

TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN



PHÁT TRIỂN PHẦN MỀM MÃ NGUỒN MỞ

Phát triển

Ứng dụng nghe nhạc trên PYTHON

GVHD: Từ Lăng Phiêu
SV: Nguyễn Anh Danh - 3121410103
Phan Duy - 3121410003
Văn Phú Hiếu - 3121410201
Đỗ Nguyễn Hoàng Tuấn - 3121410554

TP. HỒ CHÍ MINH, THÁNG 5/2024



Lời cảm ơn

Chúng em xin gửi lời cảm ơn chân thành nhất đối với các thầy cô ở khoa Công Nghệ Thông Tin, trường Đại học Sài Gòn đã tạo điều kiện cho chúng em tiếp cận và tìm hiểu để hoàn thành đồ án môn học lần này. Và chúng em cũng xin chân thành cảm ơn thầy Từ Lăng Phiêu giáo viên giảng dạy đã nhiệt tình hướng dẫn chúng em hoàn thành đồ án lần này. Trong quá trình thực hiện nghiên cứu và thực hiện làm báo cáo đồ án, do kinh nghiệm thực tế chưa được nhiều, nên bài báo cáo của chúng em có thể vẫn còn những thiếu sót và chưa được hoàn chỉnh nên mong rằng chúng em sẽ nhận được những đóng góp ý kiến đóng góp bổ ích từ thầy để chúng em có thể khắc phục cho những bài báo cáo sau.

Chúng em xin trân trọng cảm ơn thầy!



Mục lục

1	Phần 1: Mở đầu	4
1.1	Lý do chọn đề tài	4
1.2	Mục đích - mục tiêu của đề tài	4
1.3	Phạm vi đề tài	4
1.4	Nội dung đề tài	4
2	Xây dựng ứng dụng nghe nhạc trên Python bằng các thư viện của Python	5
2.1	Đôi nét về ứng dụng nghe nhạc	5
2.2	Tổng quan và phân tích	5
2.2.1	Khảo sát	5
2.2.2	Phân tích	6
2.3	Xây dựng ứng dụng nghe nhạc	8
2.3.1	Cài đặt các thư viện cần thiết	8
2.4	Các bước khởi tạo ứng dụng nghe nhạc	8
2.4.1	Khai báo thư viện	8
2.5	Sơ đồ cơ sở dữ liệu	9
2.6	Kiến trúc ứng dụng	11
2.7	Xây dựng chức năng	11



1 Phần 1: Mở đầu

1.1 Lý do chọn đề tài

Công nghệ thông tin ngày càng trở thành một phần không thể thiếu trong cuộc sống hiện đại, và ngôn ngữ lập trình Python đã và đang đóng một vai trò quan trọng trong việc phát triển các ứng dụng công nghệ thông tin. Python với cấu trúc rõ ràng, dễ đọc và dễ học, đã trở thành một trong những lựa chọn hàng đầu cho nhiều lập trình viên trên toàn thế giới.

Âm nhạc là một phần quan trọng của cuộc sống, mang lại niềm vui, sự thư giãn và là nguồn cảm hứng cho con người. Với sự phát triển của công nghệ, việc nghe nhạc đã trở nên dễ dàng hơn bao giờ hết. Tuy nhiên, việc tìm kiếm một ứng dụng nghe nhạc phù hợp với nhu cầu cá nhân không phải lúc nào cũng dễ dàng.

Chính vì vậy, chúng em đã chọn đề tài "Phát triển ứng dụng nghe nhạc sử dụng ngôn ngữ Python". Mục tiêu của chúng em là tạo ra một ứng dụng nghe nhạc đơn giản nhưng đầy đủ tính năng, dễ sử dụng và có thể tùy chỉnh theo nhu cầu của người dùng. Chúng em tin rằng, với sự linh hoạt và mạnh mẽ của Python, chúng em có thể đạt được mục tiêu này.

1.2 Mục đích - mục tiêu của đề tài

-Mục đích

- Nắm chắc được kỹ năng và kiến thức về ngôn ngữ lập trình Python.
- Tìm hiểu về cách thức hoạt động của một ứng dụng nghe nhạc.
- Tìm hiểu về thư viện Pygame, MySQL Connector, ...
- Cũng cố, áp dụng, nâng cao kiến thức đã học.

-Mục tiêu

- Vận dụng được tính chất của lập trình hướng đối tượng.
- Xây dựng một ứng dụng nghe nhạc đơn giản, dễ sử dụng.

1.3 Phạm vi đề tài

- Ứng dụng có thể tìm kiếm, phát nhạc từ cơ sở dữ liệu.
- Ứng dụng có thể tạo danh sách phát, tạo playlist.

1.4 Nội dung đề tài

Bao gồm 2 phần:

- Phần 1: Mở đầu
- Phần 2: Thực hiện ứng dụng nghe nhạc trên Python
 - Mở đầu
 - Xây dựng ứng dụng nghe nhạc bằng cách sử dụng các thư viện của Python:
 - * Phân tích yêu cầu

- * Thiết kế kiến trúc
- * Thiết kế cơ sở dữ liệu
- * Xây dựng
- * Xây dựng chức năng
- * Kiểm thử và sửa lỗi

2 Xây dựng ứng dụng nghe nhạc trên Python bằng các thư viện của Python

2.1 Đôi nét về ứng dụng nghe nhạc

-Ứng dụng nghe nhạc không chỉ là một công cụ giúp người dùng tìm kiếm và phát nhạc từ cơ sở dữ liệu. Nó còn là một nền tảng giúp người dùng trải nghiệm âm nhạc theo cách riêng của họ.

