**Libra**

Library Management

Arkadiusz Janus

1. **Wprowadzenie**

Libra jest to prosty program napisany w języku Python służący do łatwiejszego zarządzania dużym zbiorem książek. Przetrzymywanie całej bazy danych w jednym pliku bez możliwości konwersji jest męczące i niebezpieczne. Libra rozwiązuje te problemy.

W programie nie było miejsca na zastosowanie algorytmów które mógłby znacząco usprawnić działanie całości. Do samej implementacji wykorzystałem wszystkie dotąd poznane możliwości języka Python. Pojedyncza książka jest reprezentowana jako pojedynczy obiekt klasy ***Book()*** (plik book.py)***,*** z wymaganymi atrybutami i metodami pozwalającymi na dostęp do nich i ich modyfikację bez konieczności bezpośredniego dostępu. Główna funkcjonalność jest skupiona w pliku ***library.py*** w klasie ***Library().*** To tutaj wszystkie obiekty są odpowiednio dodawane, usuwane, aktualizowane, importowane, eksportowane, wyszukiwanie czy zwracane w postaci listy.

Sam interfejs programu jest na tyle przejrzysty i zrozumiały że obsługa programu nie sprawi problemu zaawansowanym użytkownikom jak i początkującym.

1. **Opis interfejsu**
   1. **Menu Główne**

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jest to pierwszy ekran który widzimy po uruchomieniu programu i do którego będziemy wracać po wykonaniu wszystkich operacji. Możemy z niego przejść do wszystkich zaimplementowanych funkcjonalności programu.

Dane opcje wybieramy za pomocą odpowiadających cyfr stojących przy nich. Wpisując konkretną liczbę z klawiatury i naciśnięcie przycisku **ENTER** zostaniemy przeniesieni do odpowiedniego ekranu danej funkcji, który wskaże nam dalsze działania lub wynik.

* 1. **Dodawanie nowej książki**

Po wybraniu pierwszej opcji z menu głównego, zostajemy natychmiastowo przeniesieni do ekranu dodania nowej książki która prosi nad o podanie wymaganych danych.

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

Po wpisaniu wymaganej wartości, naciskamy **ENTER,** aby wpisać koleją.

**Obraz zawierający tekst

Opis wygenerowany automatycznieObraz zawierający tekst

Opis wygenerowany automatycznieObraz zawierający tekst

Opis wygenerowany automatycznie**

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

Ostatnim ekranem jest informacja o tym czy udało nam się dodać książkę czy nie. Jeśli wypełniliśmy wszystkie pola poprawnie to nasz książka już znajduje się w bazie. Po wszystkim naciskamy **ENTER**, aby móc powrócić do menu głównego.

* 1. **Usuwanie książki**

Po wybraniu drugiej opcji z menu głównego, zostajemy przeniesieni do ekranu usunięcia wybranej książki. W tym ekranie możemy to zrobić na dwa sposoby.

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

* + 1. **Wyszukując po tytule**

Tutaj jesteśmy proszeni o podanie tytułu, który jeśli znajduje się w bazie danych to zostanie usunięty.

