```
    _    _ _                     ____  ____  
   / \  | | |__  _   _ _ __ ___ |  _ \| __ ) 
  / _ \ | | '_ \| | | | '_ ` _ \| | | |  _ \ 
 / ___ \| | |_) | |_| | | | | | | |_| | |_) |
/_/   \_\_|_.__/ \__,_|_| |_| |_|____/|____/ 
                                  version 1.0
```

1. **Wprowadzenie do tematu**
   Tematem omawianego projektu było przygotowanie w języku Python programu umożliwiającego przechowywanie oraz zarządzanie bazą albumów muzycznych.

   Gotowa aplikacja miała uwzględniać możliwość odczytu/zapisu bazy z pliku, dodawanie, usuwanie, wyświetlanie zawartości oraz przeszukiwanie.

2. **Opis interfejsu**
   W oparciu o wymagania projektowe utworzony został interfejs zawierający następujące części:

   a) Database Menu

```
+----------------------------------------------------+
|                    DATABASE MENU                   |
|                                                    |
| [1] Create database                                |
| [2] Delete database                                |
| [3] Load database                                  |
| [4] Exit                                           |
+----------------------------------------------------+
```

Jest to pierwsze menu wyświetlane po uruchomieniu programu. Odnajdziemy w nim opcje związane z bazą danych – jej odczyt, tworzenie, czy też usuwanie.

```
================= Create database =================

Specify database name (default: music.db) - type
 'exit' to abort):
```

Po wczytaniu lub utworzeniu bazy, użytkownik zostaje przekierowany do głównego menu, którego funkcjonalność omówiona jest w kolejnym podpunkcie.

```
================= Delete database =================

Databases found:
music.db

Specify database name (default: music.db - type 'exit'
to abort):

================== Load database ==================

Databases found:
music.db

Specify database name (default: music.db - type 'exit'
to abort):
```

W przypadku wybrania opcji wczytania lub usunięcia, dodatkowo prezentowana jest lista odnalezionych baz danych (muszą się one znajdować w folderze z programem).

b) Main Menu

```
+---------------------------------------------------+
|                     MAIN MENU                     |
|                                                   |
| [1] Add album                                     |
| [2] Delete album                                  |
| [3] Search                                        |
| [4] Print collection                              |
| [5] Database manager                              |
| [6] Exit                                          |
+---------------------------------------------------+
```

Po wczytaniu bazy wyświetlane jest menu główne, oferujące wszelkie przewidziane w wymaganiach realizacyjnych opcje, takie jak: dodawanie oraz usuwanie albumów, przeszukiwanie bazy, czy też wyświetlanie jej zawartości.

Pierwszą z funkcjonalności jest dodawanie nowego albumu do bazy. Użytkownik proszony jest kolejno o: podanie artysty, nazwy albumu oraz daty wydania. W przypadku niewypełnienia danego pola, ponownie wyświetlany jest monit o wprowadzenie wymaganych informacji.

```
==================== Add album ====================

Artist: Elvya
Album name: Untold Stories
Release year: 2015
```

Wybranie drugiej z opcji, tj. usuwania albumów, skutkuje wyświetleniem menu oferującego dwie możliwości: pozbywanie się pojedynczych wydawnictw, jak i usuwanie wszystkich albumów danego artysty.

```
+----------------------------------------------------+
|                    DELETE ALBUM                    |
|                                                    |
| [1] Delete a single release                        |
| [2] Delete all albums by an artist                 |
| [3] Go back                                        |
+----------------------------------------------------+
```

Kolejna z pozycji na stronie głównej umożliwia przeszukiwanie zawartości bazy danych poprzez podanie artysty, nazwy albumu lub roku wydania.

```
+----------------------------------------------------+
|                    SEARCH MENU                     |
|                                                    |
| [1] Search by artist                               |
| [2] Search by album name                           |
| [3] Search by release year                         |
| [4] Go back                                        |
+----------------------------------------------------+
```

Ostatnią z opcji okna głównego programu, zaraz przed **[5] Database manager**, służącą do powrotu do zarządzania plikiem bazy danych, jest menu wyświetlania listy płyt.