-Tìm kiếm và phát nhạc: Ứng dụng nghe nhạc cho phép người dùng tìm kiếm bài hát, album, nghệ sĩ yêu thích của họ từ một cơ sở dữ liệu lớn. Người dùng có thể phát nhạc trực tiếp từ ứng dụng, điều chỉnh âm lượng, chọn chế độ phát (như phát lại, lặp lại, ngẫu nhiên), và xem thông tin chi tiết về bài hát đang phát.

-Tạo danh sách phát: Người dùng có thể tạo danh sách phát cá nhân, thêm bài hát vào danh sách phát, sắp xếp thứ tự các bài hát trong danh sách phát, và chia sẻ danh sách phát với bạn bè. Điều này giúp người dùng tổ chức bộ sưu tập âm nhạc của họ theo cách mà họ muốn.

-Tạo playlist: Playlist là một tính năng mạnh mẽ giúp người dùng tổ chức và phát nhạc theo chủ đề, tâm trạng, hoặc sự kiện. Người dùng có thể tạo playlist, thêm bài hát vào playlist, và chia sẻ playlist với cộng đồng.

-Khám phá âm nhạc mới: Ứng dụng nghe nhạc thường có tính năng khám phá, giúp người dùng tìm kiếm và khám phá âm nhạc mới dựa trên sở thích âm nhạc của họ. Điều này giúp người dùng mở rộng bộ sưu tập âm nhạc của họ và khám phá những nghệ sĩ, thể loại mới.

-Như vậy, ứng dụng nghe nhạc không chỉ giúp người dùng nghe nhạc, mà còn giúp họ trải nghiệm âm nhạc theo cách riêng của họ, khám phá âm nhạc mới, và chia sẻ niềm đam mê âm nhạc với cộng đồng. Đây chính là lý do mà việc phát triển ứng dụng nghe nhạc sử dụng Python trở nên hấp dẫn và thú vị. Python với khả năng mạnh mẽ và linh hoạt của mình, cho phép chúng ta tạo ra những ứng dụng nghe nhạc phong phú và đa dạng, phục vụ cho nhu cầu ngày càng đa dạng của người dùng.

2.2 Tổng quan và phân tích

2.2.1 Khảo sát

-Ứng dụng nghe nhạc là một nền tảng trực tuyến phổ biến, đặc biệt trong cộng đồng người yêu âm nhạc và các nhóm cộng đồng trực tuyến khác. Một trong những lợi ích của ứng dụng nghe nhạc là tính linh hoạt và đa dạng của nó. Người dùng có thể tùy chỉnh các danh sách phát và quyền truy cập cho từng bài hát, tạo ra các thể loại khác nhau để quản lý bộ sưu tập âm nhạc và tùy chỉnh các cài đặt âm thanh cho phù hợp với nhu cầu của mình.

-Việc tạo ứng dụng nghe nhạc cũng là một lợi ích lớn, giúp việc quản lý bộ sưu tập âm nhạc trở nên dễ dàng hơn và giảm thiểu thời gian và công sức cho các hoạt động quản lý. Ứng dụng có thể tự động thực hiện các nhiệm vụ như kiểm tra và cập nhật thông tin bài hát, quản lý danh sách phát và nhiều tính năng khác ...

-Tuy nhiên, ứng dụng nghe nhạc cũng có một số hạn chế như việc không thể tùy chỉnh giao diện của ứng dụng hoặc các danh sách phát quá nhiều.



2.2.2 Phân tích



Thư viện	Mô tả
socket	Thư viện socket trong Python cung cấp các hàm để tạo và quản lý kết nối mạng. Cho phép tạo ra các ứng dụng mạng phức tạp như truyền file, gửi và nhận dữ liệu qua mạng, và nhiều hơn nữa. Thư viện socket hỗ trợ các giao thức mạng phổ biến như TCP và UDP, cho phép lập trình viên tương tác với các máy chủ và thiết bị khác trên mạng. Bằng cách sử dụng thư viện socket, lập trình viên có thể xây dựng các ứng dụng mạng linh hoạt và mạnh mẽ trên nền tảng Python.
os	Thư viện os trong Python cung cấp các hàm để tương tác với hệ điều hành. Cho phép thực hiện các tác vụ như quản lý file, thư mục, và các tác vụ liên quan đến hệ điều hành khác. Thư viện os giúp lập trình viên tạo ra các ứng dụng có khả năng tương tác mạnh mẽ với hệ điều hành.
sys	Thư viện sys trong Python cung cấp các hàm để tương tác với hệ thống Python. Cho phép truy cập vào các biến và hàm của hệ thống, quản lý luồng dữ liệu vào ra, và thực hiện các tác vụ liên quan đến hệ thống khác. Thư viện sys giúp lập trình viên tạo ra các ứng dụng có khả năng tương tác mạnh mẽ với hệ thống Python.
threading	Thư viện threading trong Python cung cấp các hàm để tạo và quản lý các luồng. Cho phép tạo ra các ứng dụng đa luồng, tận dụng tối đa khả năng của CPU và tăng hiệu suất của ứng dụng. Thư viện threading giúp lập trình viên tạo ra các ứng dụng đa luồng mạnh mẽ và hiệu quả.
mysql.connector	Thư viện mysql.connector trong Python cung cấp các hàm để tương tác với cơ sở dữ liệu MySQL. Cho phép tạo ra các ứng dụng có khả năng tương tác mạnh mẽ với cơ sở dữ liệu, thực hiện các tác vụ như truy vấn, cập nhật, và quản lý dữ liệu. Thư viện mysql.connector giúp lập trình viên tạo ra các ứng dụng có khả năng tương tác mạnh mẽ với cơ sở dữ liệu MySQL.
pygame	Thư viện pygame trong Python cung cấp các hàm để tạo ra các ứng dụng đồ họa, bao gồm các trò chơi và các ứng dụng đa phương tiện khác. Tạo ra các ứng dụng có đồ họa mạnh mẽ, tương tác với người dùng qua các sự kiện đầu vào, và tạo ra các hiệu ứng âm thanh và hình ảnh. Thư viện pygame giúp lập trình viên tạo ra các ứng dụng đồ họa mạnh mẽ và tương tác.
tkinter	Thư viện tkinter trong Python cung cấp các hàm để tạo ra các ứng dụng đồ họa, các ứng dụng đa phương tiện khác. Tạo ra các ứng dụng có đồ họa mạnh mẽ, tương tác với người dùng qua các sự kiện đầu vào, và tạo ra các hiệu ứng âm thanh và hình ảnh.

