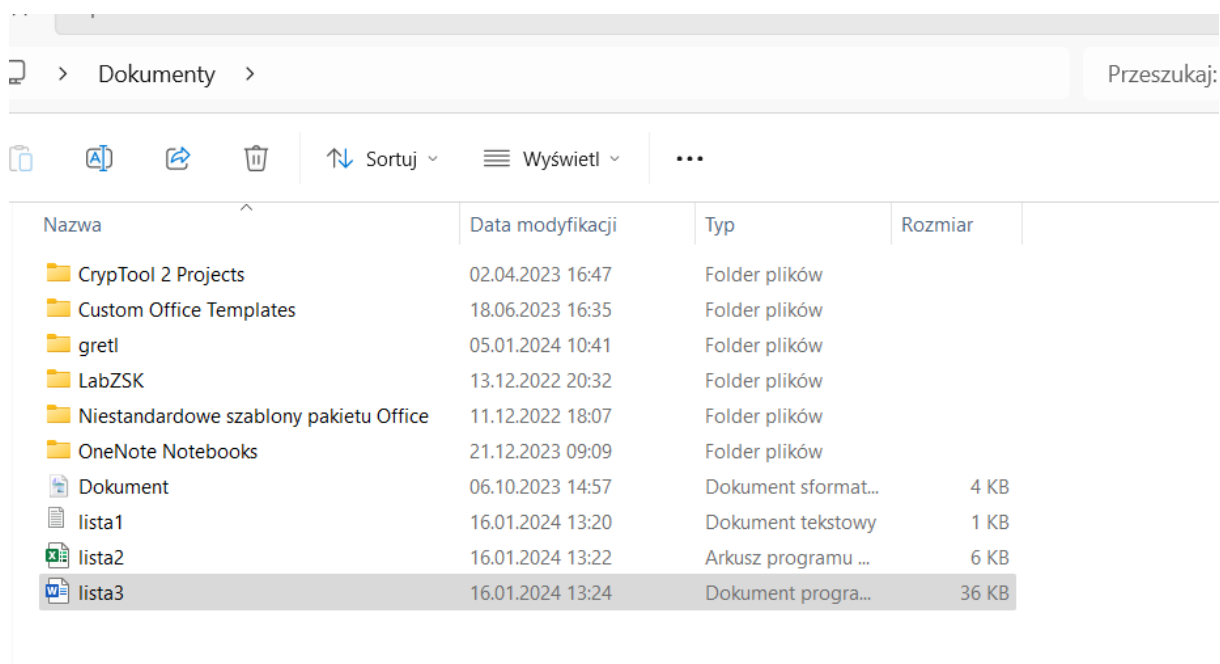
IV.13.6. Pomyślny zapis do pliku w formacie excel i wgląd do pliku jak został zapisany:



Lista zadań

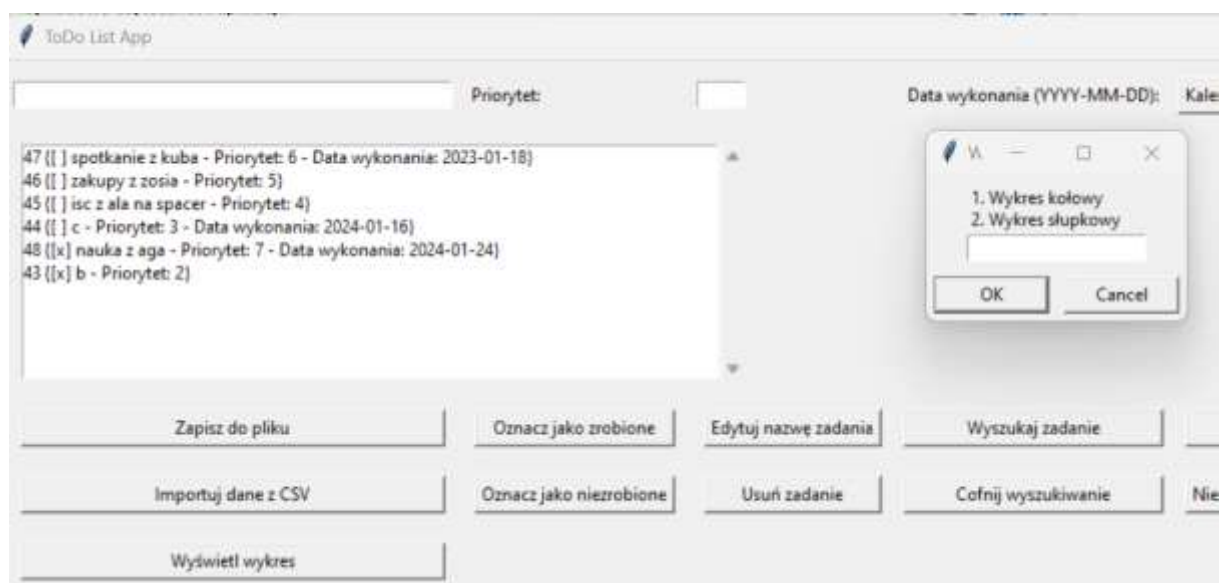
- [] b (Priorytet: 2)
- [] c (Priorytet: 3), Data wykonania: 2024-01-16
- [] isc z ala na spacer (Priorytet: 4)
- [] zakupy z zosia (Priorytet: 5)
- [] spotkanie z kuba (Priorytet: 6), Data wykonania: 2023-01-18
- [] nauka z aga (Priorytet: 7), Data wykonania: 2024-01-24

IV.13.7. Wgląd do dokumentów, czyli tam gdzie były zapisywane:

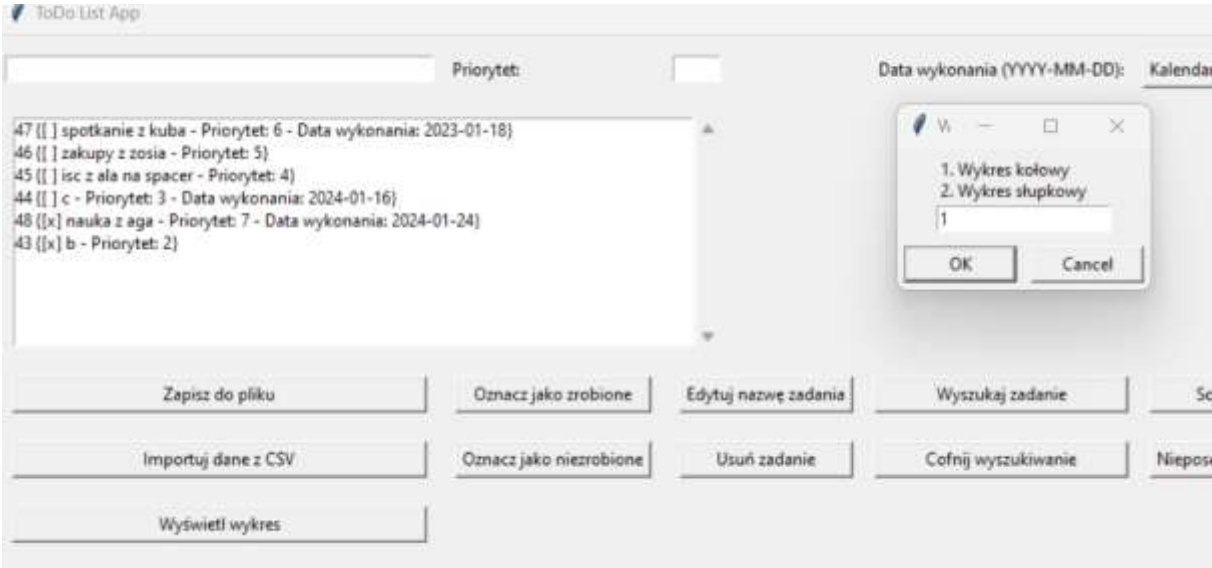


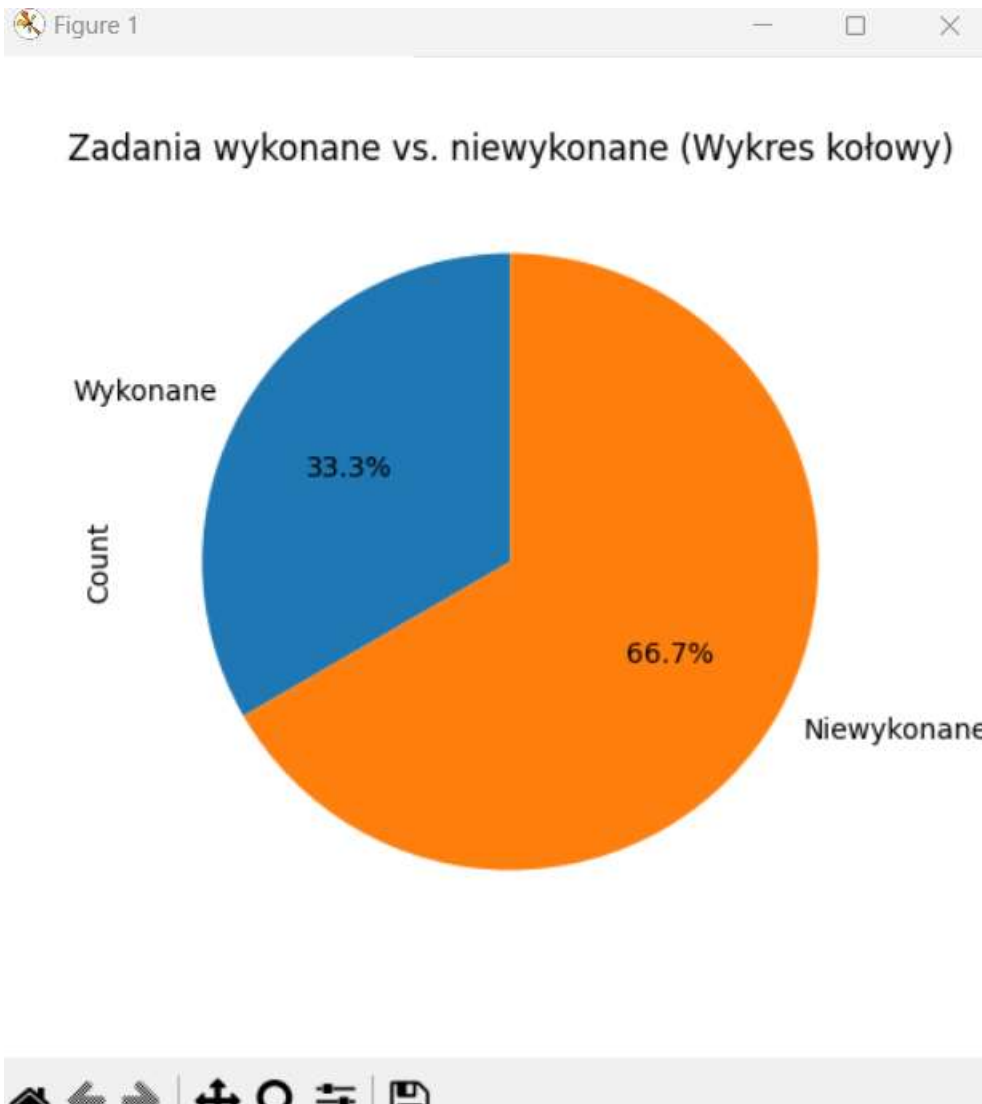
IV.14 Wykresy do statystyk

IV.14.1 Po otwarciu się okienka po kliknięciu w przycisk wyświetl wykres wyświetla się okienko wyboru w którym wybieram jaki chcę otrzymać wykres:



IV.14.2 Po wybraniu opcji 1 wyświetla się nam wykres kołowy zadań wykonanych vs niewykonanych – określający produktywność i progres wykonywanych zadań. Wykres można zapisać do pliku, zmieniać jego właściwości.





IV.14.3 Po wybraniu opcji 2 wyświetla się nam wykres słupkowy zadań z datą vs bez daty –określa ile zadań ma tzw. Deadline, ile nie. Wykres można zapisać do pliku, zmieniać jego właściwości.