Albumy wypisane mogą zostać posortowane względem artysty, nazwy wydawnictwa, jak i roku wydania.

```
+----------------------------------------------------+
|                     PRINT MENU                     |
|                                                    |
| [1] Sorted by artist                               |
| [2] Sorted by album name                           |
| [3] Sorted by release year                         |
| [4] Go back                                        |
+----------------------------------------------------+
================= Sorted by artist =================

+-------------------------------+------------------------------+---------------+
| Artist                        | Album name                   | Release year |
+-------------------------------+------------------------------+---------------+
| Cécile Corbel                 | La fiancée                   | 2014          |
| Dead Can Dance                | Aion                         | 1990          |
| Egrimonia                     | Along the Path of Diversity  | 2008          |
| Elane                         | The Fire of Glenvore         | 2004          |
| Elane                         | More Stars                   | 2016          |
| Elane                         | The Silver Falls             | 2008          |
| Elvya                         | Untold Stories               | 2015          |
| Nightwish                     | Century Child                | 2002          |
| Nightwish                     | Imaginaerum                  | 2011          |
| Sarah Blasko                  | Prelusive                    | 2002          |
| The Moon and the Nightspirit  | Ősforrás                     | 2009          |
| Within Temptation             | The Silent Force             | 2004          |
| Within Temptation             | The Unforgiving              | 2011          |
| Within Temptation             | Let Us Burn                  | 2015          |
+-------------------------------+------------------------------+---------------+
```

3. **Szczegóły implementacyjne**

   Implementacja programu oparta została o klasy wyszczególnione poniżej:

   a) **ApplicationState** – określa aktualny stan aplikacji, tj. wyświetlane menu
   b) **DatabaseLayer** – warstwa odpowiadająca za udostępnienie wygodnej formy komunikacji z bazą danych

```python
class DatabaseLayer:
    """ Provides a layer for communication with the database """

    def __init__(self, database='music.db'):
        self.database = database

    def query(self, statements, data=()):
        """ Method used for querying the database """

        with sqlite3.connect(self.database) as connection:
            connection.text_factory = lambda x: unicode(x, "utf-8",
        "ignore")
            connection.row_factory = sqlite3.Row
            cursor = connection.cursor()
            result = cursor.execute(statements, data)
            connection.commit()

        return result
```

W tej klasie wykorzystywana jest funkcjonalność biblioteki sqlite3.

c) **MenuBase** - klasa określająca podstawową funkcjonalność oferowaną przez każdą podstronę menu

```python
class MenuBase:
    """ Represents basic functionality of each menu view """

    def __init__(self):
        self.header = ""
        self.actions = []

    def get_action(self, action_id):
        """ Returns an action for a specified option „""

        action_id = int(action_id)
        action = list(ifilter(lambda a: a["id"] == action_id,
        self.actions))
        return action[0] if len(action) else None

    def print_menu(self):
        """ Self-explanatory - prints the menu „""

        separator = "+" + "-" * 50 + "+"
        print separator

        category = "|"
        cat_spacer = 25 - int(math.ceil(float(len(self.header))/2))
        category += " " * cat_spacer
        category += self.header
        category += " " * cat_spacer

        if len(self.header) % 2:
            category += " |"
        else:
            category += "|"

        print category

        print "|" + 50 * " " + "|"

        for action in self.actions:
            line = "| "
            action = "[" + str(action["id"]) + "] " + action["text"]
            line += action
            line += " " * (49 - len(action))
            line += "|"
            print line

        print separator

    @classmethod
    def print_action_header(cls, action_name):
        """ Prints header for a specified option „""

        spacer = 25 - int(math.ceil(float(len(action_name))/2))

        header = "=" * spacer
        header += " " + action_name + " "
        header += "=" * spacer
        header += "\n"

        print header
```

d) klasy dziedziczące po **MenuBase**:

- Database Menu

```
class DatabaseMenu(MenuBase):
    """ This menu includes the most common operations
        for working with databases """

    def __init__(self):
        self.header = "DATABASE MENU"
        self.actions = [
            {
                "id": 1,
                "text": "Create database",
                "func": self.create_database
            },
            {
                "id": 2,
                "text": "Delete database",
                "func": self.delete_database
            },
            {
                "id": 3,
                "text": "Load database",
                "func": self.load_database
            },
            {
                "id": 4,
                "text": "Exit",
                "func": lambda: sys.exit(0)
            }
        ]

    @classmethod
    def create_database(cls):
    """ Allows user to create database with a specified name """

        name = raw_input("Specify database name (default: music.db)
                         - type 'exit' to abort): ") or "music.db"

        if name == 'exit':
            return DatabaseMenu

        ApplicationState.album_manager = AlbumManager(name)
        return MainMenu

    @classmethod
    def delete_database(cls):
    """ Allows user to delete database with a specified name """

        files = []

        for f in os.listdir("."):
            if f.endswith(".db"):
                files.append(f)

        if not files:
            print "No database files found."
            raw_input("\nPress ENTER to go back to the previous
                     menu... ")
            return DatabaseMenu

        else:
            print "Databases found: "
```

```python
            for f in files:
                print(f)

            print

            name = raw_input("Specify database name (default:
            music.db - type 'exit' to abort): ") or "music.db"

            if name == 'exit':
                return DatabaseMenu

            elif not os.path.exists(name):
                print "The specified file does not exist.\n"
                return None

            os.remove(name)
            return DatabaseMenu

    @classmethod
    def load_database(cls):
        """ Allows user to load database with a specified name """

        files = []

        for f in os.listdir("."):
            if f.endswith(".db"):
                files.append(f)

        if not files:
            print "No database files found."
            raw_input("\nPress ENTER to go back to the previous
                         menu... ")
            return DatabaseMenu

        else:
            print "Databases found: "

            for f in files:
                print(f)

            print

            name = raw_input("Specify database name (default:
            music.db - type 'exit' to abort): ") or "music.db"

            if name == 'exit':
                return DatabaseMenu

            elif not os.path.exists(name):
                print "The specified file does not exist.\n"
                return None

            ApplicationState.album_manager = AlbumManager(name)
            return MainMenu
```

- MainMenu

```python
class MainMenu(MenuBase):
    """ Main menu view """

    def __init__(self):
        self.header = "MAIN MENU"
        self.actions = [
            {
                "id": 1,
                "text": "Add album",
                "func": self.add_album_menu
            },
            {
                "id": 2,
                "text": "Delete album",
                "func": self.delete_album_menu
            },
            {
                "id": 3,
                "text": "Search",
                "func": self.search_menu
            },
            {
                "id": 4,
                "text": "Print collection",
                "func": self.print_collection_menu
            },
            {
                "id": 5,
                "text": "Database manager",
                "func": self.database_manager
            },
            {
                "id": 6,
                "text": "Exit",
                "func": lambda: sys.exit(0)
            }
        ]

    @classmethod
    def add_album_menu(cls):
 """ Method used for adding new albums to the existing database """

        while True:
            artist = raw_input("Artist: ")

            if artist:
                break

        while True:
            album_name = raw_input("Album name: ")

            if album_name:
                break

        while True:
            release_year = raw_input("Release year: ")

            if release_year.isdigit():
                break

        AlbumManager.add_album(ApplicationState.album_manager,
                              artist, album_name, release_year)
        return MainMenu
    @classmethod
    def delete_album_menu(cls):
        """ Switches the view to Delete Album """
        return DeleteMenu

    @classmethod
    def print_collection_menu(cls):
        """ Switches the view to Print Collection """
        return PrintCollection

    @classmethod
    def database_manager(cls):
        """ Switches the view to Database Manager """
        return DatabaseMenu

    @classmethod
    def search_menu(cls):
        """ Switches the view to Search Menu """
        return SearchMenu
```

- DeleteMenu

```python
class DeleteMenu(MenuBase):
    """ This menu includes options for deleting single or multiple
        albums """

    def __init__(self):
        self.header = "DELETE ALBUM"
        self.actions = [
            {
                "id": 1,
                "text": "Delete a single release",
                "func": self.delete_a_single_release
            },
            {
                "id": 2,
                "text": "Delete all albums by an artist",
                "func": self.delete_all_by_artist
            },
            {
                "id": 3,
                "text": "Go back",
                "func": lambda: MainMenu
            }
        ]

    @classmethod
    def delete_a_single_release(cls):
    """ Allows user to delete a single album from the database """

        while True:
            artist = raw_input("Artist: ")

            if artist:
                break

        while True:
            album_name = raw_input("Album name: ")

            if album_name:
                break
        AlbumManager.delete_album(ApplicationState.album_manager,
                                  artist, album_name)
        return MainMenu

    @classmethod
    def delete_all_by_artist(cls):
    """ Allows user to delete all albums by a specified artist """

        while True:
            artist = raw_input("Artist: ")

            if artist:
                break

        AlbumManager.delete_all_by_artist(
        ApplicationState.album_manager, artist)
        return MainMenu
```

