TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

PHÁT TRIỂN PHẦN MỀM MÃ NGUỒN MỞ

Xây dựng phần mềm

Play Audio trên PYTHON

GVHD: Từ Lãng Phiêu
SV: Phạm Hoàng Vũ - 3119560083

TP. HỒ CHÍ MINH, THÁNG 2/2024

Mục lục

1 Phần giới thiệu

Kết quả của dự án là 1 phần mềm play audio, được thiết kế để cung cấp trải nghiệm nghe nhạc và âm thanh dễ dàng và tiện lợi. Phần mềm cho phép người dùng phát các tệp âm thanh mp3, quản lý danh sách phát, v.v. với một giao diện vô cùng thân thiện. Mục tiêu của dự án là mang đến một công cụ hữu ích cho việc thưởng thức âm nhạc hàng ngày.
Dự án được phát triển bằng ngôn ngữ Python cùng với các thư viện Tkinter, Pygame và trên môi trường Linux.

2 Cơ sở lý thuyết

2.1 Giới thiệu thư viện Tkinter

Tkinter là thư viện tiêu chuẩn của Python dùng để tạo giao diện đồ họa (GUI). Thư viện này cung cấp một tập hợp các widget (các thành phần giao diện như nút bấm, nhãn, hộp văn bản, v.v.) để xây dựng các ứng dụng GUI một cách dễ dàng và nhanh chóng.

Một số đặc điểm chính của thư viện Tkinter:

- **Đơn giản và dễ sử dụng**: Tkinter được tích hợp sẵn trong Python, không cần cài đặt thêm các gói bổ sung. Cú pháp của Tkinter rất trực quan và dễ học, phù hợp cho người mới bắt đầu.
- **Đa nền tảng**: Các ứng dụng được xây dựng bằng Tkinter có thể chạy trên nhiều hệ điều hành khác nhau như Windows, macOS và Linux mà không cần thay đổi mã nguồn.
- **Hỗ trợ nhiều widget**: Tkinter cung cấp nhiều loại widget khác nhau như Button, Label, Entry, Text, Frame, Canvas, v.v., giúp lập trình viên dễ dàng thiết kế các giao diện phức tạp.
- **Khả năng mở rộng**: Tkinter cho phép lập trình viên mở rộng chức năng của các widget hiện có hoặc tạo các widget mới tùy theo nhu cầu.

Tkinter được sử dụng để thiết kế giao diện cho toàn bộ dự án này.

2.2 Giới thiệu thư viện Pygame

Pygame là một thư viện của Python, được sử dụng chủ yếu để phát triển trò chơi và các ứng dụng đa phương tiện. Tuy nhiên bên cạnh đó, ngoài khả năng xử lý đồ họa và sự kiện, Pygame còn cung cấp các công cụ mạnh mẽ để xử lý âm thanh, làm cho nó trở thành một lựa chọn tuyệt vời để xây dựng các ứng dụng phát âm thanh.

Một số tính năng chính của Pygame trong việc xây dựng ứng dụng âm thanh bao gồm:

- **Phát và kiểm soát âm thanh**: Pygame hỗ trợ phát các tệp âm thanh ở nhiều định dạng khác nhau như WAV, MP3, và OGG. Người dùng có thể dễ dàng kiểm soát việc phát, tạm dừng, và dừng các bản nhạc hoặc hiệu ứng âm thanh.
- **Quản lý âm lượng**: Thư viện cho phép điều chỉnh âm lượng của các tệp âm thanh riêng lẻ hoặc toàn bộ âm thanh của ứng dụng, mang lại sự linh hoạt trong việc kiểm soát trải nghiệm nghe của người dùng.
- **Hỗ trợ đa kênh**: Pygame cung cấp khả năng phát nhiều tệp âm thanh đồng thời, hữu ích cho việc tạo ra các hiệu ứng âm thanh phong phú và đa dạng trong ứng dụng.
- **Dễ sử dụng và tích hợp**: Pygame có cú pháp đơn giản và dễ hiểu, giúp lập trình viên nhanh chóng nắm bắt và tích hợp vào ứng dụng của mình.

Ví dụ, dưới đây là mã Python đơn giản để phát một tệp âm thanh bằng Pygame:

Pygame không được tích hợp sẵn trong Python như Tkinter nên để sử dụng ta cần phải cài đặt

```
# Cài đặt Pygame bằng câu lệnh pip
pip install pygame
```

Code minh họa phát một tệp âm thanh bằng Pygame:

```
import pygame

# Khởi tạo Pygame
pygame.init()

# Tải và phát tệp âm thanh
pygame.mixer.music.load('path/to/your/audiofile.mp3')
pygame.mixer.music.play()

# Vòng lặp chính để giữ chương trình chạy
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

# Dừng phát âm thanh và thoát
pygame.mixer.music.stop()
pygame.quit()
```

Với Pygame, việc xây dựng một ứng dụng phát âm thanh trở nên đơn giản và hiệu quả, mang lại trải nghiệm nghe tuyệt vời cho người dùng.

3 Thiết kế ứng dụng

3.1 Các tính năng của ứng dụng

3.1.1 Quản lí playlist

Ứng dụng phát âm thanh không chỉ đơn thuần là một trình phát nhạc, mà còn tích hợp tính năng quản lý playlist mạnh mẽ, giúp người dùng dễ dàng quản lý và tuỳ chỉnh trải nghiệm nghe nhạc của mình. Các tính năng quản lý playlist bao gồm:

- **Tìm kiếm Playlist**: Người dùng có thể dễ dàng tìm kiếm các playlist đã lưu của mình bằng cách nhập từ khóa. Tính năng tìm kiếm nhanh chóng và hiệu quả, giúp người dùng truy cập ngay lập tức vào playlist mong muốn.
- **Thêm Playlist**: Ứng dụng cho phép người dùng tạo và lưu nhiều playlist khác nhau. Người dùng có thể đặt tên cho playlist mới, thêm các tệp âm thanh yêu thích vào đó và sắp xếp theo thứ tự mong muốn.
- **Xóa Playlist**: Khi không cần thiết, người dùng có thể dễ dàng xóa các playlist không còn sử dụng. Tính năng này giúp người dùng quản lý không gian lưu trữ và giữ cho danh sách playlist luôn gọn gàng.

Với các tính năng này, ứng dụng không chỉ mang đến trải nghiệm nghe nhạc tuyệt vời mà còn cung cấp công cụ quản lý âm nhạc tiện lợi và hiệu quả. Người dùng có thể tạo các playlist phù hợp với tâm trạng hoặc hoạt động của mình, từ đó tối ưu hóa trải nghiệm nghe nhạc hàng ngày.

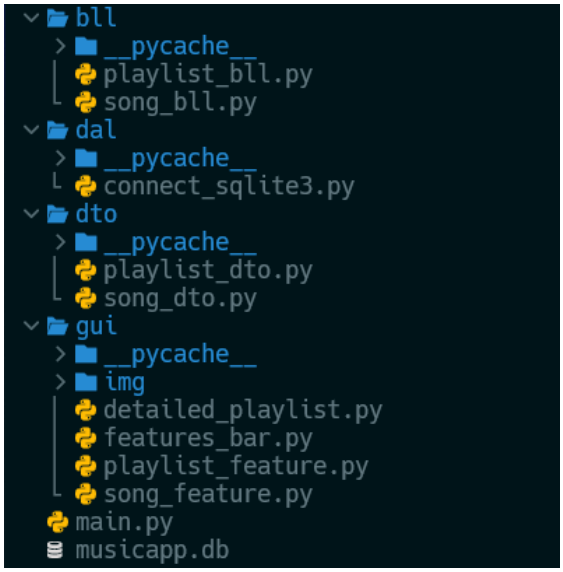
3.1.2 Quản lí bài hát

Ứng dụng phát âm thanh không chỉ cung cấp khả năng phát nhạc mà còn tích hợp các tính năng quản lý bài hát mạnh mẽ, giúp người dùng dễ dàng kiểm soát và tổ chức thư viện âm nhạc của mình. Các tính năng quản lý bài hát bao gồm:

- **Tìm kiếm Bài hát**: Người dùng có thể nhanh chóng tìm kiếm các bài hát trong thư viện của mình bằng cách nhập tên bài hát. Tính năng tìm kiếm thông minh giúp người dùng dễ dàng truy cập ngay lập tức vào bài hát mong muốn.
- **Thêm Bài hát vào Playlist**: Ứng dụng cho phép người dùng thêm các bài hát yêu thích vào các playlist khác nhau. Điều này giúp người dùng tùy chỉnh các playlist theo ý thích và nhu cầu nghe nhạc của mình.
- **Xóa Bài hát khỏi Playlist**: Khi không còn nhu cầu, người dùng có thể dễ dàng xóa các bài hát khỏi playlist. Tính năng này giúp duy trì sự gọn gàng và tổ chức cho các playlist cá nhân.

Với các tính năng này, ứng dụng không chỉ mang đến trải nghiệm nghe nhạc tuyệt vời mà còn cung cấp các công cụ quản lý bài hát tiện lợi và hiệu quả. Người dùng có thể dễ dàng tìm kiếm, tổ chức và tùy chỉnh thư viện âm nhạc của mình, từ đó nâng cao trải nghiệm nghe nhạc hàng ngày.

3.2 Cấu trúc mã nguồn



Hình 1: Cấu trúc các file

Ứng dụng được xây dựng theo kiến trúc mô hình ba lớp.