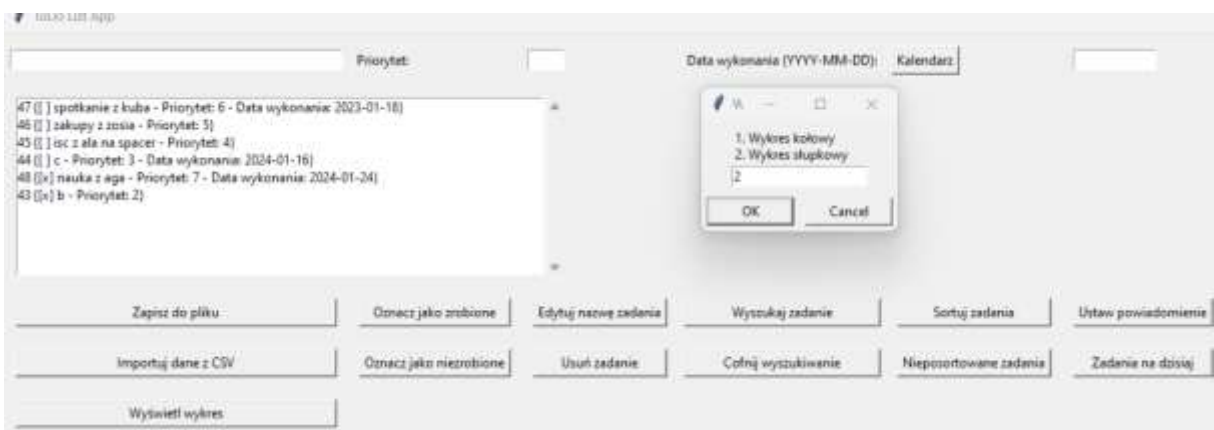
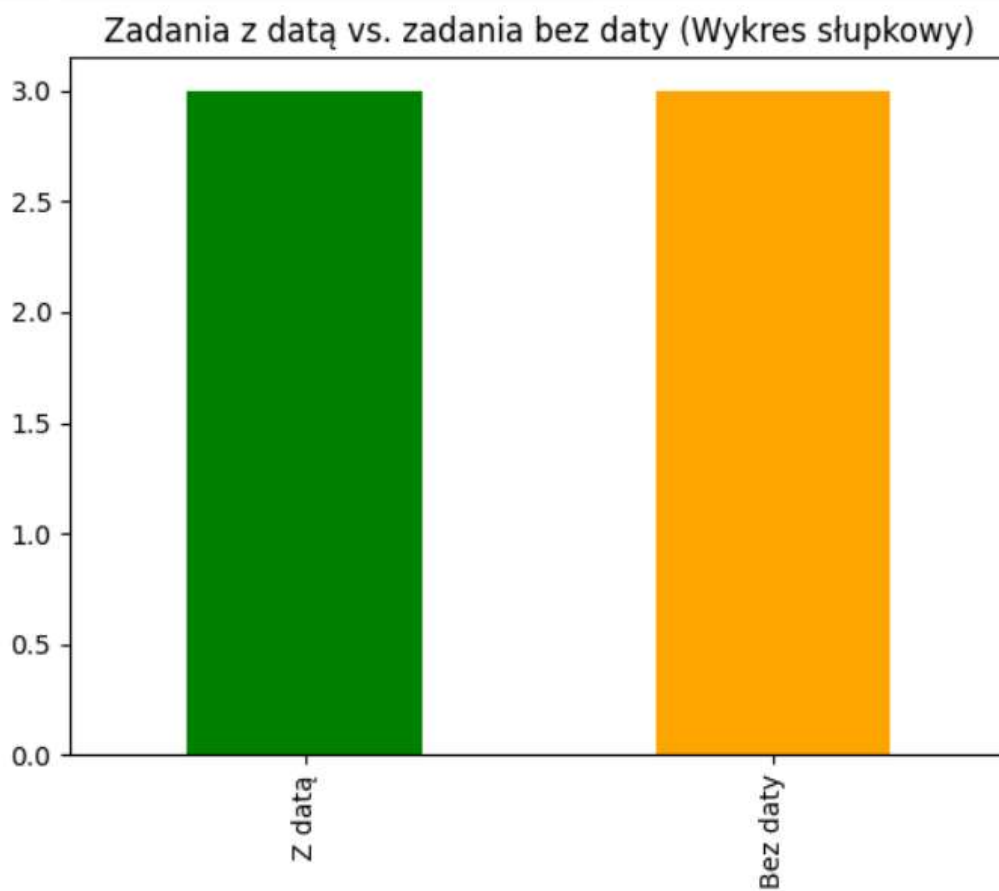