- SearchMenu

```
class SearchMenu(MenuBase):
""" Search Menu delivers various methods for filtering the database
"""

    def __init__(self):
        self.header = "SEARCH MENU"
        self.actions = [
            {
                "id": 1,
                "text": "Search by artist",
                "func": self.search_by_artist
            },
            {
                "id": 2,
                "text": "Search by album name",
                "func": self.search_by_album
            },
            {
                "id": 3,
                "text": "Search by release year",
                "func": self.search_by_year
            },
            {
                "id": 4,
                "text": "Go back",
                "func": lambda: MainMenu
            }
        ]

    @classmethod
    def search_by_artist(cls):
        """ Allows user to filter the database by artist """

        while True:
            artist = raw_input("Artist: ")

            if artist:
                break

        print
        AlbumPrinter.print_albums(AlbumManager.get_by_artist(
        ApplicationState.album_manager, artist))
        raw_input("\nPress ENTER to go back to the
                     previous menu... ")
        return SearchMenu

    @classmethod
    def search_by_album(cls):
        """ Allows user to filter the database by album name """

        while True:
            album_name = raw_input("Album name: ")

            if album_name:
                break

        print

        AlbumPrinter.print_albums(AlbumManager.get_by_name(
        ApplicationState.album_manager, album_name))
        raw_input("\nPress ENTER to go back to the
                     previous menu... ")
        return SearchMenu

    @classmethod
    def search_by_year(cls):
        """ Allows user to filter the database by release year """

        while True:
            release_year = raw_input("Release year: ")

            if release_year.isdigit():
                break

        print

        AlbumPrinter.print_albums(AlbumManager.get_by_year(
        ApplicationState.album_manager, release_year))
        raw_input("\nPress ENTER to go back to the
                     previous menu... ")
        return SearchMenu
```

- PrintCollection

```
class PrintCollection(MenuBase):
""" This menu delivers various methods for printing the database
"""

    def __init__(self):
        self.header = "PRINT MENU"
        self.actions = [
            {
                "id": 1,
                "text": "Sorted by artist",
                "func": self.sorted_by_artist
            },
            {
                "id": 2,
                "text": "Sorted by album name",
                "func": self.sorted_by_album
            },
            {
                "id": 3,
                "text": "Sorted by release year",
                "func": self.sorted_by_year
            },
            {
                "id": 4,
                "text": "Go back",
                "func": lambda: MainMenu
            }
        ]

    @classmethod
    def sorted_by_artist(cls):
        """ Allows user to print the albums sorted by artist """

        AlbumPrinter.print_albums(AlbumManager.get_albums(
        ApplicationState.album_manager))
        raw_input("\nPress ENTER to go back to the
                     previous menu... ")
        return PrintCollection

    @classmethod
    def sorted_by_album(cls):
        """ Allows user to print the albums sorted by album name
        """

        AlbumPrinter.print_albums(AlbumManager.get_albums(
        ApplicationState.album_manager, 'AlbumName'))
        raw_input("\nPress ENTER to go back to the
                     previous menu... ")
        return PrintCollection

    @classmethod
    def sorted_by_year(cls):
        """ Allows user to print the albums sorted by release year
        """

        AlbumPrinter.print_albums(AlbumManager.get_albums(
        ApplicationState.album_manager, 'ReleaseYear'))
        raw_input("\nPress ENTER to go back to the
                     previous menu... ")
        return PrintCollection
```