Bảng 1: Các thư viện Python được sử dụng cho việc phát triển ứng dụng

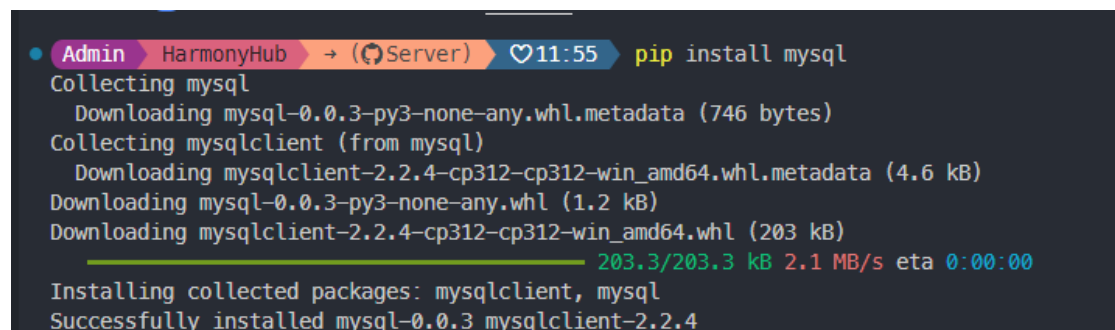
2.3 Xây dựng ứng dụng nghe nhạc

2.3.1 Cài đặt các thư viện cần thiết

-Visual Studio Code (64-bit).

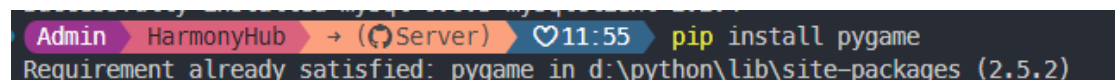
-Điều đầu tiên cần làm để lập trình ứng dụng nghe nhạc trên Python là cài đặt các thư viện cần thiết. Các thư viện này giúp chúng ta tương tác với hệ thống, tạo ra các ứng dụng đa luồng, tương tác với cơ sở dữ liệu, và tạo ra các ứng dụng đồ họa mạnh mẽ.

Sau đó hệ thống sẽ tự động cài đặt các thư viện cần thiết:



```
Admin HarmonyHub → (Server) 11:55 pip install mysql
Collecting mysql
  Downloading mysql-0.0.3-py3-none-any.whl.metadata (746 bytes)
Collecting mysqlclient (from mysql)
  Downloading mysqlclient-2.2.4-cp312-win_amd64.whl.metadata (4.6 kB)
  Downloading mysql-0.0.3-py3-none-any.whl (1.2 kB)
  Downloading mysqlclient-2.2.4-cp312-cp312-win_amd64.whl (203 kB)
    203.3/203.3 kB 2.1 MB/s eta 0:00:00
Installing collected packages: mysqlclient, mysql
Successfully installed mysql-0.0.3 mysqlclient-2.2.4
```

Hình 1: Cài đặt thư viện MySQL



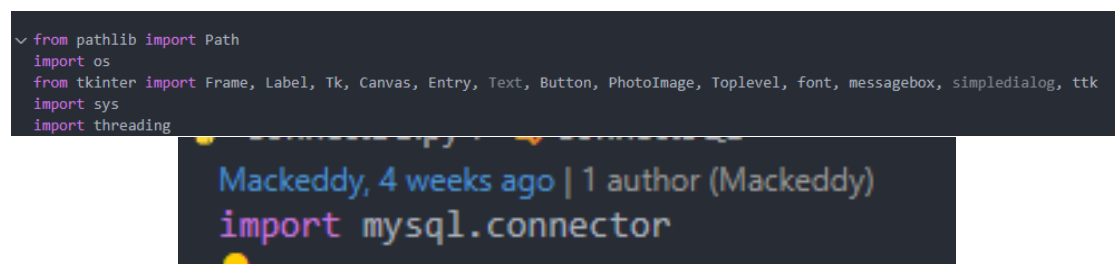
```
Admin HarmonyHub → (Server) 11:55 pip install pygame
Requirement already satisfied: pygame in d:\python\lib\site-packages (2.5.2)
```