- **GUI**: lớp tương tác trực tiếp với người dùng. Lớp này chịu trách nhiệm hiển thị và cung cấp các chức năng cần thiết để người dùng có thể tương tác với ứng dụng. Trong ứng dụng play audio này, GUI bao gồm các thành phần như cửa sổ chính, các nút điều khiển phát nhạc, và các biểu mẫu quản lý playlist và bài hát.
- **img**: chứa các file ảnh hỗ trợ tạo giao diện(không phải là thành phần chính của mô hình 3 lớp)
- **BLL**: lớp xử lý các logic nghiệp vụ của ứng dụng. Đây là nơi thực hiện các quy tắc, điều kiện và quy trình mà ứng dụng cần thực hiện. Trong ứng dụng này, BLL chịu trách nhiệm quản lý các thao tác liên quan đến playlist, như tìm kiếm, thêm, xóa playlist, cũng như quản lý các bài hát trong mỗi playlist.
- **DAL**: lớp chịu trách nhiệm truy cập và thao tác dữ liệu trong cơ sở dữ liệu. DAL bao gồm các phương thức để thực hiện các hoạt động như thêm, sửa, xóa và truy vấn dữ liệu từ cơ sở dữ liệu. Trong ứng dụng này, DAL quản lý việc lưu trữ thông tin về playlist và bài hát, đảm bảo rằng dữ liệu được lưu trữ một cách an toàn và hiệu quả.
- **DTO**: chứa các file vận chuyển dữ liệu giữa các lớp(không phải là thành phần chính của mô hình 3 lớp)

4 Kết quả xây dựng

4.1 Mã nguồn

4.1.1 main.py

Đây là file được dùng để khởi chạy toàn bộ dự án.

```
1
2from tkinter import *
3from ttkbootstrap.constants import *
4from gui.features_bar import FeaturesBar
5import ttkbootstrap as tb
6
7
8root = tb.Window(themename='darkly')
9
10root.title('Music App')
11root.geometry('500x500')
12
13# features bar
14featuresBar = FeaturesBar(root)
15featuresBar.grid(row=0, column=0, sticky="nsew", padx=5)
16
17root.mainloop()
```

- `from tkinter import *`: Nhập toàn bộ các module và hàm từ thư viện Tkinter, một thư viện chuẩn của Python dùng để tạo giao diện người dùng.
- `from ttkbootstrap.constants import *`: Nhập toàn bộ các hằng số từ thư viện ttkbootstrap, giúp tạo các giao diện người dùng với các chủ đề hiện đại.
- `from gui.features_bar import FeaturesBar`: Nhập lớp `FeaturesBar` từ module `gui.features_bar`, đây là một thành phần của giao diện người dùng trong ứng dụng.
- `import ttkbootstrap as tb`: Nhập thư viện ttkbootstrap và đặt tên viết tắt là `tb` để dễ dàng sử dụng.
- `root = tb.Window(themename='darkly')`: Dòng này tạo một cửa sổ chính cho ứng dụng với chủ đề "darkly" từ thư viện ttkbootstrap.
- `root.title('Music App')`: Đặt tiêu đề cho cửa sổ chính là "Music App".
- `root.geometry('500x500')`: Đặt kích thước cho cửa sổ chính là 500x500 pixel.
- `featuresBar = FeaturesBar(root)`: Tạo một đối tượng `FeaturesBar` và gắn nó vào cửa sổ chính `root`.
- `featuresBar.grid(row=0, column=0, sticky="nsew", padx=5)`: Đặt `featuresBar` vào lưới của cửa sổ chính tại hàng 0 và cột 0, với các tùy chọn "sticky" để mở rộng và "padx" để thêm khoảng cách ngang.

- `root.mainloop()`: Dòng này bắt đầu vòng lặp chính của giao diện người dùng, giữ cho cửa sổ mở và phản hồi các sự kiện từ người dùng cho đến khi cửa sổ bị đóng.

Tóm lại, đoạn mã này tạo ra một cửa sổ ứng dụng phát nhạc với một thanh tính năng (`FeaturesBar`) được đặt trong lưới (`grid`) của cửa sổ chính. Cửa sổ sử dụng chủ đề "darkly" từ ttkbootstrap để có giao diện hiện đại và bắt mắt.

4.1.2 features_bar.py

File này tạo giao diện thanh chức năng của ứng dụng. Cho phép người dùng lựa chọn sử dụng các chức năng của ứng dụng.

```
1
2 from tkinter import *
3 from PIL import Image, ImageTk
4 from .playlist_feature import PlaylistFeature_Frame
5 from .song_feature import SongFeature_Frame
6 from ttkbootstrap.constants import *
7 import ttkbootstrap as tb
8
9 class FeaturesBar(tb.Frame):
10     def __init__(self, master):
11         self.master = master
12         super().__init__(self.master)
13
14         # Icons (assuming these are pre-loaded elsewhere)
15         self.icon_images = {
16             "home": ImageTk.PhotoImage(Image.open('./gui/img/home.png')),
17             "playlists": ImageTk.PhotoImage(Image.open('./gui/img/playlist.png')),
18         }
19         self.icon_on_images = {
20             "home": ImageTk.PhotoImage(Image.open('./gui/img/home_on.png')),
21             "playlists": ImageTk.PhotoImage(Image.open('./gui/img/playlist_on.png')),
22         }
23
24         # Create icon labels
25         self.icon_labels = {} # Dictionary to store label references
26         for icon_name, image in self.icon_images.items():
27             label = tb.Label(self, image=image)
28             label.bind("<Enter>", self.on_enter)
29             label.bind("<Leave>", self.on_leave)
30             label.bind("<Button-1>", lambda event, icon=icon_name:
                   self.on_click(event, icon))
31             label.pack(padx=5, pady=10)
32             self.icon_labels[icon_name] = label
33
34         self.feature_frames = {}
35
36         self.playlist_feature_frame = PlaylistFeature_Frame(self.master)
37         self.playlist_feature_frame.grid_forget()
38         self.song_feature_frame = SongFeature_Frame(self.master)
39         self.song_feature_frame.grid_forget()
```

```
40
41          self.feature_frames["home"] = self.playlist_feature_frame
42          self.feature_frames["playlists"] = self.song_feature_frame
43
44
45      def on_enter(self, event):
46          event.widget.config(cursor="hand2") # Change to pointer cursor
47
48      def on_leave(self, event):
49          event.widget.config(cursor="") # Reset to default cursor
50
51      def on_click(self, event, icon_name):
52          # Update all labels using stored references
53          for name, label in self.icon_labels.items():
54              if name == icon_name:
55                  label.config(image=self.icon_on_images[name])
56                  self.feature_frames[name].grid(row=0, column=1, sticky="nsew")
57              else:
58                  label.config(image=self.icon_images[name])
59                  self.feature_frames[name].grid_forget()
```

- **__init__(self, master)**: Hàm khởi tạo của lớp FeaturesBar nhận tham số master, đại diện cho cửa sổ chính của ứng dụng.
- **on_enter, on_leave, on_click**: được gọi khi người dùng di chuột qua, rời khỏi, hoặc click vào biểu tượng tương ứng. Chúng thay đổi hình ảnh của biểu tượng và hiển thị hoặc ẩn các tính năng tương ứng trên giao diện.

4.1.3 playlist_feature.py

File này tạo giao diện chức năng quản lý playlist của ứng dụng.

```
1
2 from tkinter import *
3 from PIL import Image, ImageTk
4 from ttkbootstrap.constants import *
5 from tkinter import filedialog
6 from tkinter import simpledialog
7 from tkinter import messagebox
8 from .detailed_playlist import DetailedPlaylist_Frame
9 from dto.playlist_dto import PlaylistDTO
10 from bll.playlist_bll import PlaylistBLL
11 import ttkbootstrap as tb
12 import pygame
13 import os
14
15 class PlaylistFeature_Frame(tb.Frame):
16     def __init__(self, master):
17         self.master = master
18         super().__init__(self.master)
19
```

```
20          pygame.mixer.init()
21
22          self.playlistbll = PlaylistBLL()
23
24          self.__initComponents()
25
26      def __initComponents(self):
27          self.top_container = tb.Frame(self)
28          self.top_container.pack()
29
30          self.icon_images = {
31              "search-btn": ImageTk.PhotoImage(Image.open('./gui/img/loupe.png')),
32              "refresh-btn":
                    ImageTk.PhotoImage(Image.open('./gui/img/refresh-button.png')),
33              "shuffle": ImageTk.PhotoImage(Image.open('./gui/img/shuffle-no.png')),
34              "shuffle-yes": ImageTk.PhotoImage(Image.open('./gui/img/shuffle.png')),
35              "previous": ImageTk.PhotoImage(Image.open('./gui/img/previous.png')),
36              "play":
                    ImageTk.PhotoImage(Image.open('./gui/img/play-button-arrowhead.png')),
37              "next": ImageTk.PhotoImage(Image.open('./gui/img/next.png')),
38              "repeat": ImageTk.PhotoImage(Image.open('./gui/img/repeat.png')),
39              "repeat-yes":
                    ImageTk.PhotoImage(Image.open('./gui/img/repeat-once.png')),
40              "speaker": ImageTk.PhotoImage(Image.open('./gui/img/sound.png')),
41              "pause": ImageTk.PhotoImage(Image.open('./gui/img/pause.png')),
42          }
43
44          self.feature_name = tb.Label(self.top_container, text="Playlists",
                font=("Helvetica", 30, "bold italic"))
45          self.feature_name.grid(row=0, column=0, padx=(45, 20), pady=(10, 0))
46
47          self.search_bar_container = tb.Frame(self.top_container)
48          self.search_bar_container.grid(row=0, column=1, padx=(50, 0), pady=(10, 0))
49
50          self.search_bar = tb.Entry(self.search_bar_container, width=40)
51          self.search_bar.grid(row=0, column=0, padx=10)
52
53          self.search_btn = tb.Button(self.search_bar_container,
                image=self.icon_images["search-btn"], command=self.search)
54          self.search_btn.grid(row=0, column=1, padx=10)
55
56          self.refresh_btn = tb.Button(self.search_bar_container,
                image=self.icon_images["refresh-btn"], command=self.refresh)
57          self.refresh_btn.grid(row=0, column=2, padx=10)
58
59          self.add_playlist_btn = tb.Button(self.top_container, text="Add playlist",
                command=self.add_playlist)
60          self.add_playlist_btn.grid(row=1, column=0, pady=10)
61
62          self.delete_playlist = tb.Button(self.top_container, text="Delete playlist",
                command=self.delete_playlist)
63          self.delete_playlist.grid(row=1, column=1, pady=10)
```

```
64
65          self.playlist_box = Listbox(self.top_container, fg="green", height=20,
                width=100)
66          self.playlist_box.grid(row=2, column=1, padx=(0, 10), pady=10)
67          self.playlist_box.bind("<Double-Button-1>", self.on_double_click)
68
69          playlists = self.playlistbll.fetchPlaylists()
70          for playlist in playlists:
71              self.playlist_box.insert(END, playlist[0])
72
73
74
75      def add_playlist(self):
76          playlist_name = simpledialog.askstring("Input", "Enter playlist name:")
77          if self.playlistbll.isPlaylistExisted(playlist_name):
78              messagebox.showerror("Error", "Playlist is already existed !")
79          else:
80              playlist = PlaylistDTO(playlist_name)
81              self.playlistbll.savePlaylist(playlist)
82
83              self.playlist_box.insert(END, playlist.getName())
84
85      def delete_playlist(self):
86          selected_index = self.playlist_box.curselection()
87          if selected_index:
88              self.playlistbll.deletePlaylist(self.playlist_box.get(ACTIVE))
89              self.playlist_box.delete(selected_index)
90
91
92
93      def refresh(self):
94          self.playlist_box.delete(0, END)
95          playlists = self.playlistbll.fetchPlaylists()
96          for playlist in playlists:
97              self.playlist_box.insert(END, playlist[0])
98
99      def search(self):
100         self.playlist_box.delete(0, END)
101         playlists = self.playlistbll.fetchPlaylist(self.search_bar.get())
102         for playlist in playlists:
103             self.playlist_box.insert(END, playlist[0])
104
105
106
107     def on_double_click(self, event):
108         widget = event.widget
109         selection = widget.curselection()
110
111         if selection:
112             index = selection[0]
113             value = widget.get(index)
114             self.detailed_playlist_frame = DetailedPlaylist_Frame(self.master, value)
```

```
115            self.detailed_playlist_frame.grid(row=0, column=1, sticky="nsew")
```