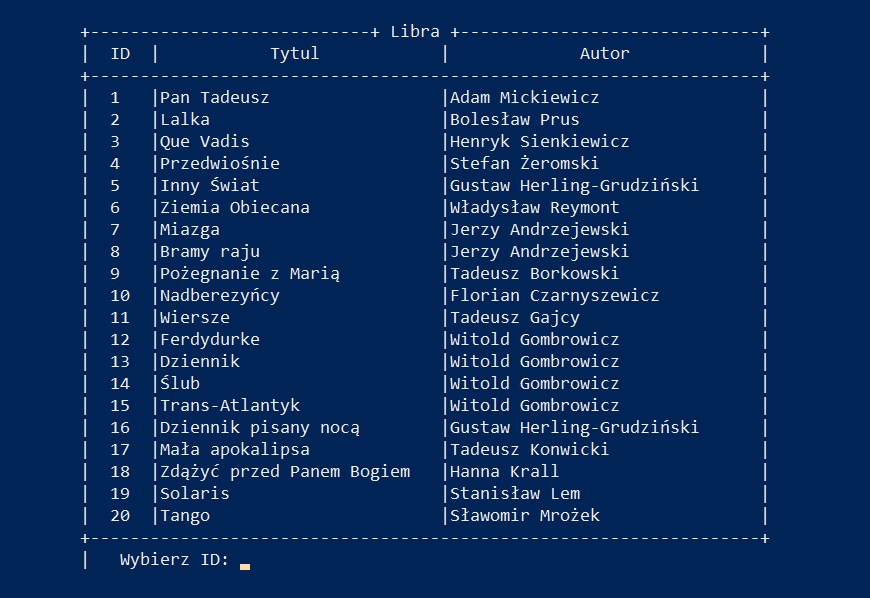
**Obraz zawierający tekst

Opis wygenerowany automatycznieObraz zawierający tekst

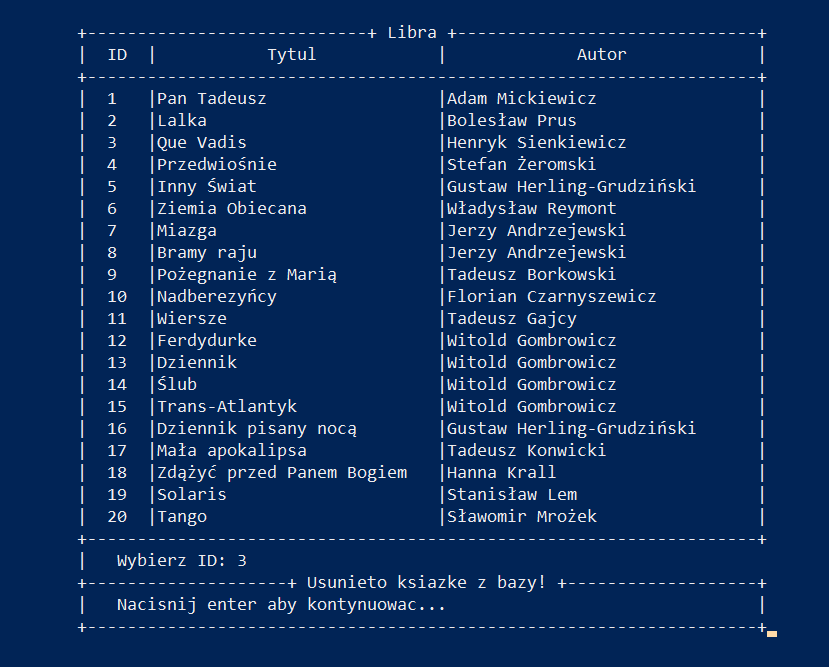
Opis wygenerowany automatycznie**

* + 1. **Wyszukując po ID**

Natomiast na tym ekranie mamy przed sobą cała listę wszystkich pozycji zawartych w bazie, dzięki czemu jesteśmy w stanie odszukać odpowiednią książkę i wpisać jej numer ID.

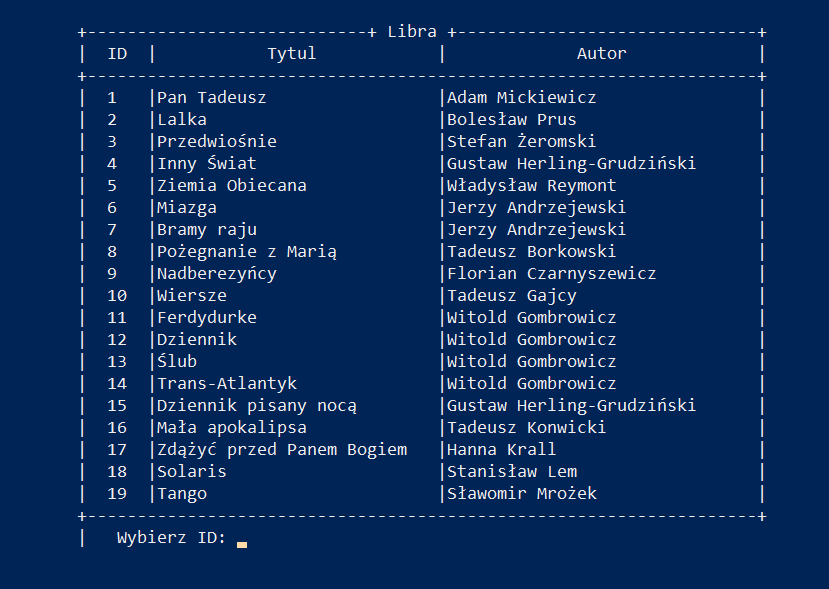


Wybór potwierdzamy naciśnięciem klawisza **ENTER**.

****

Po wykonaniu operacji naszym oczom ukazuje się komunikat potwierdzający wykonanie zadania

Możemy przejść jeszcze raz do tego ekranu aby faktycznie przekonać się o tym że książka zniknęła z naszej bazy.

****

* 1. **Wyszukiwanie książki w bazie**

Program również oferuje możliwość pokazania bardziej szczegółowego opisu danej pozycji. Dodając nową książkę widzieliśmy rubrykę „Opis” która nie była wyświetlana na poprzednich ekranach, gównie aby poprawić czytelność, ponieważ podczas usuwania pozycji z listy, interesuje nas głównie jego tytuł i autor. Tutaj możemy się zapoznać z ciekawszymi detalami książki.

W tym miejscu znajduje się również możliwość edycji danych wybranej książki, jak również kolejny sposób na usunięcie jej z bazy.

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

* 1. **Edycja danych książki**

Będąc w ekranie szczegółów książki, po wybraniu 1 opcji pojawia nam się okno w którym mamy możliwość wyboru który atrybut chcielibyśmy zmodyfikować. Możemy dokonać edycji każdej wartości (poza ID).