Hình 2: Cài đặt thư viện Pygame (Do đã cài đặt nên ta sẽ không cần cài nữa)

2.4 Các bước khởi tạo ứng dụng nghe nhạc

2.4.1 Khai báo thư viện

Sau khi cài đặt các thư viện cần thiết, ta tiến hành khai báo các thư viện cần dùng:



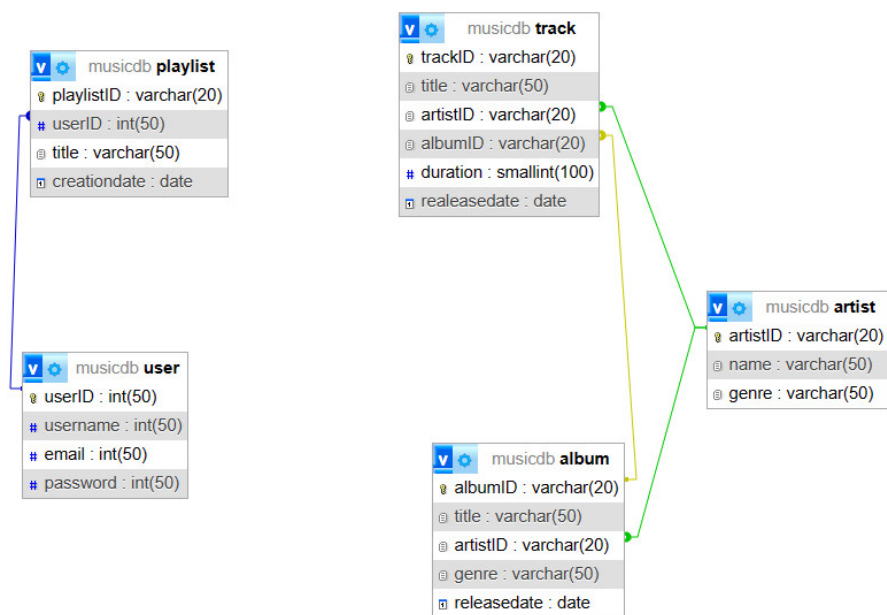
```
from pathlib import Path
import os
from tkinter import Frame, Label, Tk, Canvas, Entry, Text, Button, PhotoImage, Toplevel, font, messagebox, simpledialog, ttk
import sys
import threading

Mackeddy, 4 weeks ago | 1 author (Mackeddy)
import mysql.connector
```

Hình 3: Khai báo thư viện

2.5 Sơ đồ cơ sở dữ liệu

- Để lưu trữ thông tin về bài hát, album, nghệ sĩ, và các thông tin khác, chúng ta cần tạo một cơ sở dữ liệu.
- Chúng ta đồng thời sẽ sử dụng XAMPP để tạo cơ sở dữ liệu MySQL và thiết kế cơ sở dữ liệu cho ứng dụng trên đó.
- Chúng ta có sơ đồ cơ sở dữ liệu như sau:



Hình 4: Sơ đồ cơ sở dữ liệu cho ứng dụng nghe nhạc

-Sau khi chúng ta đã có sơ đồ cơ sở dữ liệu, chúng ta sẽ tiến hành kết nối cơ sở dữ liệu với ứng dụng thông qua thư viện MySQL Connector và thực hiện các thao tác truy vấn, cập nhật dữ liệu từ cơ sở dữ liệu.

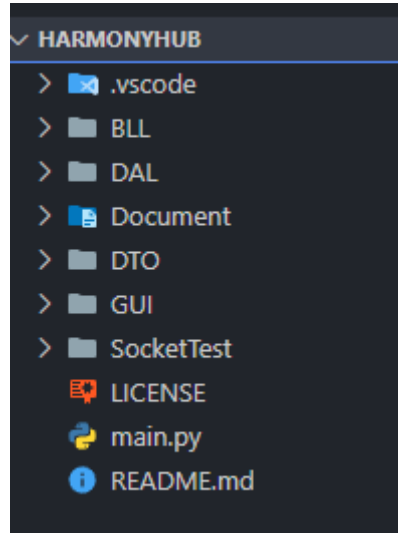
```
connectdb.py 7/11
Mackeddy, 4 weeks ago | 1 author (Mackeddy)
import mysql.connector

Mackeddy, 4 weeks ago | 1 author (Mackeddy)
class ConnectSQL():
    @staticmethod
    def connect_mysql():
        try:
            connection = mysql.connector.connect(
                host='localhost',
                database='musicdb', "musicdb": Unknown word.
                user='root',
                password=''
            )
            if connection.is_connected():
                print("Connected to MySQL database")
                return connection
            else:
                print("Connection failed")
                return None
        except mysql.connector.Error as error:
            print("Error while connecting to MySQL", error)
            return None
```

Hình 5: Khai báo thư viện và kết nối cơ sở dữ liệu thông qua class ConnectDB.py

2.6 Kiến trúc ứng dụng

Về kiến trúc thiết kế mô hình phát triển ứng dụng nghe nhạc, chúng ta sẽ sử dụng mô hình 3 lớp (3-tier architecture) bao gồm 3 lớp chính: DAL, BLL, và GUI.