- **__init__(self, master)**: Hàm khởi tạo của lớp PlaylistFeature_Frame nhận tham số master, đại diện cho cửa sổ chính của ứng dụng.
- **__initComponents(self)**: Khởi tạo các thành phần cho chức năng quản lý playlist.
- **add_playlist(self)**: Tạo và thêm playlist vào database khi người dùng nhấn vào nút "Add playlist".
- **delete_playlist(self)**: Xóa playlist khỏi database khi người dùng nhấn vào nút "Delete playlist".
- **search(self)**: Tìm kiếm playlist khi người dùng nhập tên playlist và nhấn nút search
- **refresh(self)**: Hiển thị lại tất cả các playlist hiện có trong database.
- **on_double_click(self)**: Khi người dùng click đúp chuột vào một playlist bất kì thì sẽ hiển thị tất cả các bài hát hiện có trong playlist đó.

4.1.4 song_feature.py

File này tạo giao diện chức năng quản lý playlist của ứng dụng.

```
1
2 class SongFeature_Frame(tb.Frame):
3     def __init__(self, master):
4         super().__init__(master)
5
6         pygame.mixer.init()
7
8         self.songbll = SongBLL()
9
10        self.isPaused = False
11        self.is_shuffled = False
12        self.is_repeated = False
13
14        self.__initComponents()
15
16    def __initComponents(self):
17        self.top_container = tb.Frame(self)
18        self.top_container.pack()
19
20        self.icon_images = {
21            "search-btn": ImageTk.PhotoImage(Image.open('./gui/img/loupe.png')),
22            "refresh-btn":
                  ImageTk.PhotoImage(Image.open('./gui/img/refresh-button.png')),
23            "shuffle": ImageTk.PhotoImage(Image.open('./gui/img/shuffle-no.png')),
24            "shuffle-yes": ImageTk.PhotoImage(Image.open('./gui/img/shuffle.png')),
25            "previous": ImageTk.PhotoImage(Image.open('./gui/img/previous.png')),
26            "play":
                  ImageTk.PhotoImage(Image.open('./gui/img/play-button-arrowhead.png')),
```

```
            "next": ImageTk.PhotoImage(Image.open('./gui/img/next.png')),
            "repeat": ImageTk.PhotoImage(Image.open('./gui/img/repeat.png')),
            "repeat-yes":
                ImageTk.PhotoImage(Image.open('./gui/img/repeat-once.png')),
            "speaker": ImageTk.PhotoImage(Image.open('./gui/img/sound.png')),
            "pause": ImageTk.PhotoImage(Image.open('./gui/img/pause.png')),
        }

        self.feature_name = tb.Label(self.top_container, text="Songs",
            font=("Helvetica", 30, "bold italic"))
        self.feature_name.grid(row=0, column=0, padx=(45, 20), pady=(10, 0))

        self.search_bar_container = tb.Frame(self.top_container)
        self.search_bar_container.grid(row=0, column=1, padx=(50, 0), pady=(10, 0))

        self.search_bar = tb.Entry(self.search_bar_container, width=40)
        self.search_bar.grid(row=0, column=0, padx=10)

        self.search_btn = tb.Button(self.search_bar_container,
            image=self.icon_images["search-btn"], command=self.search)
        self.search_btn.grid(row=0, column=1, padx=10)

        self.refresh_btn = tb.Button(self.search_bar_container,
            image=self.icon_images["refresh-btn"], command=self.refresh)
        self.refresh_btn.grid(row=0, column=2, padx=10)

        self.add_song_btn = tb.Button(self.top_container, text="Add song",
            command=self.add_song)
        self.add_song_btn.grid(row=1, column=0, pady=10)

        self.delete_song = tb.Button(self.top_container, text="Delete song",
            command=self.delete_song)
        self.delete_song.grid(row=1, column=1, pady=10)

        self.song_box = Listbox(self.top_container, fg="green", height=20, width=100)
        self.song_box.grid(row=2, column=1, padx=(0, 10), pady=10)

        songs = self.songbll.fetchSongs()
        for song in songs:
            self.song_box.insert(END, song[0])

        # Buttons Container
        self.initBtnContainer()


    def add_song(self):
        song_path = filedialog.askopenfilename(initialdir="~/Music", title="Choose a
            song", filetypes=(("mp3 Files", "*.mp3"), ))
        song_name = song_path.split("/")
        song_name = song_name[-1]
        song_name = song_name.split(".")
        song_name = song_name[0]
```

```
72
73          if self.songbll.isSongExisted(song_name):
74              messagebox.showerror("Error", "Song is already existed !")
75          else:
76              song = SongDTO(song_name, song_path)
77              self.songbll.saveSong(song)
78
79              self.song_box.insert(END, song_name)
80
81      def delete_song(self):
82          selected_index = self.song_box.curselection()
83          if selected_index:
84              self.songbll.deleteSong(self.song_box.get(ACTIVE))
85              self.song_box.delete(selected_index)
86
87      def refresh(self):
88          self.song_box.delete(0, END)
89          songs = self.songbll.fetchSongs()
90          for song in songs:
91              self.song_box.insert(END, song[0])
92
93      def search(self):
94          self.song_box.delete(0, END)
95          songs = self.songbll.fetchSong(self.search_bar.get())
96          for song in songs:
97              self.song_box.insert(END, song[0])
98
99      def initBtnContainer(self):
100         self.btn_container_frame = tb.Frame(self.top_container)
101         self.btn_container_frame.grid(row=3, column=1, sticky="nsew", pady=(10, 0))
102
103         self.current_song = tb.Label(self.btn_container_frame)
104         self.current_song.pack(pady=10)
105
106         self.slider = ttk.Scale(self.btn_container_frame, from_=0, to=100,
                orient=HORIZONTAL, value=0, command=self.slide, length=500)
107         self.slider.pack(pady=10)
108
109
110         self.status_bar = tb.Label(self.btn_container_frame, font=("Helvetica", 18))
111         self.status_bar.pack(pady=5)
112
113         self.btn_container = tb.Frame(self.btn_container_frame)
114         self.btn_container.pack()
115
116         self.shuffle_btn = tb.Label(self.btn_container,
                image=self.icon_images['shuffle'])
117         self.shuffle_btn.grid(row=0, column=0, padx=5)
118         self.shuffle_btn.bind("<Enter>", self.on_enter)
119         self.shuffle_btn.bind("<Leave>", self.on_leave)
120         self.shuffle_btn.bind("<Button-1>", self.on_click_shuffle)
121
```

```
122         self.previous_btn = tb.Label(self.btn_container,
                image=self.icon_images['previous'])
123         self.previous_btn.grid(row=0, column=1, padx=5)
124         self.previous_btn.bind("<Enter>", self.on_enter)
125         self.previous_btn.bind("<Leave>", self.on_leave)
126         self.previous_btn.bind("<Button-1>", self.on_click_previous)
127 
128         self.play_btn = tb.Label(self.btn_container, image=self.icon_images['play'])
129         self.play_btn.grid(row=0, column=2, padx=5)
130         self.play_btn.bind("<Enter>", self.on_enter)
131         self.play_btn.bind("<Leave>", self.on_leave)
132         self.play_btn.bind("<Button-1>", self.on_click_play)
133 
134         self.pause_btn = tb.Label(self.btn_container,
                image=self.icon_images['pause'])
135         self.pause_btn.grid(row=0, column=3, padx=5)
136         self.pause_btn.bind("<Enter>", self.on_enter)
137         self.pause_btn.bind("<Leave>", self.on_leave)
138         self.pause_btn.bind("<Button-1>", self.on_click_pause)
139 
140         self.next_btn = tb.Label(self.btn_container, image=self.icon_images['next'])
141         self.next_btn.grid(row=0, column=4, padx=5)
142         self.next_btn.bind("<Enter>", self.on_enter)
143         self.next_btn.bind("<Leave>", self.on_leave)
144         self.next_btn.bind("<Button-1>", self.on_click_next)
145 
146         self.repeat_btn = tb.Label(self.btn_container,
                image=self.icon_images['repeat'])
147         self.repeat_btn.grid(row=0, column=5, padx=5)
148         self.repeat_btn.bind("<Enter>", self.on_enter)
149         self.repeat_btn.bind("<Leave>", self.on_leave)
150         self.repeat_btn.bind("<Button-1>", self.on_click_repeat)
151 
152     def on_enter(self, event):
153         event.widget.config(cursor="hand2") # Change to pointer cursor
154 
155     def on_leave(self, event):
156         event.widget.config(cursor="") # Reset to default cursor
157 
158     def play_time(self):
159         current_time = pygame.mixer.music.get_pos() / 1000
160 
161         converted_current_time = time.strftime("%M:%S", time.gmtime(current_time))
162         curent_song = self.song_box.curselection()
163         song_name = self.song_box.get(curent_song)
164         song_path = self.songbll.getSongPath(song_name)
165         self.song_length = MP3(song_path[0][0]).info.length
166         self.converted_song_length = time.strftime("%M:%S",
                time.gmtime(self.song_length))
167 
168 
169         current_time += 1
```

```
170
171         if int(self.slider.get()) == int(self.song_length):
172             self.status_bar.configure(text=f"{self.converted_song_length} /
                    {self.converted_song_length}")
173         elif self.isPaused:
174             pass
175         elif int(self.slider.get()) == int(current_time):
176             # slider has not been moved
177             slider_position = int(self.song_length)
178             self.slider.config(to=slider_position, value=int(current_time))
179         else:
180             # slider has been moved
181             slider_position = int(self.song_length)
182             self.slider.config(to=slider_position, value=int(self.slider.get()))
183
184             # convert time to format
185             converted_current_time = time.strftime("%M:%S",
                    time.gmtime(int(self.slider.get())))
186
187             # output time to status bar
188             self.status_bar.configure(text=f"{converted_current_time} /
                    {self.converted_song_length}")
189
190             # Move this thing along by one second
191             next_time = int(self.slider.get()) + 1
192             self.slider.config(to=slider_position, value=next_time)
193
194
195
196         # self.slider.config(value=int(current_time))
197
198
199         # update time
200         self.status_bar.after(1000, self.play_time)
201
202     def slide(self, X):
203         # self.slider_lb.config(text=f"{int(self.slider.get())} of
                {self.converted_song_length}")
204         song = self.song_box.get(ACTIVE)
205         song_path = self.songbll.getSongPath(song)
206
207         pygame.mixer.music.load(song_path[0][0])
208         pygame.mixer.music.play(loops=0, start=int(self.slider.get()))
209
210
211
212     def on_click_play(self, event):
213         song = self.song_box.get(ACTIVE)
214
215         self.current_song.config(text=song)
216         self.status_bar.config(text="")
217         self.slider.config(value=0)
```

```
218
219         song_path = self.songbll.getSongPath(song)
220
221         pygame.mixer.music.load(song_path[0][0])
222         pygame.mixer.music.play(loops=0)
223
224         self.play_time()
225
226         # slider_position = int(self.song_length)
227         # self.slider.config(to=slider_position, value=0)
228
229     def on_click_pause(self, event):
230         if self.isPaused:
231             pygame.mixer.music.unpause()
232             self.isPaused = False
233         else:
234             pygame.mixer.music.pause()
235             self.isPaused = True
236
237     def on_click_previous(self, event):
238         selected_song = self.song_box.curselection()[0]
239         self.song_box.selection_clear(0, END)
240
241         if selected_song > 0:
242             selected_song = selected_song - 1
243         else:
244             selected_song = self.song_box.size() - 1
245
246         self.song_box.selection_set(selected_song)
247
248         song = self.song_box.get(selected_song)
249         self.current_song.config(text=song)
250         self.status_bar.config(text="")
251         self.slider.config(value=0)
252
253         song_path = self.songbll.getSongPath(song)
254
255         pygame.mixer.music.load(song_path[0][0])
256         pygame.mixer.music.play(loops=0)
257
258         self.play_time()
259
260
261     def on_click_next(self, event):
262         selected_song = self.song_box.curselection()[0]
263         self.song_box.selection_clear(0, END)
264
265         if selected_song == self.song_box.size() - 1:
266             selected_song = 0
267         else:
268             selected_song = selected_song + 1
269
```

```
        self.song_box.selection_set(selected_song)

        song = self.song_box.get(selected_song)
        self.current_song.config(text=song)
        self.status_bar.config(text="")
        self.slider.config(value=0)

        song_path = self.songbll.getSongPath(song)

        pygame.mixer.music.load(song_path[0][0])
        pygame.mixer.music.play(loops=0)

        self.play_time()
```