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

**Obraz zawierający tekst

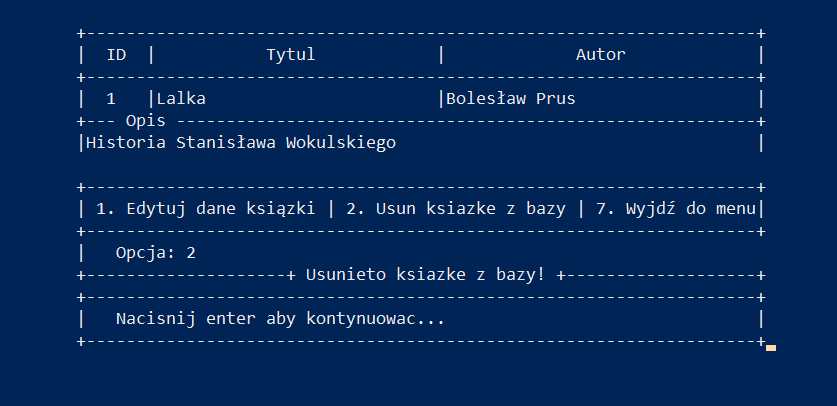
Opis wygenerowany automatycznie**

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

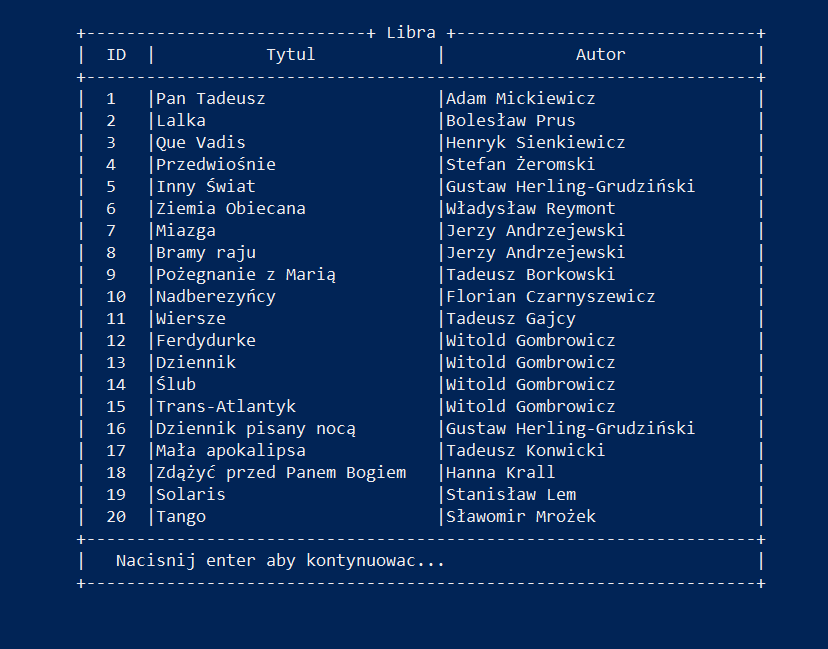
* 1. **Usuwanie danej książki z listy**

Mamy tutaj kolejną możliwość pozbycia się książki z bazy danych. Wystarczy wyszukać jej nazwę i wybrać drugą opcję.

****

* 1. **Listowanie wszystkich pozycji w bazie**

Aplikacja posiada również możliwość wypisania wszystkich pozycji zawartych w bazie danych. Przy 20 pozycjach całość jest wyświetlana na terminalu o rozmiarze 80x25, przy większej ilości wymagane będzie przewijanie całej listy. Naciśniecie przycisku **ENTER** spowoduje powrót do menu głównego.

****

* 1. **Importowanie z pliku**

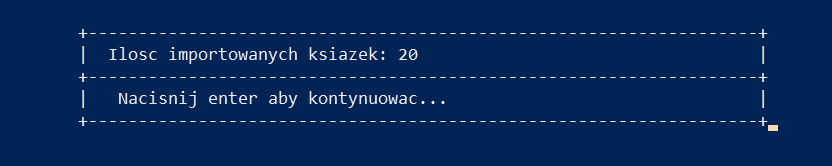
W aplikacji jedną z bardziej użytecznych opcji aplikacji jest importowanie danych z pliku. Zamiast wpisywać od nowa wszystkie pozycje za każdym uruchomieniem możemy skorzystać z możliwości już gotowego pliku, który bez problemu możemy zapisać w naszej bazie. Program obsługuje 2 najpopularniejsze rozszerzenia:

* JSON
* CSV (dane muszą być rozdzielone średnikami)

Aby zaimportować dane z pliku należy wybrać odpowiednią opcję, a następnie podać pełną nazwę pliku (jeśli znajduje się w tym samym folderze, lub pełną lokalizację jeśli znajduje się w innym miejscu) wraz z odpowiednim rozszerzeniem. W przypadku próby podania nieodpowiedniego rozszerzenia pojawi się komunikat o błędzie.

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

****

* 1. **Eksportowanie bazy do pliku**

Mając już możliwość importu z pliku, nie można byłoby zapomnieć o eksporcie który będzie pozwalał przechować w bezpieczny sposób aktualny stan bazy danych. Mamy tutaj dostępne 3 najpopularniejsze formaty przechowywania tego typu danych, są to:

* JSON
* CSV
* TXT – formą jest podobny od CSV, z tym że zamiast średników są przecinki

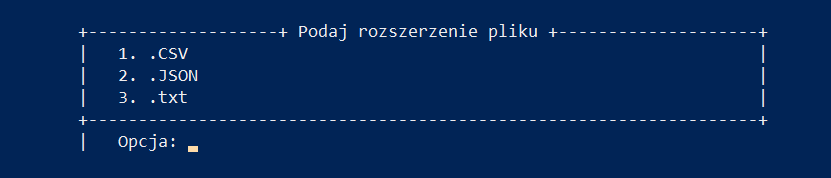
Mając tą funkcjonalność jesteśmy w stanie wykorzystać ten program jako konwerter danych z jednego formatu na drugi.

Aby eksportować dane z pliku wystarczy podać nazwę (aby zapisać w folderze w którym znajduje się uruchomiony program) lub lokalizacje z nazwą (aby zapisać w innym miejscu). W tym miejscu nie jest wymagane podanie rozszerzenia.