Hình 6: Kiến trúc ứng dụng nghe nhạc

2.7 Xây dựng chức năng

Để xây dựng chức năng cho ứng dụng nghe nhạc, chúng ta sẽ tạo ra các class đối tượng (Object) tương ứng với các bảng trong cơ sở dữ liệu ở thư mục DTO (Data Transfer Object):

```
MrDemos, 2 weeks ago | 2 authors (Mackedy and others)
▼ class AlbumDTO:
  ▼ def __init__(self, albumID, title, artistID, genre, releasedate):
      self.albumID = albumID
      self.title = title
      self.artistID = artistID
      self.genre = genre
      self.releasedate = releasedate MrDemos, 2 weeks ago • updat
```

```
MrDemoa, 2 weeks ago | 2 authors (Mackeddy and others)
class ArtistDTO: Mackeddy, 4 weeks ago • Them DTO, hoan thien chuc nang DAL, BLL

def __init__(self, artistID, name, genre):
    self.artistID = artistID
    self.name = name
    self.genre = genre

Mackeddy, last week | 1 author (Mackeddy)
class PLDetailDTO: Mackeddy, last week • Them chuc nang DAL, BLL

def __init__(self, PlaylistID, UserID, trackID):
    self.PlaylistID = PlaylistID
    self.UserID = UserID
    self.trackID = trackID

Mackeddy, last week | 1 author (Mackeddy)
class PlayListDTO:
    def __init__(self, playlistID, userID, trackID, title, createiondate):
        self.playlistID = playlistID "playlist": Unknown word.
        self.userID = userID
        self.trackID = trackID
        self.title = title
        self.createiondate = createiondate Mackeddy, 4 weeks ago • Them DTO, hoan thien chuc nang DAL, BLL

MrDemoa, 2 weeks ago | 2 authors (Mackeddy and others)
class TrackDTO: Mackeddy, 4 weeks ago • Them DTO, hoan thien chuc nang DAL, BLL

def __init__(self, trackID, title, artistID, albumID, duration, releasedate):
    self.trackID = trackID
    self.title = title
    self.artistID = artistID
    self.albumID = albumID
    self.duration = duration
    self.releasedate = releasedate "releasedate": Unknown word.
```

```
Mackeddy, 4 weeks ago | 1 author (Mackeddy)  
class UserDTO:      Mackeddy, 4 weeks ago • Them DTO, hoan t  
    def __init__(self, userID, username, email, password):  
        self.userID = userID  
        self.username = username  
        self.email = email  
        self.password = password
```

Hình 7: Các class DTO

-Sau khi chúng ta đã tạo các class DTO, chúng ta sẽ tiến hành tạo các class ở tầng DAL (Data Access Layer) để thực hiện các thao tác truy vấn, cập nhật dữ liệu từ cơ sở dữ liệu:

```
PlaylistDAL.py X
DAL > PlaylistDAL.py > ...
Mackedy, last week | 2 authors (Mackedy and others)
1 import os
2 import sys
3 sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
4
5
6 from DAL.ConnectDB import ConnectSQL
7 from DTO.PlayListDTO import PlayListDTO
8
9 Mackedy, last week | 2 authors (Mackedy and others)
10 class PlaylistDAL:
11
12     con = ConnectSQL.connect_mysql()
13
14     def getAllData(self):
15         global con
16         cursor = PlaylistDAL.con.cursor()
17         cursor.execute("select * from playlist")
18         records = cursor.fetchall()
19         cursor.close()
20         return records
21
22     def getDataPlaylistFromUserID(self, userID):
23         cursor = self.con.cursor()
24         cursor.execute("select playlistID, title, creationdate from playlist where userID = %s", (userID))
25         records = cursor.fetchall()
26         cursor.close()
27         return records
28
29     def generatePlaylistID(self):
30         cursor = self.con.cursor()
31         cursor.execute("select PlaylistID from playlist order by playlistID desc limit 1")
32         playlist_id = cursor.fetchone()
33         if playlist_id:
34             id = playlist_id[0]
35             id = int(id[1:]) + 1
36         else:
37             id = 1
38         return "PL" + str(id)
39
40     def insert(self, playlist_dto):
41         cursor = self.con.cursor()
42         cursor.execute("insert into playlist values(%s, %s, %s, %s)", (playlist_dto.playlistID, playlist_dto.userID, playlist_dto.trackID, playlist_dto.title, playlist_dto.creationdate))
43         self.con.commit()
44         cursor.close()
45
46     def delete(id):
47         global con
48         cursor = con.cursor()
49         cursor.execute("delete from playlist where playlistID = %s", (id,))
50         count = int(cursor.rowcount)
51         con.commit()
52         cursor.close()
53
54         if count > 0:
55             print("Xoa thanh cong")
56         else:
57             print("ma khong ton tai")
58
59     def update(playlist_dto):
60         global con
61         cursor = con.cursor()
62         cursor.execute("update playlist set userID = %s, title = %s, creationdate = %s where playlistID = %s",
63             ("cr", (playlist_dto.userID, playlist_dto.title, playlist_dto.creationdate, playlist_dto.playlistID))
64         )
65         con.commit()
66         cursor.close()
```

Hình 8: Code cho class PlaylistDAL gồm các phương thức CRUD (Create, Read, Update, Delete)

-Giải thích:

- Phương thức **getAllData(self)** trả về danh sách tất cả các playlist trong cơ sở dữ liệu.