- **__init__(self, master)**: Hàm khởi tạo của class SongFeature_Frame. "pygame.mixer.init()" khởi tạo pygame để play bài hát.
- **__initComponents(self)**: Phương thức khởi tạo các components cho chức năng quản lý playlist.
- **add_song(self)**: giúp người dùng chọn một file bài hát, kiểm tra xem bài hát đó đã tồn tại trong hệ thống hay chưa, và nếu chưa, sẽ lưu bài hát mới vào hệ thống và cập nhật giao diện người dùng để hiển thị bài hát mới này.
  - **song_path = filedialog.askopenfilename(initialdir=" /Music", title="Choose a song", filetypes=(("mp3 Files", "*.mp3"),))**:
    + Hộp thoại tệp được mở để người dùng chọn tệp bài hát. Đường dẫn của tệp được chọn sẽ được lưu trong biến song_path.
    + initialdir=" /Music": Thư mục mở ban đầu là thư mục Music trong thư mục người dùng.
    + title="Choose a song": Tiêu đề của hộp thoại là "Choose a song".
    + filetypes=(("mp3 Files", "*.mp3"),): Chỉ cho phép chọn các tệp có đuôi .mp3.
- **delete_song(self)**: giúp người dùng xóa một bài hát khỏi danh sách. Nó thực hiện các bước sau:
  + Lấy chỉ mục của bài hát đang được chọn trong danh sách.
  + Kiểm tra xem có bài hát nào được chọn hay không.
  + Nếu có, xóa bài hát khỏi hệ thống lưu trữ và cập nhật giao diện người dùng để loại bỏ bài hát khỏi danh sách hiển thị.
- **refresh(self)**: Phương thức giúp cập nhật danh sách bài hát trong giao diện người dùng bằng cách
  + Xóa toàn bộ danh sách hiện tại.
  + Lấy danh sách mới nhất các bài hát từ hệ thống.
  + Thêm từng bài hát từ danh sách mới vào song_box để hiển thị.
  Phương thức này đảm bảo rằng danh sách bài hát luôn được cập nhật và hiển thị đúng nhất theo dữ liệu hiện tại trong hệ thống.
- **search(self)**: Phương thức giúp giúp người dùng tìm kiếm các bài hát dựa trên từ khóa tìm kiếm và hiển thị kết quả trong giao diện người dùng bằng cách:
  + Xóa toàn bộ danh sách hiện tại.

+ Lấy danh sách các bài hát phù hợp với từ khóa tìm kiếm từ hệ thống.
+ Thêm từng bài hát từ danh sách kết quả vào song_box để hiển thị.
Hàm này giúp người dùng dễ dàng tìm kiếm và xem các bài hát có trong hệ thống dựa trên từ khóa nhập vào.

- **initBtnContainer(self)**: Phương thức này giúp xây dựng giao diện điều khiển cho ứng dụng phát nhạc, cung cấp các chức năng cơ bản để người dùng tương tác với danh sách bài hát và điều khiển việc phát nhạc.

- **on_click_play(self, event)**: Phương thức được thiết kế để thực hiện các bước sau khi người dùng nhấn nút "Play":
+ Lấy tên của bài hát hiện tại từ danh sách bài hát.
+ Cập nhật nhãn hiển thị tên bài hát hiện tại, thanh trạng thái và thanh trượt.
+ Lấy đường dẫn của bài hát từ songbll.
+ Tải và phát bài hát bằng thư viện Pygame.
+ Bắt đầu cập nhật thời gian phát nhạc và giao diện người dùng mỗi giây thông qua hàm play_time.
Hàm này giúp đảm bảo rằng khi người dùng nhấn nút "Play", bài hát sẽ được phát và giao diện người dùng sẽ hiển thị thông tin chính xác và cập nhật liên tục.

- **play_time(self)**:
+ "current_time = pygame.mixer.music.get_pos() / 1000": Lấy vị trí hiện tại của bài hát đang phát (tính bằng mili giây) từ pygame.mixer.music và chuyển đổi sang giây.
+ "converted_current_time = time.strftime("%M:%S", time.gmtime(current_time))": Chuyển đổi thời gian hiện tại sang định dạng phút giây
+ Từ dòng 162 -> 166: Lấy bài hát hiện tại và đường dẫn của nó sau đó lấy độ dài của bài hát (tính bằng giây) từ tệp MP3 và chuyển đổi độ dài của bài hát sang định dạng phút giây.
+ Từ dòng 169 -> 192: Cập nhật thanh trượt và thanh trạng thái
+ current_time += 1: Tăng thời gian hiện tại thêm 1 giây.
+ Nếu thanh trượt đang ở vị trí cuối cùng (int(self.slider.get()) == int(self.song_length)), cập nhật thanh trạng thái để hiển thị thời gian đã hết.
+ Nếu bài hát đang tạm dừng (self.isPaused), không làm gì cả.
+ Nếu thanh trượt đang ở vị trí hiện tại của bài hát (int(self.slider.get()) == int(current_time)), cập nhật giá trị của thanh trượt và vị trí bài hát.
+ Nếu thanh trượt đã được di chuyển (else), cập nhật giá trị của thanh trượt, chuyển đổi thời gian hiện tại sang định dạng phút giây, và cập nhật thanh trạng thái. + "self.status_bar.after(1000, self.play_time)": Đặt hàm play_time để gọi lại sau 1 giây, giúp cập nhật thời gian hiện tại và giao diện người dùng mỗi giây.
Phương thức play_time(self) giúp đảm bảo rằng khi người dùng nhấn nút "Play", bài hát sẽ được phát và giao diện người dùng sẽ hiển thị thông tin chính xác và cập nhật liên tục.

- **slide(self, X)**: được sử dụng để xử lý sự kiện khi thanh trượt (slider) thay đổi giá trị, cho phép người dùng tua đến một vị trí cụ thể trong bài hát
+ "song = self.song_box.get(ACTIVE)": Lấy tên của bài hát đang được chọn từ song_box.
+ "song_path = self.songbll.getSongPath(song)": Lấy đường dẫn của bài hát từ songbll.
+ "pygame.mixer.music.load(song_path[0][0])": Tải bài hát từ đường dẫn.
+ "pygame.mixer.music.play(loops=0, start=int(self.slider.get()))": Phát bài hát từ vị trí được xác định bởi giá trị hiện tại của thanh trượt. "start=int(self.slider.get())" cho biết vị trí bắt đầu phát bài hát tính bằng giây.

Phương thức slide giúp đảm bảo rằng khi người dùng di chuyển thanh trượt, bài hát sẽ được tua đến đúng vị trí mong muốn và tiếp tục phát từ đó.

- **on_click_pause(self, event)**: Dừng bài hát hiện tại.
- **on_click_previous(self, event)**: Phát bài hát trước đó.
- **on_click_next(self, event)**: Phát bài hát tiếp theo.