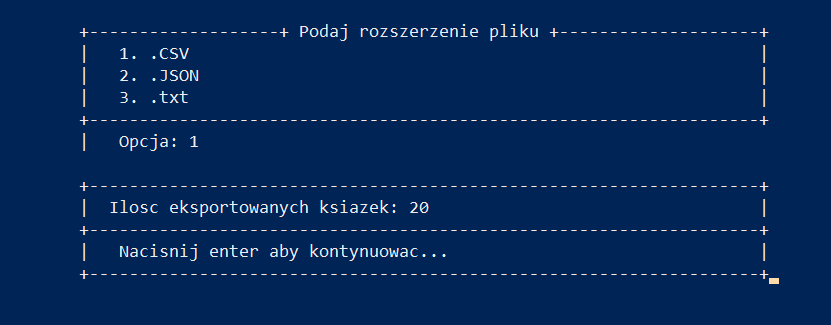
**Obraz zawierający tekst

Opis wygenerowany automatycznie**

Dopiero następnym krokiem jest wybór odpowiadającego nam rozszerzenia.

****

Po wyborze którego otrzymamy komunikat o ilości wyeksportowanych pozycji, a w wybranej przez nas lokalizacji powinien pojawić się nowy plik zawierający te książki.

****

1. **Uwagi na temat implementacji**

Program składa się osobno z części implementującej wszystkie operacje i części interfejsu dla użytkownika, dzięki czemu jest otwarty na możliwość rozbudowy o profesjonalny interfejs graficzny, lub być zaimportowany jako część większego modułu.

Funkcjonalność całej aplikacji jest podzielona na kilkanaście podprogramów które mogą działać również osobno, a odpowiednio połączone oferują wyżej opisaną funkcjonalność. Zadbałem również o odpowiednią strukturę i porządek. Dzięki czemu w głównej lokalizacji znajdują się najpotrzebniejsze skrypty, a pozostałe są umieszczone w odpowiednich pod folderach.

W folderze **test** znajdują się skrypty testujące całą implementację, wykorzystujące do tego moduł **unittest**. Co dla ***book.*py** kod jest testowany osobno, to dla ***library.py***, nie dało się tego zrobić w ten sposób ponieważ sama klasa jest uzależniona od obiektów Book, jednakże pozwoliło to na dokładne sprawdzenie współdziałania całej struktury. W folderze znajduje się również kolejny pod folder **test\_files** zawierający przykładowe pliki które z danymi pozwalające przetestować całość funkcjonalności.

Natomiast w folderze **views** zostały umieszczone wszystkie skrypty dotyczące interfejsu użytkownika. Każdy plik odpowiada za swoją własną funkcjonalność. Pozwoliło to ograniczyć zakres bezpośredniego dostępu do poszczególnych zasobów klas wewnętrznych. Użytkownikowi jest udostępniona tylko wąska część całości dostępu, aby nie udało mu się nadużyć swoich uprawnień.

Przechodząc do samej implementacji, spójrzmy w jaki sposób wyglądają nasze klasy na których operujemy i metody które nam w tym pomagają. Zacznijmy może od najmniejszego elementu w naszej układance, a mianowicie samej książki która jest reprezentowana przez klasę ***Book().***

**book.py**

**\_id = 1**

**def \_\_init\_\_(self, title,**

**authors\_firstname, authors\_surname,**

**description ):**

**if len(title.replace(" ", "")) < 3:**

**raise ValueError**

**self.ID = Book.\_id**

**self.title = title**

**self.authors\_firstname = authors\_firstname**

**self.authors\_surname = authors\_surname**

**self.description = description**

**Book.\_id += 1**

Konstruktor samej klasy jest wyjątkowo prosty, składa się tylko z najpotrzebniejszych rzeczy, czyli przypisaniu odpowiednich wartości do danego obiektu. Z każdym nowym utworzonym elementem jest zwiększania zmienna **\_id,** która odpowiada za specjalny identyfikator obiektu w całej bazie. W swojej implementacji przyjąłem że spośród wszystkich możliwych pustych atrybutów, akurat tytuł musi zawsze zostać podany i dlatego tylko dla niego jest sprawdzana możliwość bycia stringiem za krótkim na tytuł. Dla reszty założyłem że istnieją książki których autor może być po prostu nieznany.

Pozostałym kodem klasy są metody służące do uzyskiwania dostępu do tych atrybutów i ich aktualizowania, np. :

book.py

def get\_ID(self):

        return self.ID

def update\_ID(self, new\_ID):

        self.ID = new\_ID

Posiadając możliwość tworzenia odpowiedniej ilości obiektów reprezentującej książkę potrzebowaliśmy jakiegoś zbiornika który pozwoliłby nam przechowywać, czyli pewnego rodzaju biblioteki, klasy pozwalającej operować na całym zbiorze. Taką funkcjonalność oferuje klasa ***Library().***Jej konstruktorem jest po prostu utworzeni Pythonowej listy która będzie przechowywać wszystkie pozycje.

***Dodawanie książki do bazy***

library.py

def addBook(self, book):

        if self.size() != 0:

            \_id = self.database[-1].get\_ID()

        if self.size() == 0:

            \_id = 0

        book.update\_ID(\_id+1)

        ...