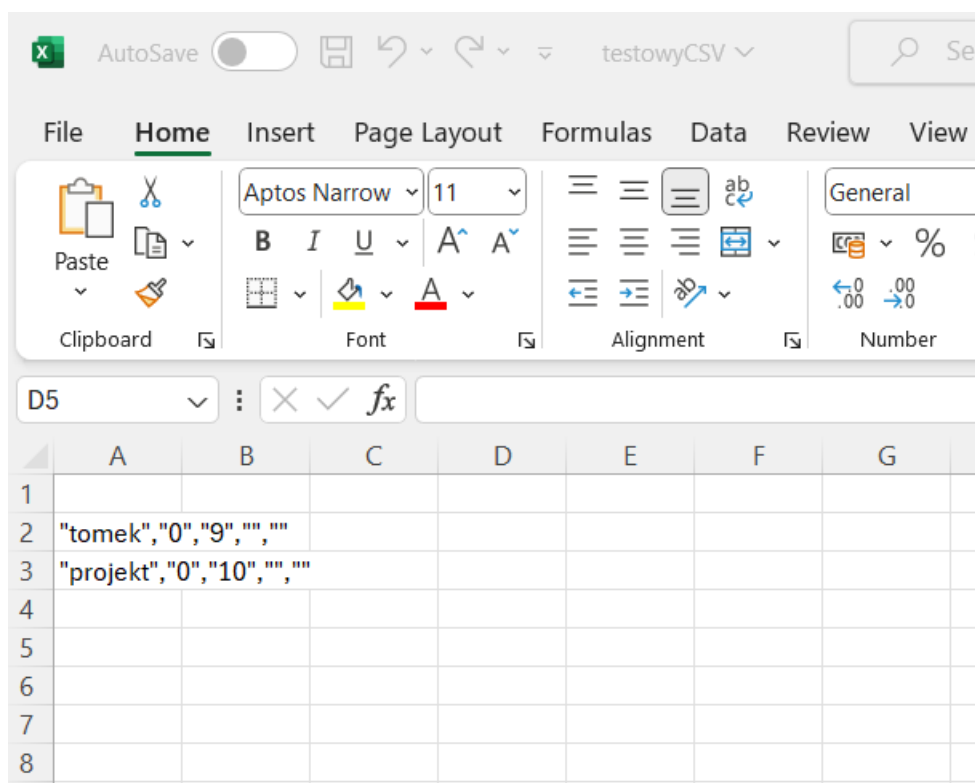