d) **UserInterface** - klasa odpowiadająca za obsługę menu (zmiany widoków, wykonywanie funkcji przypisanych do poszczególnych opcji, itd.)

```
class UserInterface:
    """ Main class for maintaining the user interface """

    current_menu = None

    def __init__(self): pass

    @classmethod
    def change_menu(cls, new_menu):
        """ Method used for switching menu views """

        cls.current_menu = new_menu
        cls.clear_screen()
        cls.current_menu.print_menu()
        cls.perform_actions()

    @classmethod
    def perform_actions(cls):
        """ Method used for performing actions linked to the given
menu options """

        while True:
            action_id = raw_input("\nEnter your choice: ")
            action = cls.current_menu.get_action(action_id)

            if action is not None:
                break

            print "Index out of range"

        cls.clear_screen()
        cls.current_menu.print_action_header(action["text"])

        new_menu = None

        while True:
            if new_menu is not None:
                break

            new_menu = action["func"]()

        cls.change_menu(new_menu())

    @classmethod
    def print_logo(cls):
        """ Prints the application logo """

        print r"""
       _    _ _                     ____  ____
      / \  | | |__  _   _ _ __ ___ |  _ \| __ )
     / _ \ | | '_ \| | | | '_ ` _ \| | | |  _ \
    / ___ \| | |_) | |_| | | | | | | |_| | |_) |
   /_/   \_\_|_.__/ \__,_|_| |_| |_|____/|____/
                                     version 1.0"""
        print

    @classmethod
    def clear_screen(cls):
        """ Method used for clearing the console screen """

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

e) **AlbumPrinter** - dostarcza funkcjonalność związaną z wypisywaniem tabeli z zawartością bazy albumów

```
class AlbumPrinter:
    """ This class offers functionality required for printing
        the database contents """

    def __init__(self): pass

    @classmethod
    def print_albums(cls, data):

        data = list(data)
        if not data:
            print "No results."
            return

        header = ["Artist", "Album name", "Release year"]
        widths = [len(header[0]), len(header[1]), len(header[2])]
        max_values = []

        max_values.append(max([len(row['Artist'])
                                for row in data]))
        max_values.append(max([len(row['AlbumName'])
                                for row in data]))
        max_values.append(max([len(str(row['ReleaseYear']))
                                for row in data]))

        for i in range(3):
            if max_values[i] > widths[i]:
                widths[i] = max_values[i]

        separator = "+"
        row_format = "|"

        for width in widths:
            row_format += " %-" + "%ss |" % (width,)
            separator += "-" * width + "--+"

        print separator
        print (row_format % (header[0], header[1], header[2]))
        print separator

        for row in data:
            print (row_format % (row["Artist"],
                   row["AlbumName"], row["ReleaseYear"]))

        print separator
```

f) **AlbumManager** - oferuje metody związane z zarządzaniem zawartością bazy danych

```
class AlbumManager:
""" This class offers all methods related to the management of the
database contents """

    def __init__(self, database='music.db'):
        self.database = DatabaseLayer(database)
        self.database.query("CREATE TABLE IF NOT EXISTS
        Collection(ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        Artist TEXT NOT NULL, AlbumName TEXT NOT NULL, ReleaseYear
        INTEGER NOT NULL)")

    def delete_database(self):
        self.database.query("DROP DATABASE Collection")

    def add_album(self, artist, album_name, release_year):
        self.database.query("INSERT INTO Collection (Artist,
        AlbumName, ReleaseYear) VALUES (?, ?, ?)", (artist,
        album_name, release_year))

    def delete_album(self, artist, album_name):
        self.database.query("DELETE FROM Collection WHERE Artist=?
        AND AlbumName=?", (artist, album_name))
    def delete_all_by_artist(self, artist):
        self.database.query("DELETE FROM Collection WHERE
        Artist=?", (artist, ))

    def get_albums(self, key='Artist'):
        return self.database.query("SELECT Artist, AlbumName,
        ReleaseYear FROM Collection ORDER BY " + key)

    def get_by_artist(self, artist, key='ReleaseYear'):
        return self.database.query("SELECT Artist, AlbumName,
        ReleaseYear FROM Collection WHERE Artist=? ORDER BY " +
        key, (artist, ))

    def get_by_name(self, album_name, key='Artist'):
        return self.database.query("SELECT Artist, AlbumName,
        ReleaseYear FROM Collection WHERE AlbumName=? ORDER BY " +
        key, (album_name,))

    def get_by_year(self, release_year, key='Artist'):
        return self.database.query("SELECT Artist, AlbumName,
        ReleaseYear FROM Collection WHERE ReleaseYear=? ORDER BY "
        + key, (release_year,))
```

4. Podsumowanie

Zrealizowany projekt spełnia wszelkie założenia postawione w wymaganiach, oferując przy tym dodatkową funkcjonalność, związaną z przeszukiwaniem oraz prezentacją danych względem określonych kryteriów.