4.1.5 detailed_playlist.py

Khi người dùng nhấp đúp vào một playlist bất kỳ, một giao diện được tạo ra hiển thị tất cả các bài hát trong playlist đó. File này tạo ra giao diện đó và các component liên quan.

```
1
2 class DetailedPlaylist_Frame(tb.Frame):
3     def __init__(self, master, playlist):
4         super().__init__(master)
5
6         self.playlist = playlist
7         self.isPaused = False
8         self.is_shuffled = False
9         self.is_repeated = False
10
11        pygame.mixer.init()
12
13        self.songbll = SongBLL()
14
15        self.__initComponents()
16
17    def __initComponents(self):
18        self.top_container = tb.Frame(self)
19        self.top_container.pack()
20
21        self.icon_images = {
22            "search-btn": ImageTk.PhotoImage(Image.open('./gui/img/loupe.png')),
23            "refresh-btn":
                  ImageTk.PhotoImage(Image.open('./gui/img/refresh-button.png')),
24            "shuffle": ImageTk.PhotoImage(Image.open('./gui/img/shuffle-no.png')),
25            "shuffle-yes": ImageTk.PhotoImage(Image.open('./gui/img/shuffle.png')),
26            "previous": ImageTk.PhotoImage(Image.open('./gui/img/previous.png')),
27            "play":
                  ImageTk.PhotoImage(Image.open('./gui/img/play-button-arrowhead.png')),
28            "next": ImageTk.PhotoImage(Image.open('./gui/img/next.png')),
29            "repeat": ImageTk.PhotoImage(Image.open('./gui/img/repeat.png')),
30            "repeat-yes":
                  ImageTk.PhotoImage(Image.open('./gui/img/repeat-once.png')),
31            "speaker": ImageTk.PhotoImage(Image.open('./gui/img/sound.png')),
32            "pause": ImageTk.PhotoImage(Image.open('./gui/img/pause.png')),
33        }
34
35        self.feature_name = tb.Label(self.top_container, text=self.playlist,
              font=("Helvetica", 30, "bold italic"))
```

```
36          self.feature_name.grid(row=0, column=0, padx=(45, 20), pady=(10, 0))
37
38          self.search_bar_container = tb.Frame(self.top_container)
39          self.search_bar_container.grid(row=0, column=1, padx=(50, 0), pady=(10, 0))
40
41          self.search_bar = tb.Entry(self.search_bar_container, width=40)
42          self.search_bar.grid(row=0, column=0, padx=10)
43
44          self.search_btn = tb.Button(self.search_bar_container,
                image=self.icon_images["search-btn"], command=self.search)
45          self.search_btn.grid(row=0, column=1, padx=10)
46
47          self.refresh_btn = tb.Button(self.search_bar_container,
                image=self.icon_images["refresh-btn"], command=self.refresh)
48          self.refresh_btn.grid(row=0, column=2, padx=10)
49
50          self.back_btn = tb.Button(self.top_container, text="Back", command=self.back)
51          self.back_btn.grid(row=1, column=0, pady=10)
52
53          self.add_song_btn = tb.Button(self.top_container, text="Add song",
                command=self.add_song)
54          self.add_song_btn.grid(row=1, column=1, pady=10)
55
56          self.delete_song = tb.Button(self.top_container, text="Delete song",
                command=self.delete_song)
57          self.delete_song.grid(row=1, column=2, pady=10)
58
59          self.song_box = Listbox(self.top_container, fg="green", height=20, width=100)
60          self.song_box.grid(row=2, column=1, padx=(0, 10), pady=10)
61
62          songs = self.songbll.fetchSongsInPlaylist(self.playlist)
63          for song in songs:
64              self.song_box.insert(END, song[0])
65
66          # Buttons Container
67          self.initBtnContainer()
68
69
70      def back(self):
71          self.destroy()
72
73      def add_song(self):
74          song_path = filedialog.askopenfilename(initialdir="~/Music", title="Choose a
                song", filetypes=(("mp3 Files", "*.mp3"), ))
75          song_name = song_path.split("/")
76          song_name = song_name[-1]
77          song_name = song_name.split(".")
78          song_name = song_name[0]
79
80
81          if not self.songbll.isSongExisted(song_name):
82              song = SongDTO(song_name, song_path)
```

```
83              self.songbll.saveSong(song)
84
85          self.songbll.addSongToPlaylist(self.playlist, song_name)
86          self.song_box.insert(END, song_name)
87
88      def delete_song(self):
89          selected_index = self.song_box.curselection()
90          if selected_index:
91              self.songbll.deleteSongInPlaylist(self.playlist,
                    self.song_box.get(ACTIVE))
92              self.song_box.delete(selected_index)
93
94      def refresh(self):
95          self.song_box.delete(0, END)
96          songs = self.songbll.fetchSongsInPlaylist(self.playlist)
97          for song in songs:
98              self.song_box.insert(END, song[0])
99
100     def search(self):
101         self.song_box.delete(0, END)
102         songs = self.songbll.fetchSongInPlaylist(self.playlist,
                self.search_bar.get())
103         for song in songs:
104             self.song_box.insert(END, song[1])
105
106     def initBtnContainer(self):
107         self.btn_container_frame = tb.Frame(self.top_container)
108         self.btn_container_frame.grid(row=3, column=1, sticky="nsew", pady=(10, 0))
109
110         self.current_song = tb.Label(self.btn_container_frame)
111         self.current_song.pack(pady=10)
112
113         self.slider = ttk.Scale(self.btn_container_frame, from_=0, to=100,
                orient=HORIZONTAL, value=0, command=self.slide, length=500)
114         self.slider.pack(pady=10)
115
116         self.status_bar = tb.Label(self.btn_container_frame, font=("Helvetica", 18))
117         self.status_bar.pack(pady=5)
118
119         self.btn_container = tb.Frame(self.btn_container_frame)
120         self.btn_container.pack()
121
122         self.shuffle_btn = tb.Label(self.btn_container,
                image=self.icon_images['shuffle'])
123         self.shuffle_btn.grid(row=0, column=0, padx=5)
124         self.shuffle_btn.bind("<Enter>", self.on_enter)
125         self.shuffle_btn.bind("<Leave>", self.on_leave)
126         self.shuffle_btn.bind("<Button-1>", self.on_click_shuffle)
127
128         self.previous_btn = tb.Label(self.btn_container,
                image=self.icon_images['previous'])
129         self.previous_btn.grid(row=0, column=1, padx=5)
```

```
130         self.previous_btn.bind("<Enter>", self.on_enter)
131         self.previous_btn.bind("<Leave>", self.on_leave)
132         self.previous_btn.bind("<Button-1>", self.on_click_previous)
133 
134         self.play_btn = tb.Label(self.btn_container, image=self.icon_images['play'])
135         self.play_btn.grid(row=0, column=2, padx=5)
136         self.play_btn.bind("<Enter>", self.on_enter)
137         self.play_btn.bind("<Leave>", self.on_leave)
138         self.play_btn.bind("<Button-1>", self.on_click_play)
139 
140         self.pause_btn = tb.Label(self.btn_container,
                image=self.icon_images['pause'])
141         self.pause_btn.grid(row=0, column=3, padx=5)
142         self.pause_btn.bind("<Enter>", self.on_enter)
143         self.pause_btn.bind("<Leave>", self.on_leave)
144         self.pause_btn.bind("<Button-1>", self.on_click_pause)
145 
146         self.next_btn = tb.Label(self.btn_container, image=self.icon_images['next'])
147         self.next_btn.grid(row=0, column=4, padx=5)
148         self.next_btn.bind("<Enter>", self.on_enter)
149         self.next_btn.bind("<Leave>", self.on_leave)
150         self.next_btn.bind("<Button-1>", self.on_click_next)
151 
152         self.repeat_btn = tb.Label(self.btn_container,
                image=self.icon_images['repeat'])
153         self.repeat_btn.grid(row=0, column=5, padx=5)
154         self.repeat_btn.bind("<Enter>", self.on_enter)
155         self.repeat_btn.bind("<Leave>", self.on_leave)
156         self.repeat_btn.bind("<Button-1>", self.on_click_repeat)
```

- **__init__(self, master)**: Hàm khởi tạo của class DetailedPlaylist_Frame. "pygame.mixer.init()" khởi tạo pygame để play bài hát.
- **__initComponents(self)**: Phương thức khởi tạo các component liên quan.
- **back(self)**: Hủy tất cả các component chứa trong DetailedPlaylist_Frame để quay lại chức năng quản lý các playlist.
- **add_song(self)**: giúp người dùng chọn một file bài hát, và thêm bài hát đó vào playlist.
  - **song_path = filedialog.askopenfilename(initialdir=" /Music", title="Choose a song", filetypes=(("mp3 Files", "*.mp3"),))**:
    + Hộp thoại tệp được mở để người dùng chọn tệp bài hát. Đường dẫn của tệp được chọn sẽ được lưu trong biến song_path.
    + initialdir=" /Music": Thư mục mở ban đầu là thư mục Music trong thư mục người dùng.
    + title="Choose a song": Tiêu đề của hộp thoại là "Choose a song".
    + filetypes=(("mp3 Files", "*.mp3"),): Chỉ cho phép chọn các tệp có đuôi .mp3.
- **delete_song(self)**: giúp người dùng xóa một bài hát khỏi playlist. Nó thực hiện các bước sau:
  + Lấy chỉ mục của bài hát đang được chọn trong playlist.

+ Kiểm tra xem có bài hát nào được chọn hay không.
+ Nếu có, xóa bài hát khỏi hệ thống lưu trữ và cập nhật giao diện người dùng để loại bỏ bài hát khỏi playlist.

- **refresh(self)**: Phương thức giúp cập nhật danh sách bài hát của playlist trong giao diện người dùng bằng cách
+ Xóa toàn bộ danh sách hiện tại.
+ Lấy danh sách mới nhất các bài hát từ hệ thống.
+ Thêm từng bài hát từ danh sách mới vào song_box để hiển thị.
Phương thức này đảm bảo rằng danh sách bài hát của playlist luôn được cập nhật và hiển thị đúng nhất theo dữ liệu hiện tại trong hệ thống.

- **search(self)**: Phương thức giúp người dùng tìm kiếm các bài hát dựa trên từ khóa tìm kiếm và hiển thị kết quả trong giao diện người dùng bằng cách:
+ Xóa toàn bộ danh sách hiện tại.
+ Lấy danh sách các bài hát phù hợp với từ khóa tìm kiếm từ hệ thống.
+ Thêm từng bài hát từ danh sách kết quả vào song_box để hiển thị.
Hàm này giúp người dùng dễ dàng tìm kiếm và xem các bài hát có trong playlist dựa trên từ khóa nhập vào.