Figure 1

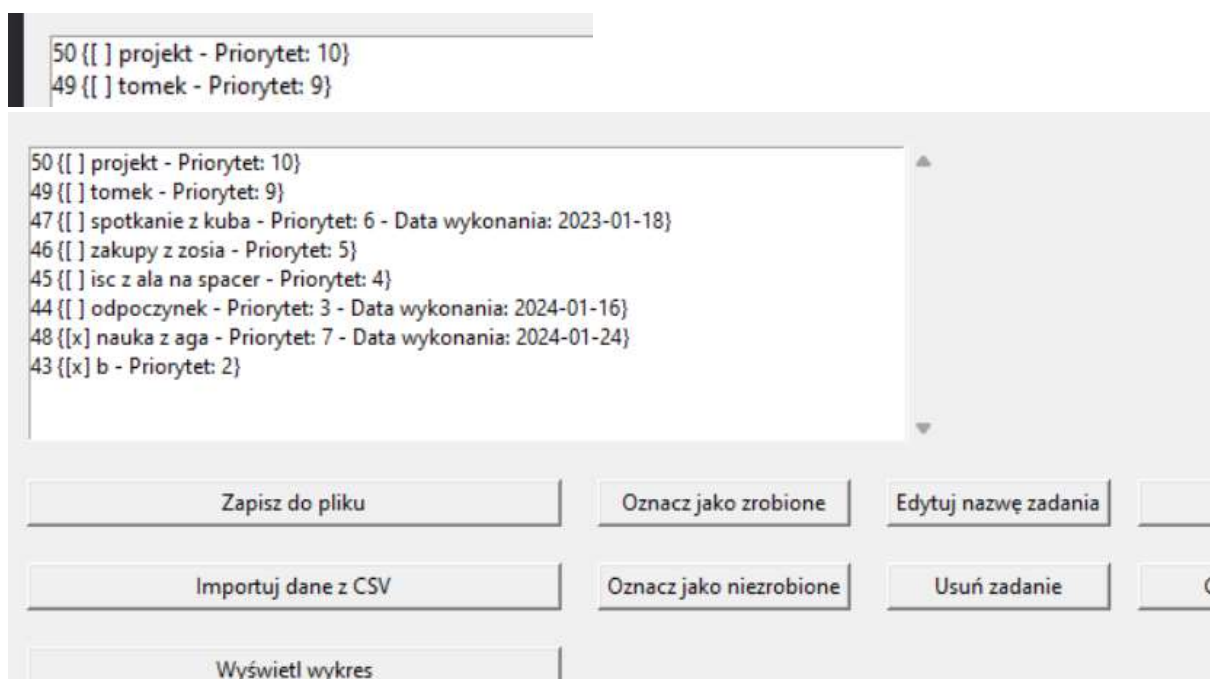


IV.15.Importuj dane z csv

IV.15.1 Plik CSV przykładowy aby pokazać import z niego:



IV.15.2 Zaimportowane dane do listy z pliku:



V. Podsumowanie oraz wnioski

V.1. Podsumowanie

Aplikacja "ToDo List App" to kompleksowe narzędzie, które zostało stworzone z myślą o łatwym i skutecznym zarządzaniu codziennymi obowiązkami. Oto szczegółowe podsumowanie, uwzględniające różnorodne funkcje i aspekty aplikacji.

Interfejs Graficzny i Dostosowywanie: Aplikacja posiada intuicyjny interfejs graficzny, oparty na bibliotece Tkinter w języku Python. Ten interfejs jest elastyczny i dostosowywany do zmiany rozmiaru okna, co umożliwia płynne korzystanie z aplikacji na różnych urządzeniach i zróżnicowanych rozdzielczościach ekranów. Przejrzysty układ interfejsu ułatwia użytkownikowi szybkie dodawanie, edytowanie i przeglądanie zadań.

Baza Danych SQLite: Aplikacja wykorzystuje bazę danych SQLite do przechowywania zadań. Ta struktura umożliwia efektywne zarządzanie informacjami o zadaniach, gwarantując jednocześnie spójność danych. Dzięki bazie danych użytkownik może wykonywać operacje takie jak dodawanie, usuwanie, edytowanie oraz wczytywanie zadań. Jednakże nie działa na tyle dobrze lub nie została wykorzystana na tyle dobrze i nie udało się przypisywać zadań do użytkowników.

Zapis i Import Zadań: Aplikacja umożliwia zapisywanie listy zadań do różnych formatów plików, w tym tekstowego (txt), arkusza kalkulacyjnego (xlsx) i dokumentu tekstowego (docx). To przydatne narzędzie do archiwizacji i udostępniania zadań. Dodatkowo, funkcja importu z plików CSV ułatwia przenoszenie danych między różnymi systemami.

Powiadomienia: Wprowadzenie funkcji powiadomień pozwala użytkownikowi na ustawianie przypomnień dla konkretnych zadań na określone godziny. Aczkolwiek nie udało się połączyć aplikacji z powiadomieniami push na pulpit. Jednakże lista zadań do wykonania na dziś umożliwi użytkownikowi być na bieżąco.