+Chi tiết hơn:
 - Đầu tiên, ta sẽ thiết lập kết nối đến cơ sở dữ liệu thông qua class ConnectDB.
 - Sau đó, ta sẽ tạo 1 cursor để thực hiện các thao tác truy vấn dữ liệu.
 - Cuối cùng, ta sẽ thực hiện truy vấn dữ liệu và trả về kết quả lưu về records
- Phương thức **getDataPlayListFromUserId(self, userID)** trả về thông tin chi tiết của một playlist dựa vào id.

+Chi tiết hơn:
 - Đầu tiên, ta sẽ thiết lập kết nối đến cơ sở dữ liệu thông qua class ConnectDB.
 - Sau đó, ta sẽ tạo 1 cursor để thực hiện các thao tác truy vấn dữ liệu.
 - Cuối cùng, ta sẽ thực hiện truy vấn dữ liệu và trả về kết quả lưu về records (trong trường hợp này, ta sẽ truy vấn dữ liệu dựa vào id của user).
- Phương thức **generatePlayListID(self)** tạo một ID duy nhất cho playlist mới.

+Chi tiết hơn:
 - Đầu tiên, ta sẽ thiết lập kết nối đến cơ sở dữ liệu thông qua class ConnectDB.
 - Sau đó, ta sẽ tạo 1 cursor để thực hiện các thao tác truy vấn dữ liệu.
 - Lấy ra kết quả đầu tiên của query, kiểm tra xem nếu nó tồn tại thì lấy ID đó, loại bỏ ký tự đầu tiên (giả sử là 'PL'), chuyển phần còn lại thành số và cộng thêm một. Điều này đảm bảo rằng mỗi playlist có một ID duy nhất.
 - Nếu không nhận được ID trả về, điều đó có nghĩa là chưa có playlist nào trong cơ sở dữ liệu. Vì vậy, nó đơn giản là đặt ID cho playlist mới là 1.
 - Cuối cùng, nó thêm 'PL' vào đầu ID (để chỉ ra rằng đó là Playlist ID) và trả về nó. Đây sẽ là ID duy nhất cho playlist mới.
- Phương thức **insert(self, playlist_dto)** thêm một playlist mới vào cơ sở dữ liệu.

+Chi tiết hơn:
 - Đầu tiên, ta sẽ thiết lập kết nối đến cơ sở dữ liệu thông qua class ConnectDB.
 - Sau đó, ta sẽ tạo 1 cursor để thực hiện các thao tác truy vấn dữ liệu.
 - Sau khi chuẩn bị xong câu lệnh, nó được thực thi bằng cách sử dụng con trỏ (Cursor). Điều này thêm playlist mới vào cơ sở dữ liệu.
 - Tuy nhiên, chỉ thực thi câu lệnh không đủ. Các thay đổi cần được lưu lại. Đó là lý do tại sao self.con.commit() được sử dụng - nó lưu lại bất kỳ thay đổi nào được thực hiện kể từ lần cuối cùng thay đổi được lưu lại.
 - Cuối cùng, con trỏ được đóng. Điều này được thực hiện khi chúng ta đã hoàn thành với nó, để giải phóng tài nguyên.
- Phương thức **deletePlayList(playlistID)** xóa một playlist dựa vào id.



+Chi tiết hơn:

-Đầu tiên, ta thiết lập kết nối và tạo con trỏ (Cursor).

-Sau đó, ta thực thi câu lệnh DELETE để xóa playlist dựa vào id và đếm số lượng dòng bị ảnh hưởng.

-Nếu số lượng dòng bị ảnh hưởng lớn hơn 0, điều đó có nghĩa là playlist đã được xóa thành công.

-Cuối cùng, ta lưu lại các thay đổi và đóng con trỏ.

- Phương thức **updatePlaylist(playlist_dto)** cập nhật thông tin của một playlist.

+Cách thức hoạt động tương tự như phương thức insert, nhưng thay vì thêm mới, nó sẽ cập nhật thông tin của playlist đã tồn tại.

```
AlbumDAL.py
DAL > AlbumDAL.py > AlbumDAL
...
1  import os
2  import sys
3  sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
4
5
6  from DAL.ConnectDB import ConnectSQL
7  from DTO.AlbumDTO import AlbumDTO
8
9  ...
10 class AlbumDAL:
11     con = ConnectSQL.connect_mysql()
12
13     def getAllData(self):
14         global con
15         cursor = AlbumDAL.con.cursor()
16         cursor.execute("select * from album")
17         records = cursor.fetchall()
18         cursor.close()
19         return records
20
21     def generateAlbumID(self):
22         cursor = self.con.cursor()
23         cursor.execute("select albumID from album order by albumID desc limit 1")
24         album_id = cursor.fetchone()
25         if album_id :
26             id = album_id[0]
27             id = int(id[2:]) + 1
28         else:
29             id = 1
30         return "AT" + str(id)
31
32
33     def insert(self,album_dto):
34         cursor = self.con.cursor()
35         cursor.execute("insert into album values(%s, %s, %s, %s, %s)", (album_dto.albumID,
36 album_dto.title, album_dto.artistID, album_dto.genre, album_dto.releasedate)) "releasedate": Unknown word.
37         self.con.commit()
38         cursor.close()
39
40     def update(self,album_dto):
41         cursor = self.con.cursor()
42         cursor.execute("update album set title = %s, artistID = %s, genre = %s, releasedate = %s where albumID = %s",
43 (album_dto.title, album_dto.artistID, album_dto.genre, album_dto.releasedate,album_dto.albumID))
44         self.con.commit()
45         cursor.close()
46
47     def getTracksFromAlbumID(self, albumID):
48         cursor = self.con.cursor()
49         cursor.execute("select * from track where albumID = %s", (albumID))
50         records = cursor.fetchall()
51         cursor.close()
52         return records
```