- **initBtnContainer(self)**: Phương thức này giúp xây dựng giao diện điều khiển cho ứng dụng phát nhạc, cung cấp các chức năng cơ bản để người dùng tương tác với danh sách bài hát và điều khiển việc phát nhạc.

- **Ngoài ra còn có các phương thức**:
+ play_time
+ slide
+ on_click_play
+ on_click_pause, on_click_previous, on_click_next
=> Các phương thức này hoạt động y như trong song_feature.py

4.1.6 playlist_bll.py và song_bll.py

Lớp PlaylistBLL, SongBLL đóng vai trò là tầng logic nghiệp vụ, trung gian giữa giao diện người dùng và cơ sở dữ liệu. Nó cung cấp các phương thức để lưu, kiểm tra, lấy và xóa playlist, bài hát thông qua việc sử dụng lớp ConnectSQLite3.

```
1
2 class PlaylistBLL:
3     def __init__(self):
4         self.connectDB = ConnectSQLite3()
5
6     def savePlaylist(self, playlist: PlaylistDTO):
7         self.connectDB.savePlaylist(playlist)
8
9     def isPlaylistExisted(self, playlist_name):
10        return self.connectDB.isPlaylistExisted(playlist_name)
11
12    def fetchPlaylists(self):
13        rows = self.connectDB.fetchPlaylists()
14        return rows
```

```
15
16    def fetchPlaylist(self, playlist_name):
17        rows = self.connectDB.fetchPlaylist(playlist_name)
18        return rows
19
20    def deletePlaylist(self, playlist_name):
21        self.connectDB.deletePlaylist(playlist_name)
```

```
1
2 class SongBLL:
3     def __init__(self):
4         self.connectDB = ConnectSQLite3()
5
6     def saveSong(self, song: SongDTO):
7         self.connectDB.saveSong(song)
8
9     def fetchSongs(self):
10        songs = self.connectDB.fetchSongs()
11        return songs
12
13    def fetchSong(self, song_name):
14        songs = self.connectDB.fetchSong(song_name)
15        return songs
16
17    def fetchSongsInPlaylist(self, playlist_name):
18        songs = self.connectDB.fetchSongsInPlaylist(playlist_name)
19        return songs
20
21    def isSongExisted(self, song_name):
22        return self.connectDB.isSongExisted(song_name)
23
24    def getSongPath(self, song_name):
25        return self.connectDB.getSongPath(song_name)
26
27    def deleteSong(self, song_name):
28        self.connectDB.deleteSong(song_name)
29
30    def addSongToPlaylist(self, playlist_name, song_name):
31        self.connectDB.addSongToPlaylist(playlist_name, song_name)
32
33    def deleteSongInPlaylist(self, playlist_name, song_name):
34        self.connectDB.deleteSongInPlaylist(playlist_name, song_name)
35
36    def fetchSongInPlaylist(self, playlist_name, song_name):
37        return self.connectDB.fetchSongInPlaylist(playlist_name, song_name)
```

4.1.7 playlist_dto.py và song_dto.py

PlaylistDTO và SongDTO được sử dụng để truyền dữ liệu giữa các thành phần khác nhau trong ứng dụng, như giữa tầng giao diện người dùng và tầng xử lý logic nghiệp vụ.

```
1
2class PlaylistDTO:
3    def __init__(self, name):
4        self.name = name
5
6    def getName(self):
7        return self.name
```

```
1
2class SongDTO:
3    def __init__(self, name, path):
4        self.name = name
5        self.path = path
6
7    def getName(self):
8        return self.name
9
10   def getPath(self):
11       return self.path
```

4.1.8 connect_sqlite3.py

Lớp ConnectSQLite3 cung cấp các phương thức để thực hiện các thao tác cơ bản như thêm, sửa, xóa và truy vấn dữ liệu trong cơ sở dữ liệu SQLite cho các thực thể như bài hát và playlist. Điều này giúp quản lý dữ liệu trong ứng dụng âm nhạc một cách hiệu quả.

```
1
2class ConnectSQLite3:
3    def __init__(self):
4        self.conn = sqlite3.connect("musicapp.db")
5
6        self.cursor = self.conn.cursor()
7
8        self.createTables()
9
10   def createTables(self):
11       self.cursor.execute("""CREATE TABLE IF NOT EXISTS playlists (
12               name text PRIMARY KEY)""")
13
14       self.cursor.execute("""CREATE TABLE IF NOT EXISTS songs (
15               name text PRIMARY KEY,
16               path text)""")
17
```

```
18          self.cursor.execute("""CREATE TABLE IF NOT EXISTS detailed_playlists (
19                  playlist_name text,
20                  song_name text,
21                  foreign key (playlist_name) references playlists (name),
22                  foreign key (song_name) references songs (name)
23          )""")
24
25          self.conn.commit()
26
27      def saveSong(self, song: SongDTO):
28          self.cursor.execute("INSERT INTO songs (name, path) VALUES (?, ?)",
                (song.getName(), song.getPath()))
29          self.conn.commit()
30
31      def fetchSongs(self):
32          self.cursor.execute("SELECT * FROM songs")
33          rows = self.cursor.fetchall()
34          return rows
35
36      def fetchSong(self, song_name):
37          self.cursor.execute("SELECT * FROM songs WHERE name LIKE ?", ('%' +
                song_name + '%',))
38          rows = self.cursor.fetchall()
39          return rows
40
41      def fetchPlaylists(self):
42          self.cursor.execute("SELECT * FROM playlists")
43          rows = self.cursor.fetchall()
44          return rows
45
46      def fetchPlaylist(self, playlist_name):
47          self.cursor.execute("SELECT * FROM playlists WHERE name LIKE ?", ('%' +
                playlist_name + '%',))
48          rows = self.cursor.fetchall()
49          return rows
50
51      def fetchSongsInPlaylist(self, playlist_name):
52          self.cursor.execute("SELECT song_name FROM detailed_playlists WHERE
                playlist_name=?", (playlist_name,))
53          rows = self.cursor.fetchall()
54          return rows
55
56      def fetchSongInPlaylist(self, playlist_name, song_name):
57          self.cursor.execute("SELECT * FROM detailed_playlists WHERE playlist_name =
                ? and song_name LIKE ?",
58                              (playlist_name, '%' + song_name + '%',))
59          rows = self.cursor.fetchall()
60          return rows
61
62
63      def isSongExisted(self, song_name):
64          self.cursor.execute("SELECT * FROM songs WHERE name=?", (song_name,))
```

```
65          rows = self.cursor.fetchall()
66          return bool(rows)
67
68      def getSongPath(self, song_name):
69          self.cursor.execute("SELECT path FROM songs WHERE name=?", (song_name,))
70          rows = self.cursor.fetchall()
71          return rows
72
73      def deleteSong(self, song_name):
74          self.cursor.execute("delete from songs where name=?", (song_name,))
75          self.conn.commit()
76
77      def savePlaylist(self, playlist: PlaylistDTO):
78          self.cursor.execute("INSERT INTO playlists (name) VALUES (?)",
                (playlist.getName(),))
79          self.conn.commit()
80
81      def isPlaylistExisted(self, playlist_name):
82          self.cursor.execute("select * from playlists where name=?", (playlist_name,))
83          rows = self.cursor.fetchall()
84          return bool(rows)
85
86      def deletePlaylist(self, playlist_name):
87          self.cursor.execute("delete from playlists where name=?", (playlist_name,))
88          self.conn.commit()
89
90      def addSongToPlaylist(self, playlist_name, song_name):
91          self.cursor.execute("INSERT INTO detailed_playlists (playlist_name,
                song_name) VALUES (?, ?)",
92                              (playlist_name, song_name,))
93          self.conn.commit()
94
95      def deleteSongInPlaylist(self, playlist_name, song_name):
96          self.cursor.execute("delete from detailed_playlists where playlist_name=?
                and song_name=?",
97                              (playlist_name, song_name,))
98          self.conn.commit()
99
100     def closeConnection(self):
101         self.conn.close()
```

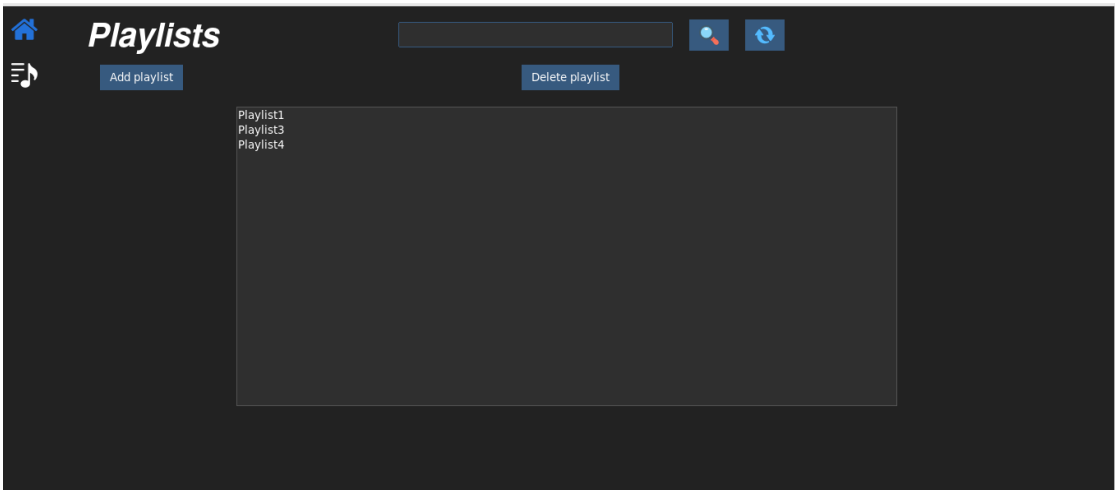
- **__init__(self)**: Phương thức khởi tạo của lớp, tạo kết nối đến cơ sở dữ liệu SQLite và tạo bảng nếu chưa tồn tại.
- **createTables(self)**: Tạo bảng trong cơ sở dữ liệu nếu chưa tồn tại.
- **saveSong(self, song: SongDTO)**: Thêm một bài hát mới vào bảng songs.
- **fetchSongs(self)**: Lấy tất cả các bài hát từ bảng songs.
- **fetchSong(self, song_name)**: Tìm kiếm bài hát theo tên.
- **savePlaylist(self, playlist: PlaylistDTO)**: Thêm một playlist mới vào bảng playlists.