Generowanie wykresów: Pozwala użytkownikowi zwizualizować sobie jego postępy.

Responsywność Interfejsu: Aplikacja dba o responsywność interfejsu, co sprawia, że korzystanie z niej jest przyjemne niezależnie od używanego urządzenia. Elastyczność interfejsu zwiększa użyteczność aplikacji i umożliwia efektywne zarządzanie zadaniami.

V.2 Wnioski

Aplikacja "ToDo List App" stanowi kompleksowe narzędzie do zarządzania zadaniami, łączące w sobie różnorodne funkcje, począwszy od podstawowych, takich jak dodawanie zadań, aż po bardziej zaawansowane, takie jak generowanie wykresów czy obsługa powiadomień. To sprawia, że użytkownik ma szerokie pole elastyczności do dostosowywania narzędzia do swoich indywidualnych potrzeb i preferencji.

Kolejnym istotnym aspektem jest skupienie na użytkowniku, co przejawia się w starannie zaprojektowanym interfejsie graficznym i responsywności aplikacji. Dzięki temu korzystanie z niej jest dość płynne i przyjemne. Intuicyjne przyciski i funkcje umożliwiają także szybkie przyswajanie obsługi przez nowych użytkowników. Przyciski z podobnymi funkcjami są umieszczone obok siebie, co jest wygodne dla użytkownika.

Baza danych oparta na SQLite przyczynia się do konsystencji i trwałości danych. Operacje na bazie danych umożliwiają efektywne zarządzanie zadaniami, jednakże jeśli chodzi o dodawanie kilku użytkowników nie udało się połączyć tabel co prowadzi do ograniczenia zastosowań tej aplikacji.

Ciekawą funkcjonalnością jest dynamiczne dostosowywanie priorytetów w oparciu o aktualny kontekst i obciążenie. Pozwala to elastycznie zarządzać ważnością zadań, dostosowując priorytety do bieżących potrzeb, co z kolei zwiększa efektywność planowania.

Generowanie wykresów na podstawie danych z listy zadań dostarcza użytkownikowi czytelną reprezentację postępu i struktury zadań. Wizualizacja ta ułatwia identyfikację obszarów wymagających uwagi oraz śledzenie ogólnej wydajności.

Możliwość zapisywania i importowania zadań z plików o różnych formatach czyni aplikację bardziej interoperacyjną, pozwalając użytkownikom dostosować sposób przechowywania danych do swoich preferencji lub wysyłania swojej listy zadań np. do szefa w firmie lub po prostu do znajomych.

Funkcja ustawiania powiadomień skutecznie wspiera użytkowników w przypominaniu o zbliżających się zadaniach, wpływając pozytywnie na terminowość realizacji zadań oraz efektywne zarządzanie czasem. Minusem jest to, że nie zadziałały powiadomienia systemowe, co też nie do końca pozwala sprawdzić całą poprawność.

Ochrona przed błędami, takie jak walidacja danych wejściowych, stanowi dodatkową warstwę bezpieczeństwa, minimalizując potencjalne błędy użytkownika. Komunikaty ostrzegawcze i informacyjne pozwalają szybko zrozumieć ewentualne problemy i skutecznie je rozwiązać.

Co do logowania i funkcji hashującej działa poprawnie, aczkolwiek problem polegał na połączeniu tabel, co ostatecznie nie pozwoliło jeszcze bardziej rozbudować aplikacji.

Podsumowując, "ToDo List App" to nie tylko narzędzie do zarządzania zadaniami, ale kompleksowa aplikacja, która wspiera elastyczność, efektywność i komfort użytkownika, odpowiadając na różnorodne potrzeby użytkownika.

V.3. Plan na możliwy rozwój w przyszłości

W przyszłości:

- Zdecydowanie naprawa bazy danych, po to aby była funkcja logowania się do aplikacji
- Unowocześnienie GUI- nowszy design
- Możliwość przeniesienia tego i zrobienie z tego aplikacji mobilnej
- Współpraca z kalendarzem na telefonie/ z innymi aplikacjami
- dostosowanie i połączenie aplikacji np. ze smartwatchem