Dodawanie książki do bazy opiera się na operacji zaktualizowania listy o nowy element przy pewnych dodatkowych założeniach. Należało osobno rozpatrzeć różne przypadki nadania kolejnego numeru ID każdemu z nowo dodanych obiektów, w taki sposób aby zachować ciągłość numeryczną, aby każdy miał o jeden większy niż poprzednik. Sam mechanizm nadający numer ID podczas tworzenia książki niestety się nie sprawdzał.

library.py

def addBook(self, book):

...

is\_not\_in\_database = True

        title = book.get\_title()

        for \_book in self.database:

            if title.lower() == \_book.get\_title().lower():

                is\_not\_in\_database = False

        if is\_not\_in\_database:

            if title != "":

                self.database.append(book)

            return 1

        return -1

Dodatkową subtelnością którą należało uwzględnić podczas dodawania nowej książki było rozpatrzenie czy książka nie znajduje się już w bazie. Funkcja ostatecznie zwracała informacje o tym czy rzeczywiście udało się dodać obiekt, a która była wykorzystywana w interfejsie do wyświetlenia odpowiedniego komunikatu.

add\_book.py

def print\_add\_book\_menu():

    """ Funkcja wyswietlajaca menu dodawania ksiazki do bazydanych """

...

    try:

        result = int(library.addBook(Book(title, authors\_firstname, authors\_surname, description)))

        if result == 1:

            print("""        +--------------------+ Dodano ksiazke do bazy! +--------------------+\n""", end="")

        elif result == -1:

            print("""        +-------------------+ Ksiazka jest juz w bazie! +-------------------+\n""", end="")

    except ValueError:

        print("""        +-------------+ Nie mozna dodac ksiazki bez tytulu! +---------------+\n""", end="")

W zależności od otrzymanego wyniku operacji dodawania, zostaje wypisany odpowiedni komunikat. Pamiętając że klasa ***Book()***rzuca wyjątkiem ***ValueError*** przy próbie utworzenia obiektu nie spełniającego wymagań, zostaje on tutaj złapany i obsłużony.

***Usuwanie książk z bazy***

library.py

 def removeBook(self, book):

        """ Funkcja sluzaca do usuwania ksiazek z bazy danych """

        if book is not None:

            \_id = book.get\_ID()

            if book in self.database:

                self.database.remove(book)

            for i in range(\_id-1, self.size()):

                new\_id = self.database[i].get\_ID()-1

                self.database[i].update\_ID(new\_id)

Usuwanie książki również jest prostą operacją. Cała funkcjonalność opiera się na sprawdzeniu czy faktycznie się znajduje w bazie danych i odpowiednim zmniejszeniu indeksu wszystkich następnych pozycji, aby zachować ciągłość.

Plik interfejsu korzysta jednak nie tylko z samej metody usuwania, lecz do poprawnego działania potrzebuje trzech dodatkowych metod które pozwalają mu znaleźć odpowiednią pozycje.

library.py

def find\_book\_by\_title(self, title):

        """ Funkcja wyszukujaca ksiazke w bazie danych za pomoca tytulu"""

        for book in self.database:

            if book.get\_title().lower() == title.lower():

                return book

        return None

def find\_book\_by\_ID(self, ID):

        """ Funkcja wyszukujaca ksiazke w bazie danych za pomoca ID """

        for book in self.database:

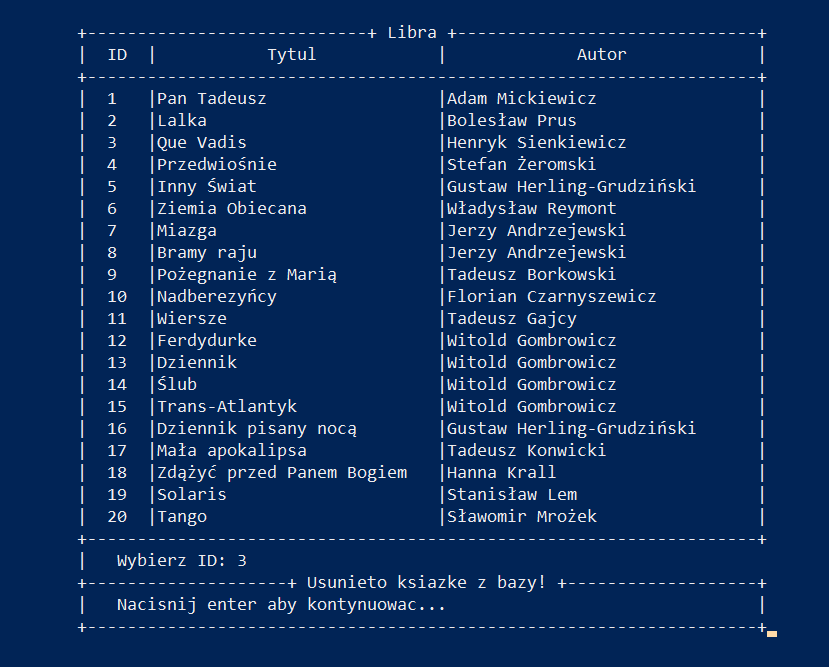
            if book.get\_ID() == int(ID):

                return book

        return None

Metody **find\_book\_by\_title()i find\_book\_by\_ID()** przeszukują całą bazę danych w poszukiwaniu odpowiadającego tytułu lub ID. W przypadku znalezienia, cały obiekt jest zwracany, natomiast w przypadku przeciwnym, zwracany jest pusty obiekt **None.**

Plik interfejsu **find\_book.py,** w sposobie wyszukiwania po ID wykorzystuje również funkcję interfejsu **print\_all\_books()** wypisującą wszystkie pozycje zawarte w bazie w formie czytelnej tabeli, dzięki czemu łatwiej lokalizować daną książkę której chcielibyśmy się pozbyć.