- **fetchPlaylists(self)**: Lấy tất cả các playlist từ bảng playlists.
- **fetchPlaylist(self, playlist_name)**: Tìm kiếm playlist theo tên.
- **fetchSongsInPlaylist(self, playlist_name)**: Lấy tất cả các bài hát trong một playlist.
- **fetchSongInPlaylist(self, playlist_name, song_name)**: Tìm kiếm bài hát trong một playlist.
- **addSongToPlaylist(self, playlist_name, song_name)**: Thêm một bài hát vào một playlist.
- **deleteSongInPlaylist(self, playlist_name, song_name)**: Xóa một bài hát khỏi một playlist.
- **isSongExisted(self, song_name)**: Kiểm tra xem một bài hát có tồn tại hay không.
- **isPlaylistExisted(self, playlist_name)**: Kiểm tra xem một playlist có tồn tại hay không.
- **deleteSong(self, song_name)**: Xóa một bài hát khỏi cơ sở dữ liệu.
- **deletePlaylist(self, playlist_name)**: Xóa một playlist khỏi cơ sở dữ liệu.
- **closeConnection(self)**: Đóng kết nối đến cơ sở dữ liệu.

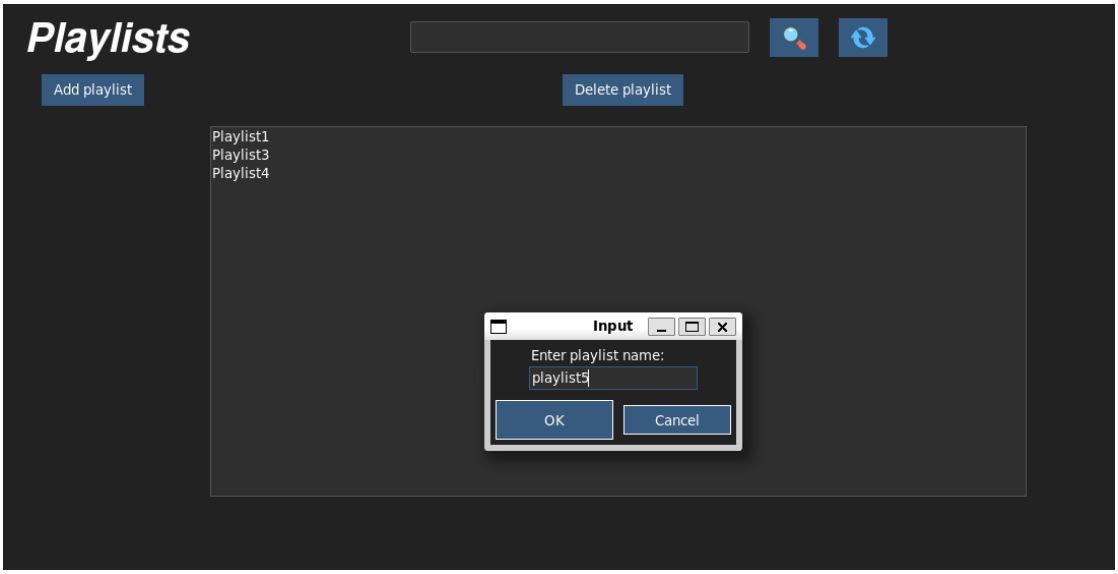
4.1.9 musicapp.db

File database được tạo ra bởi lớp ConnectSQLite3 trong connect_sqlite3.py

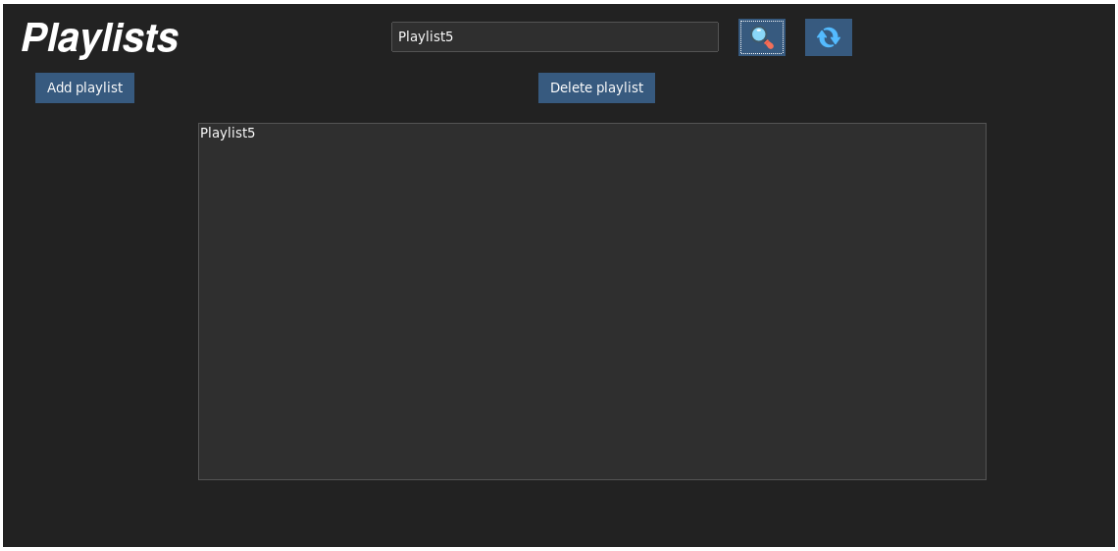
4.2 Giao diện



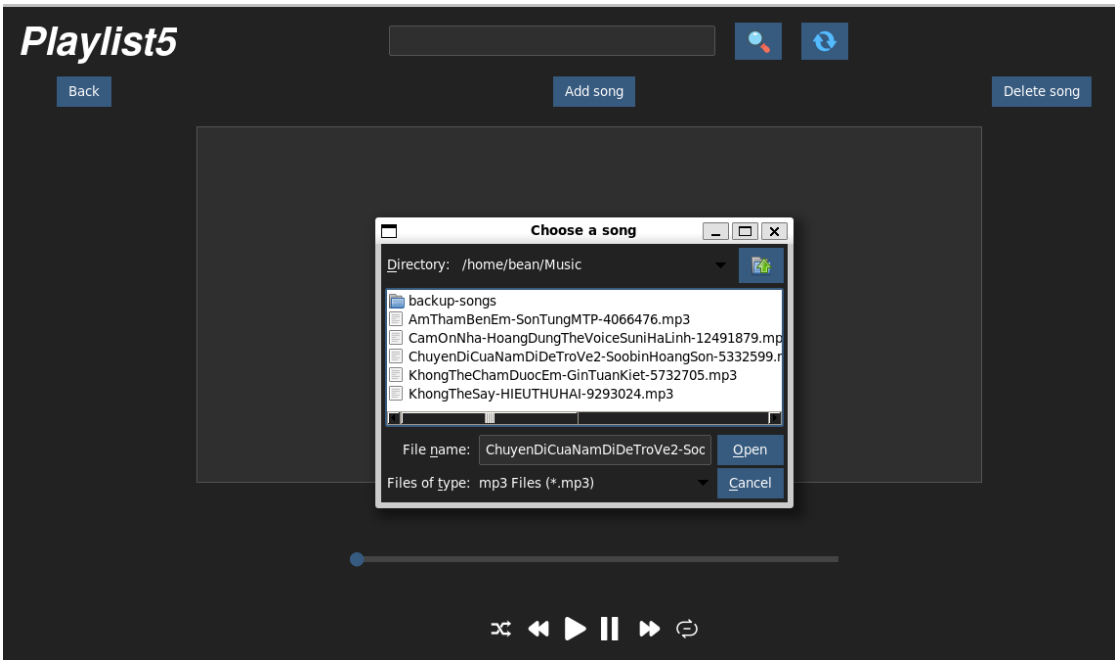