Hình 9: Code cho class AlbumDAL gồm các phương thức đọc, thêm, sửa

-Giải thích:

- Phương thức **getAllData(self)** trả về danh sách tất cả các album trong cơ sở dữ liệu.
 - + Tương tự như PlaylistDAL chỉ khác là trả về danh sách album.
- Phương thức **generateAlbumID(self, albumID)** tạo một ID duy nhất cho album mới.
 - +Chi tiết hơn:
 - Đầu tiên, chúng ta thiết lập kết nối với cơ sở dữ liệu.
 - Sau đó, chúng ta yêu cầu cơ sở dữ liệu trả về ID của album được thêm gần đây nhất. Chúng ta làm điều này bằng cách chạy một lệnh yêu cầu "cho tôi albumID từ bảng album, nhưng hãy đảm bảo rằng bạn đưa cho tôi cái cuối cùng bạn có".
 - Bây giờ, có hai khả năng: hoặc chúng ta nhận được một ID trả về, hoặc không. Nếu chúng ta nhận được một ID, điều đó có nghĩa là đã có một số album trong cơ sở dữ liệu. Vì vậy, chúng ta lấy ID đó, loại bỏ hai ký tự đầu tiên (giả sử là 'AT'), chuyển phần còn lại thành một số và cộng thêm một vào nó. Điều này đảm bảo rằng mỗi album có một ID duy nhất.
 - Nếu chúng ta không nhận được ID trả về, điều đó có nghĩa là chưa có album nào trong cơ sở dữ liệu. Vì vậy, chúng ta đơn giản là đặt ID cho album mới là 1.
 - Cuối cùng, chúng ta thêm 'AT' vào đầu ID (để chỉ ra rằng đó là Album ID), và trả về nó. Đây sẽ là ID duy nhất cho album mới.
- Phương thức **insert(self, album_dto)** tạo một album mới trong cơ sở dữ liệu.
 - +Tương tự như hàm insert của PlaylistDAL chỉ khác là thêm mới album.
- Phương thức **update(album_dto)** sửa thông tin của một album.
 - +Tương tự như hàm update của PlaylistDAL chỉ khác là sửa album.
- Phương thức **getTracksFromAlbumID(self, albumID)** lấy ra danh sách các bài hát theo albumID.
 - +Tương tự như hàm getDataPlaylistFromUserId của PlaylistDAL chỉ khác là lấy ra danh sách bài hát theo albumID.

```
ArtistDAL.py
DAL > ArtistDAL.py > ArtistDAL > update
...
1 import os
2 import sys
3 sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
4
5 from DAL.ConnectDB import ConnectSQL
6 from DTO.ArtistDTO import ArtistDTO
7
8 ...
9
10 class ArtistDAL:
11
12     con = ConnectSQL.connect_mysql()
13
14     def getAllData(self):
15         global con
16         cursor = ArtistDAL.con.cursor()
17         cursor.execute("select * from artist")
18         records = cursor.fetchall()
19         cursor.close()
20         return records
21
22     def generateArtistID(self):
23         cursor = self.con.cursor()
24         cursor.execute("select artistID from artist order by artistID desc limit 1")
25         artist_id = cursor.fetchone()
26         if artist_id :
27             id = artist_id[0]
28             id = int(id[2:]) + 1
29         else:
30             id = 1
31         return "AT" + str(id)
32
33     def insert(self,artist_dto):
34         cursor = self.con.cursor()
35         cursor.execute("insert into artist values(%s, %s, %s)", (artist_dto.artistID, artist_dto.name, artist_dto.genre))
36         self.con.commit()
37         cursor.close()
38
39     def update(self, artist_dto):
40         cursor = self.con.cursor()
41         cursor.execute("update artist set name = %s, genre = %s where artistID = %s", (artist_dto.name,
42         artist_dto.genre, artist_dto.artistID))
43         self.con.commit()
44         cursor.close()
45
46     def getTracksFromArtistID(self, artistID):
47         cursor = self.con.cursor()
48         cursor.execute("select * from track where artistID = %s", (artistID))
49         records = cursor.fetchall()
50         cursor.close()
51         return records
```

Hình 10: Code cho class ArtistDAL gồm các phương thức đọc, thêm, sửa

-Giải thích:

- Phương thức **getAllData(self)** trả về danh sách tất cả các nghệ sĩ trong cơ sở dữ liệu.
- Phương thức **generateArtistID(self)** tạo một ID duy nhất cho nghệ sĩ mới.
- Phương thức **insert(self, artist_dto)** thêm một nghệ sĩ mới vào cơ sở dữ liệu.
- Phương thức **update(artist_dto)** sửa thông tin của một nghệ sĩ.
- Phương thức **getTracksFromArtistID(self, artistID)** lấy ra danh sách các bài hát theo artistID.

```
L > PLDetailDAL.py > PLDetailDAL > deleteTrackInPlaylist
...
1 import os
2 import sys
3 sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
4
5
6 from DAL.ConnectDB import ConnectSQL
7 from DTO.PLDetailDTO import PLDetailDTO
8
9 ...
10 class PLDetailDAL:
11     con = ConnectSQL.connect_mysql()
12
13     def insertTracktoPlaylist(self, pldetail_dto): "Trackto": Unknown word.
14         cursor = self.con.cursor()
15         cursor.execute("insert into detail_playlist values(%s, %s, %s)", (pldetail_dto.playlistID, pldetail_dto.userID,
16                                                                           pldetail_dto.trackID)) "pldetail": Unknown w
17         self.con.commit()
18         cursor.close()
19
20     def deleteTrackInPlaylist(self, trackID):
21         cursor = self.con.cursor()
22         cursor.execute("delete from detail_playlist where tracktrackID = %s", (trackID,)) "tracktrack": Unknown word.
23         count = int(cursor.rowcount)
24         self.con.commit()
25         cursor.close()
26
27         if count > 0:
28             return True
29         return False
```