****

***Wyszukiwanie książki w bazie***

Książki są wyszukiwane w całej bazie danych za wyżej opisanej funkcji **find\_book\_by\_title().**

Znacznie ciekawszym aspektem jest pozornie prozaiczna czynność wypisania szczegółowych informacji samej książki.

Co samo ID, tytuł i autor są wyświetlane w sposób dokładnie taki sam jak widzimy na powyższej grafice dla jednego wiersza, to jednak już sam opis pozycji wymagał więcej zaangażowania. Aby móc zachować ustalone wcześniej formatowanie które obejmowało maksymalnie 67 znaków wymaganym było dopisanie odpowiedniej funkcji która będzie dzielić opis przekraczający tę wartość na części o jednakowej długości. Taką funkcję w tym programie pełni:

find\_book.py

def split\_string\_into\_chunks(string):

    """ Funkcja dzieli lancuchy znakowe na partie po 67 znakow """

    lenght\_of\_chunk = 67

    chunks = [string[i:i+lenght\_of\_chunk] for i in range(0, len(string), lenght\_of\_chunk)]

    return chunks

Aby wypisać opis w tej formie należało rozpatrzyć kilka przypadków aby interfejs w żaden sposób nie miał okazji się „rozjeżdżać” i tak dzięki temu mamy osobne sytuacje dla krótszych opisów, jak i dłuższych które uwzględniają sytuację ostatniej linii która nie rozpocznie nowej.

find\_book.py

def print\_find\_book\_menu():

...

            description = book.get\_description()

            if type(description) == type(list()):

                description\_string = ""

                for i in description:

                    description\_string += str(i)

            else:

                description\_string = description

            if len(description\_string) > 67:

                description\_string =

split\_string\_into\_chunks(description\_string)

                for i in range(len(description\_string)):

                    if i != len(description\_string)-1:

                        print(f"        |{description\_string[i]:<67}|")

                    else:

                        print(f"        |{description\_string[i]:<67}|", end="")

            else:

                print(f"        |{description\_string:<67}|")

...

***Edytowanie danych książki w bazie***

Dostęp do możliwości edytowania wymaga uprzedniego znalezienia jej w bazie za pomocą tytułu. To otwiera nad dostęp do funkcjonalności udostępnianej przez interfejs **print\_update\_book\_menu(),** który wykorzystuje metodę klasy **Library():**

library.py

    def update\_book\_info(self, book, \*args):

        """ Funkcja pozwalajaca modyfikowac poszczegolne atrybuty ksiazki """

        if args[1] == "":

            return -1

        if 'title' in args:

            try:

                book.update\_title(args[1])

            except ValueError:

                return -1

        elif 'authors\_firstname' in args:

            book.update\_authors\_firstname(args[1])

        elif 'authors\_surname' in args:

            book.update\_authors\_surname(args[1])

        elif 'description' in args:

            book.update\_description(args[1])

        else:

            pass

Która w zależności od podanego argumentu modyfikuje odpowiednie wartości książki wywołując odpowiednie metody obiektu ***book***, np.

book.py

def update\_title(self, new\_title):

        if len(new\_title.replace(" ", "")) < 3:

            raise ValueError

        self.title = new\_title

Która również blokuje możliwość pozostawienia książki bez tytułu i zgłasza wyjątek w przypadku takiej próby. Wyjątek oczywiście zostaje złapany i odpowiednio obsłużony w kodzie głównej funkcji.

find\_book.py

def print\_update\_book\_menu(book):

...

elif option == 1:

        new\_title = input("        |   Podaj nowy tytuł: ")

        result = library.update\_book\_info(book, "title", new\_title)

        if result is not None and int(result) == -1:

            print("        |   Niepoprawny tytuł!")

...

***Eksportowanie całej bazy do pliku***

Program oferuje również możliwość eksportu wszystkich pozycji zawartych w bazie do pliku z odpowiednim rozszerzeniem. Za obsługę tej funkcjonalności odpowiada interfejs **print\_export\_books\_menu(),** wraz z metodami:

library.py

def export\_books\_to\_csv\_file(self, file\_name):

        """ Funkcja eksportujaca baze danych do pliku z rozszerzeniem .csv """

        file = open(file\_name+".csv", "w", encoding="utf-8", newline="")

        csv\_writer = csv.writer(file, delimiter=";")

        counter = 0

        header = "ID;title;authors\_firstname;authors\_surname;description"

        csv\_writer.writerow([header])

        for book in self.database:

            csv\_writer.writerow([book.get\_ID(), book.get\_title(),

                             book.get\_authors\_firstname(), book.get\_authors\_surname(),

                             book.get\_description()])

            counter += 1

        file.close()

        return counter

def export\_books\_to\_json\_file(self, file\_name):

        """ Funkcja eksportujaca baze danych do pliku z rozszerzeniem .json """

        file = open(file\_name+".json", "w", encoding="utf-8")

        counter = 0

        file.write("[\n")

        for i in range(len(self.database)):

            json.dump(self.database[i].\_\_dict\_\_, file, ensure\_ascii=False, indent=4)

            if i != len(self.database)-1:

                file.write(",")

            counter += 1

        file.write("\n]")

        file.close()

        return counter

def export\_books\_to\_txt\_file(self, file\_name):