Hình 2: Quản Lý Playlist



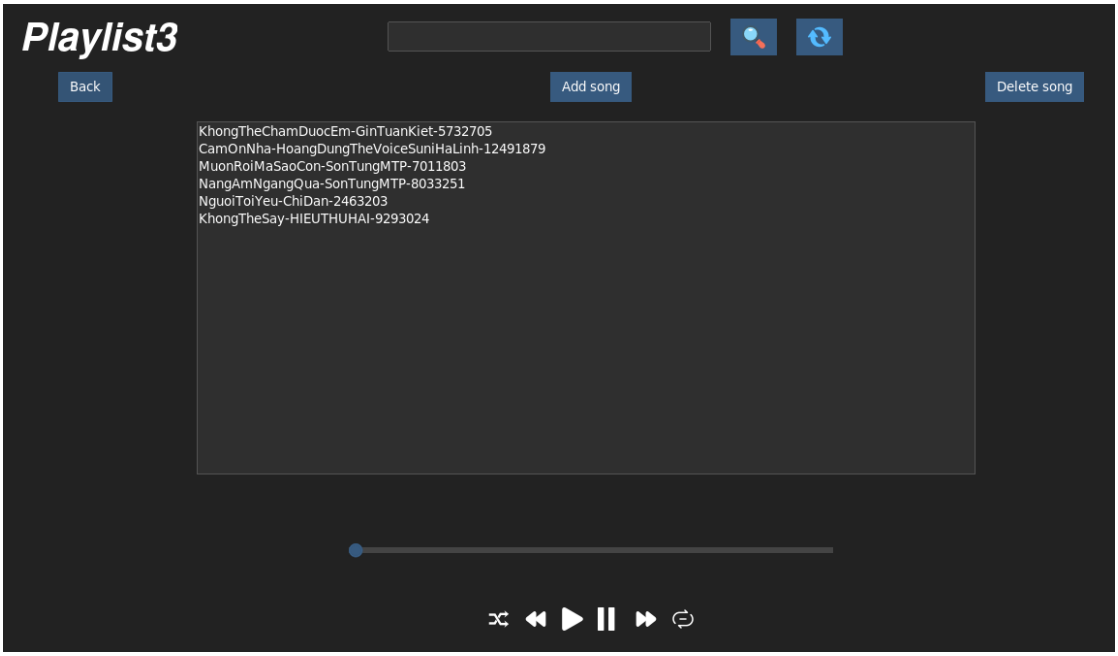
Hình 3: Tạo Playlist



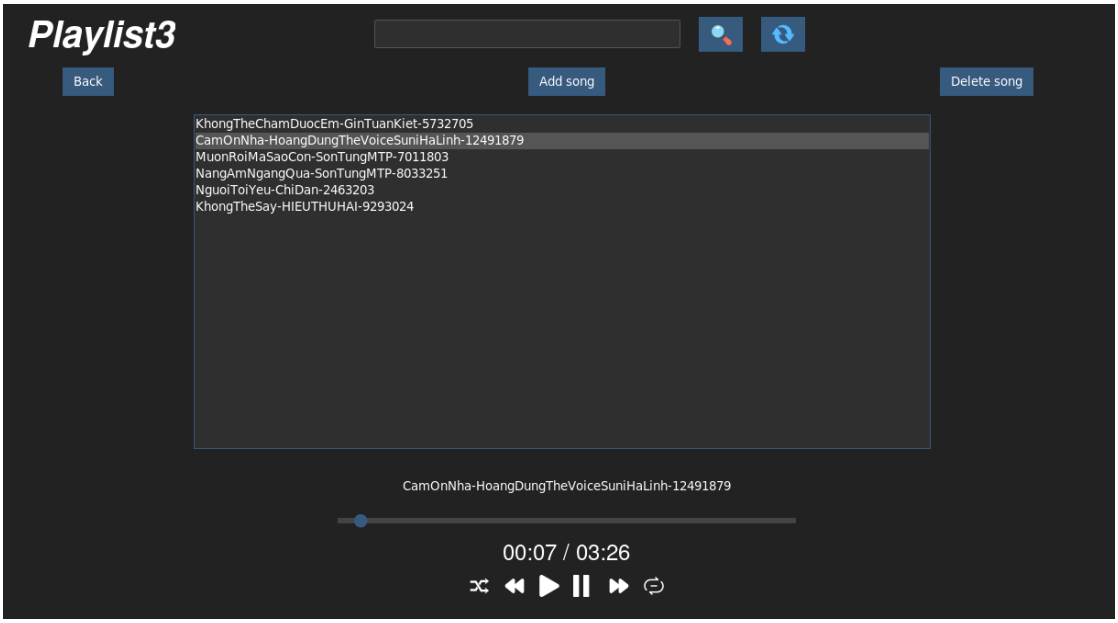
Hình 4: Tìm kiếm Playlist



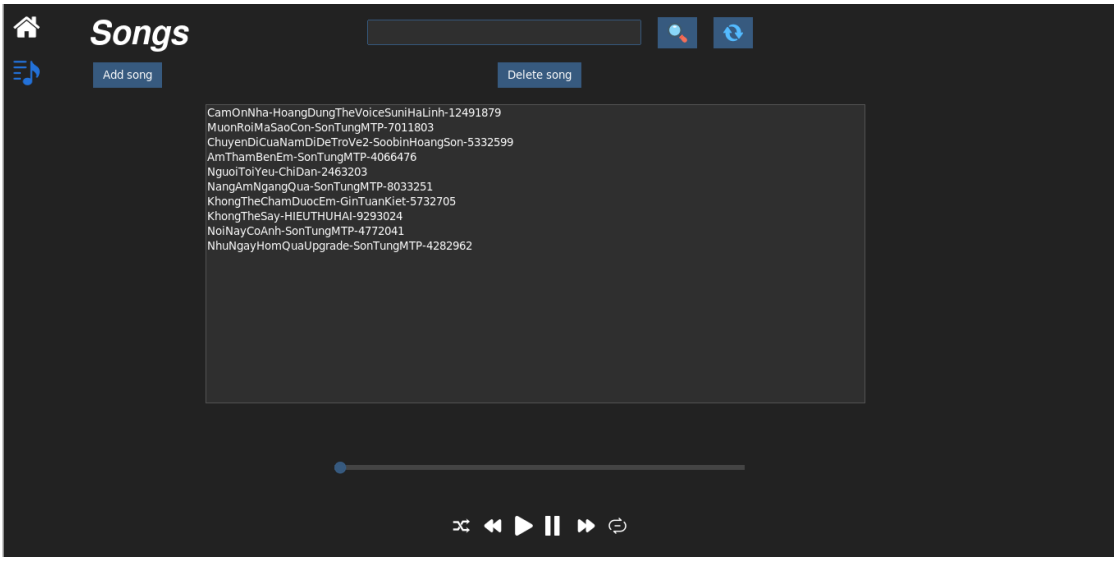
Hình 5: Thêm Bài Hát Vào Playlist



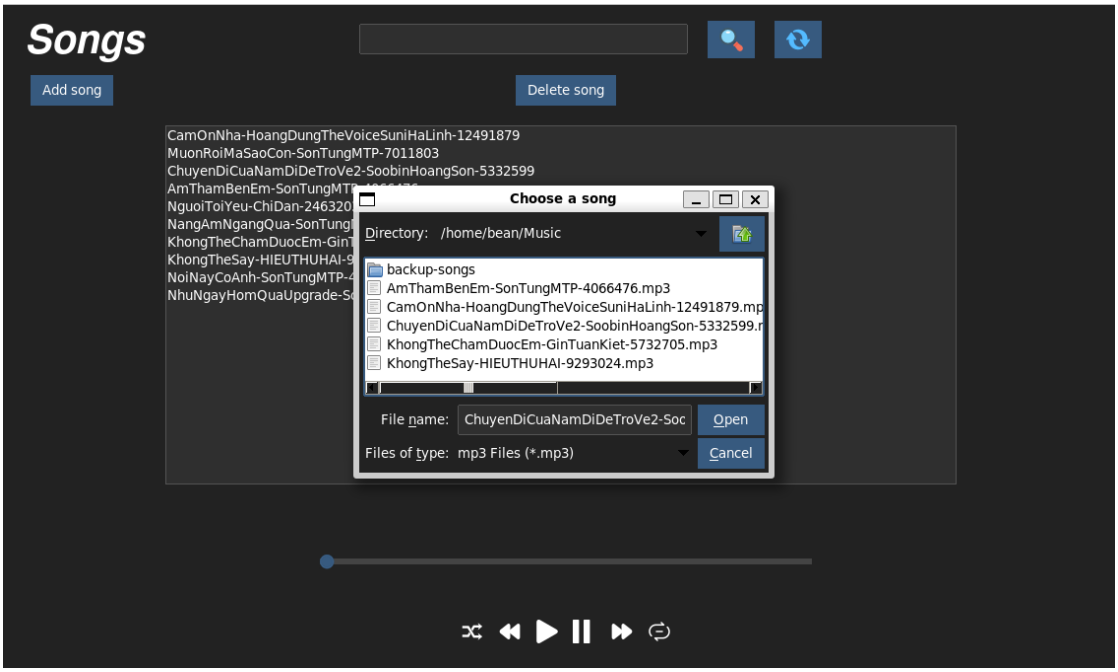
Hình 6: Danh sách các bài hát trong 1 playlist bất kỳ



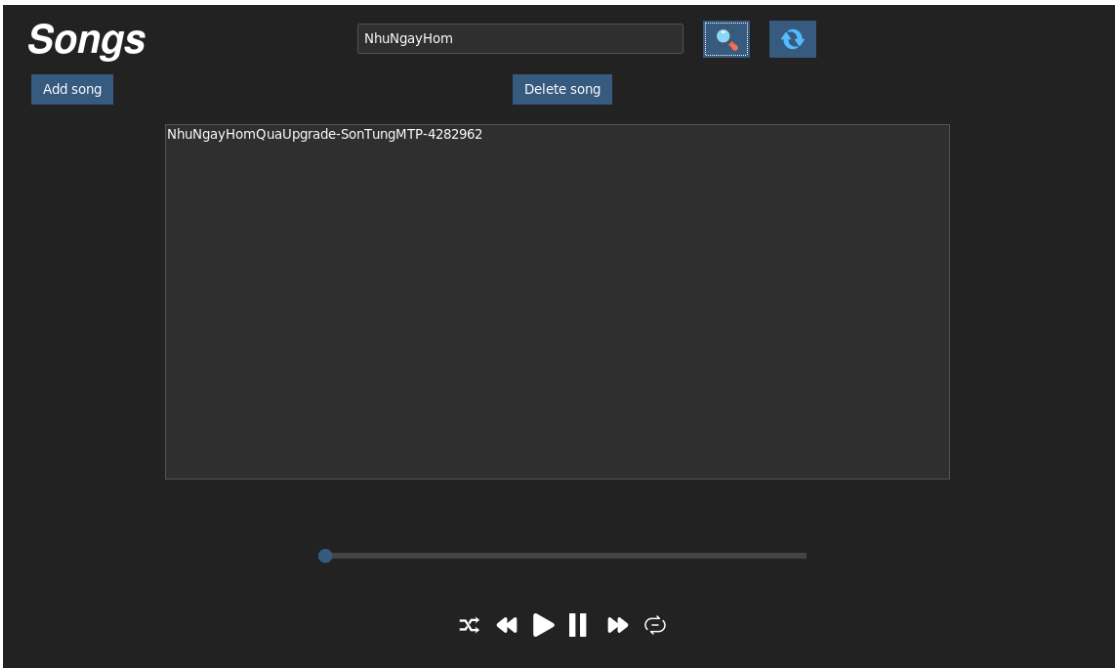
Hình 7: Play 1 Bài Hát



Hình 8: Quản Lý Bài Hát



Hình 9: Thêm Bài Hát



Hình 10: Tìm Kiếm Bài Hát

5 Cài đặt ứng dụng

- Sử dụng câu lệnh ***git clone -b master –single-branch https://github.com/phvu0430/Tkinter-Music-App.git*** để tải ứng dụng về máy.
- Sau đó di chuyển vào thư mục ***Tkinter-Music-App*** và chạy lệnh ***python3 main.py*** để khởi động ứng dụng.

Tài liệu

[1] John Elder, ***Tkinter Widget Quick Reference Guide***, *Independently published, 2018.*

[2] TTKBootstrap "**link: https://ttkbootstrap.readthedocs.io/en/latest/styleguide/**", *style guide*, lần truy cập cuối: 25/04/2024.

[3] Pygame "**link: https://www.pygame.org/docs/ref/music.html/**", *pygame.mixer.music*, lần truy cập cuối: 03/05/2024.