        """ Funkcja eksportujaca baze danych do pliku z rozszerzeniem .txt """

        file = open(file\_name+".txt", "w", encoding="utf-8")

        counter = 0

        header = "ID,title,authors\_firstname,authors\_surname,description"

        file.write(header)

        for book in self.database:

            file.write(f"\n{book.get\_ID()},{book.get\_title()},{book.get\_authors\_firstname()}, {book.get\_authors\_surname()}, {book.get\_description()}")

            counter += 1

        file.close()

        return counter

Wszystkie opierają się na tym samym sposobie działania. W argumentach otrzymują nazwę pliku do którego mają być eksportowane wszystkie obiekty, a którą modyfikują o odpowiednie rozszerzenie. Do poszczególnych formatów wykorzystałem odpowiednie narzędzia które pozwoliły optymalnie wykonać zapis do pliku. Dla formatów:

- CSV – wykorzystałem metodę csv.writer(), z modułu csv znajdującego się w podstawowej bibliotece pythona

- JSON – wykorzystałem metodę json.dump(), z modułu json również znajdującego się w podstawowej bibliotece pythona

- TXT – tutaj po prostu zastosowałem zwykłą metodę file.write(), zapisującą prosto do otwartego pliku

Zapis każdej z metod jest dostosowany do warunków stawianych przed dany format, jedynie w rozszerzeniu **.txt,** mieliśmy wolną rękę na sposób reprezentacji. Uznałem ze najczytelniejszym z nich będzie zmodyfikowany format **CSV**, w którym średniki zostały zastąpione przecinkami.

Po każdej operacji plik do zapisu jest zamykany i zwracany jest licznik który zawiera informację o ilości eksportowanych rekordów, która jest wypisywana na ekran za pomocą interfejsu.

export\_books.py

def print\_export\_books\_menu():

  ...

try:

        option = int(input("        |   Opcja: "))

    except ValueError:

        print("Niepoprawna wartosc!")

    if option == 1:

        counter = library.export\_books\_to\_csv\_file(file\_name)

        print(

        f"""\r

        +-------------------------------------------------------------------+

        |  Ilosc eksportowanych ksiazek: {counter:<16}                   |

        +-------------------------------------------------------------------+"""

        )

    elif option == 2:

        counter = library.export\_books\_to\_json\_file(file\_name)

        print(

        f"""\r

        +-------------------------------------------------------------------+

        |  Ilosc eksportowanych ksiazek: {counter:<16}                   |

        +-------------------------------------------------------------------+"""

        )

    elif option == 3:

        counter = library.export\_books\_to\_txt\_file(file\_name)

        print(

        f"""\r

        +-------------------------------------------------------------------+

        |  Ilosc eksportowanych ksiazek: {counter:<16}                   |

        +-------------------------------------------------------------------+"""

        )

    else:

        print("        |   Niepoprawna opcja!")

***Importowanie danych z pliku***

Import jest zrealizowany w bardzo podobny sposób co poprzednie funkcjonalności. Operacja posiada własny interfejs który wykorzystuje dwie metody biblioteki wykonujące wszystkie czynności.

library.py

def import\_books\_form\_csv\_file(self, file\_name):

        """ Funkcja importujaca baze danych ksiazek z pliku .csv """

        try:

            file = open(file\_name, "r", encoding="utf-8")

        except FileNotFoundError:

            raise FileNotFoundError

        csv\_reader = csv.reader(file)

        ...

    def import\_books\_from\_json\_file(self, file\_name):

        """ Funkcja importujaca baze danych ksiazek z pliku .json """

        try:

            file = open(file\_name, "r", encoding="utf-8")

        except FileNotFoundError:

            raise FileNotFoundError

        ...

Same jednak funkcje importu wymagały zadbania o większą ilość szczegółów których eksport nie wymagał. Jednym z nich jest np. sprawdzenie czy plik który chcemy zaimportować w ogóle istnieje. Jeśli program nie może znaleźć odpowiedniego pliku zgłasza wyjątek, który jest łapany i obsługiwany w interfejsie:

import\_books.py

def print\_import\_books\_menu():

    ...

        try:

            counter = library.import\_books\_form\_csv\_file(file\_name)

            print(

        f"""\r

        +-------------------------------------------------------------------+

        |  Ilosc importowanych ksiazek: {counter:<16}                    |

        +-------------------------------------------------------------------+"""

        )

        except FileNotFoundError:

            print("        |   Nieznaleziono takiego pliku!")

    elif file\_extension == "json":

        try:

            counter = library.import\_books\_from\_json\_file(file\_name)

            print(

        f"""\r

        +-------------------------------------------------------------------+

        |  Ilosc importowanych ksiazek: {counter:<16}                    |

        +-------------------------------------------------------------------+"""

        )

        except FileNotFoundError:

            print("        |   Nieznaleziono takiego pliku!")

    else:

        print("""        +---------------------+ Importuj dane z pliku +---------------------+

        |   Niepoprawne rozszerzenie!                                      |""")

Cały program jest również opakowany w zabezpieczenia przed podaniem niepoprawnej wartości:

import\_books.py

def print\_import\_books\_menu():

    print("""

        +---------------------+ Importuj dane z pliku +---------------------+"""

    )

    try:

        file\_name = input("        |   Podaj nazwe pliku (z rozszerzeniem): ")

    except ValueError:

        print("Niepoprawna wartosc!")

    cls()

...

W programie wielokrotnie przewijała się również funkcja cls(),która odpowiadała za czyszczenie całego terminalu aby zwiększyć czytelność całości programu.

clear\_terminal.py

def cls():

    """ Funkcja czyszczaca caly terminal """

    os.system('cls' if os.name=='nt' else 'clear')

***Główny interfejs i wywołanie***

Całość programu jest obsługiwana za pomocą głównego interfejsu znajdującego się w menu.py a który jest wywoływany przez główny plik main.py w którym zaimportowane są wszystkie wcześniej wymienione interfejsy, klasy i funkcje.

main.py

from book import \*

from library import \*

from lib\_creator import library

from clear\_terminal import cls

from views.menu import print\_main\_menu

from views.add\_book import print\_add\_book\_menu

from views.remove\_book import print\_delete\_book\_menu

from views.print\_all\_books import print\_all\_books

from views.find\_book import print\_find\_book\_menu

from views.export\_books import print\_export\_books\_menu

from views.import\_books import print\_import\_books\_menu

from goon import go\_on

...

def main():

    """ Funkcja odpowiadajaca za cala dzialanie calego programu """

    while True:

        cls()

        print\_main\_menu()

        try:

            option = int(input("        |   Opcja: "))

        except ValueError:

            print("Niepoprawna wartosc!")

            continue

        choose\_option(option)

        go\_on()

Całe działanie programu opiera się na nieskończonej pętli ***while*** która może zostać zakończona albo przez wybranie odpowiedniej opcji, albo przez przerwanie działania całego programu z poziomu systemu (np. ctrl + c w systemie Windows).

W znacznej większości plików występował również plik odpowiadający za tworzenie całej biblioteki:

lib\_creator.py

from library import \*

from book import \*

library = Library()

Dzięki czemu zawsze mieliśmy pewność że operujemy na tej samej bazie danych.

Dla całego programu jest stworzony plik wykonawczy **run.py** którego wywołanie powoduje uruchomienie całego programu.

run.py

from main import main

# """ Plik uruchamiajacy caly program """

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. **Testy**

Całym projekcie nie można zapomnieć o przetestowaniu wszystkich funkcjonalności. Poza samymi testami empirycznymi (przez zwykłe korzystanie z programu), zadbałem również o testy jednostkowe poszczególnych funkcjonalności. W folderze **test** znajdują się dwa pliki. Jeden z nich testuje funkcjonalność klasy ***Book***, drugi natomiast klasy ***Library.***

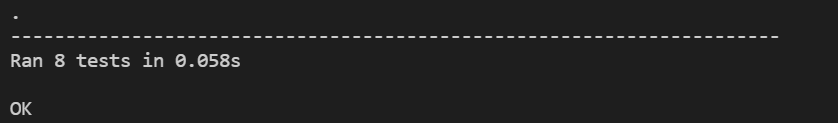
W pierwszy test badał wszystkie możliwości utworzenia danego obiektu, w tym sytuacje kiedy obiektu utworzyć się nie dało i zgłaszany był wyjątek. Dodatkowo sprawdzane zostały wszystkie możliwe metody udostępniające dane, jak i aktualizujące je.

W drugim teście zostały gruntownie przetestowane wszystkie funkcje odpowiadające za tworzenie nowej bazy, dodawanie do niej książek, usuwanie ich, pobieranie całej bazy, eksportowanie i importowanie bazy z/do pliku. W pod folderze **test\_files** znajdują się pliki pomocnicze które pozwalają przeprowadzić całość testów. Dla wszystkich zaproponowanych przypadków sprawdzenia funkcjonalności nie zostały wykryte żadne błędy.

Aby uruchomić wszystkie testy należy w głównym folderze wpisać:

python -m unittest

I wszystkie możliwe przypadki które potrafiłem przewidzieć przechodzą:



Przeprowadziłem jedynie testy samych funkcjonalności zakładając że interfejsy dbają jedynie o ich obsługę i warstwę graficzną. One natomiast zostały gruntownie sprawdzone podczas testów empirycznych, podczas których nie udało mi się znaleźć żadnych błędów.

1. **Uruchomienie programu**

Wszystkie funkcjonalności programu zostały zaimplementowane samodzielnie z wykorzystaniem podstawowych modułów zawartych w podstawowej bibliotece języka Python, dzięki czemu nie jest wymagana instalacja dodatkowych pakietów. Program był testowany dla pythona w wersji **3.9.1**, więc ta wersja i wyższe są zalecane.

Aby uruchomić program należy przejść w terminalu do głównej lokalizacji programu, a następnie wpisać:

python run.py

1. **Podsumowanie**

Libra to gruntownie przetestowana aplikacja ułatwiająca zarządzanie dużymi zasobami danych. Cała funkcjonalność została zaimplementowana, aby zapewnić maksymalne bezpieczeństwo zestawu danych. Interfejsy natomiast cechują się maksymalnym wykorzystaniem funkcjonalności, przy jednoczesnym zachowaniu przejrzystości dzięki której mogą z niej korzystać nie tylko doświadczeni użytkownicy, lecz również początkujący. Cały system został choć jest w wersji finalnej, to jest zaprojektowany w taki sposób że jest otwarty na dalszą rozbudowę, więc z łatwością można wykorzystać jego funkcjonalność w innych projektach czy rozbudować go o nowy interfejs.

1. **Wykorzystane źródła**

* [**https://ufkapano.github.io/algorytmy/index.html**](https://ufkapano.github.io/algorytmy/index.html)
* [**https://realpython.com/**](https://realpython.com/)
* **https://stackoverflow.com/**