Hình 11: Code cho class PLDetailDAL gồm 2 phương thức thêm và xóa

-Giải thích:

- Phương thức **insertTracktoPlaylist(self, pldetail_dto)** thêm một bài hát vào playlist.
- Phương thức **deleteTrackInPlaylist(playlistID, trackID)** xóa một bài hát khỏi playlist theo trackID.

```
TrackDAL.py
DAL > TrackDAL.py > TrackDAL > update
You, yesterday | 3 authors (Mackeddy and others)
1 import os
2 import sys
3 sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
4
5
6 from DAL.ConnectDB import ConnectSQL
7 from DTO.TrackDTO import TrackDTO
8
9 You, yesterday | 3 authors (Mackeddy and others)
10 class TrackDAL():
11     con = ConnectSQL.connect_mysql()
12
13     def getAllData(self):
14         global con
15         cursor = TrackDAL.con.cursor()
16         cursor.execute("select * from track")
17         records = cursor.fetchall()
18         cursor.close()
19         return records
20
21     def generateTrackID(self):
22         cursor = self.con.cursor()
23         cursor.execute("select trackID from track order by trackID desc limit 1")
24         track_id = cursor.fetchone()
25         if track_id:
26             id = track_id[0]
27             id = int(id[1:]) + 1
28         else:
29             id = 1
30         return "T" + str(id)
31
32     def insert(self, track_dto):
33         try:
34             cursor = self.con.cursor()
35             if track_dto.albumID == "None":
36                 track_dto.albumID = None
37             cursor.execute("insert into track values(%s, %s, %s, %s, %s, %s)",
38 (track_dto.trackID, track_dto.title, track_dto.artistID, track_dto.albumID, track_dto.duration, track_dto.releasedate))
39             self.con.commit()
40             cursor.close()
41         except Exception as e:
42             print("DAL TRACK:", str(e))
43
44     #Chua xong
45     def delete(self, trackID):
46         cursor = self.con.cursor()
47         cursor.execute("delete from track where trackID = %s", (trackID,))
48         count = int(cursor.rowcount)
49         self.con.commit()
50         cursor.close()
51
52         if count > 0:
53             print("Xoa thanh cong")
54         else:
55             print("ma khong ton tai")
56
57     def update(self, track_dto):
58         cursor = self.con.cursor()
59         cursor.execute("update track set title = %s, artistID = %s, albumID = %s, duration = %s, realeasedate = %s where trackID = %s",
60 (track_dto.title, track_dto.artistID, track_dto.albumID, track_dto.duration, track_dto.releasedate, track_dto.trackID))
61         self.con.commit()
62         cursor.close()
63
64 trackdal = TrackDAL() "trackdal": Unknown word.
65 #trackdal.generateTrackID() "trackdal": Unknown word.
```

Hình 12: Code cho class TrackDAL gồm các phương thức CRUD (Create, Read, Update, Delete)



-Giải thích:

- Phương thức **getAllData(self)** trả về danh sách tất cả các bài hát trong cơ sở dữ liệu.
- Phương thức **generateTrackID(self)** tạo một ID duy nhất cho bài hát mới.
- Phương thức **insert(self, track_dto)** thêm một bài hát mới vào cơ sở dữ liệu.
- Phương thức **update(track_dto)** sửa thông tin của một bài hát.
- Phương thức **deleteTrack(trackID)** xóa một bài hát dựa vào id.



```
UserDAL.py M
DAL > UserDAL.py > ...
...
1 import os
2 import sys
3 sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
4 from DAL.ConnectDB import ConnectSQL
5 from DTO.UserDTO import UserDTO
6 ...
7
8 class UserDAL():
9
10     con = ConnectSQL.connect_mysql()
11
12     def getAllData(self):
13         global con
14         cursor = UserDAL.con.cursor()
15         cursor.execute("select * from user")
16         records = cursor.fetchall()
17         cursor.close()
18         return records
19
20     def generateUserID(self):
21         cursor = self.con.cursor()
22         cursor.execute("select userID from user order by userID desc limit 1")
23         user_id = cursor.fetchone()
24         if user_id :
25             id = user_id[0] + 1
26         else:
27             id = 1
28         return id
29
30     def insert(self, user_dto):
31         cursor = self.con.cursor()
32         cursor.execute("insert into user values(%s, %s, %s, %s)", (user_dto.userID, user_dto.username,
33                                                                     user_dto.email, user_dto.password))
34         self.con.commit()
35         cursor.close()
36
37     def checkUsernameAndPass(self, username, password):
38         cursor = self.con.cursor()
39         cursor.execute("select * from user where username = %s and password = %s", (username, password))
40         records = cursor.fetchall()
41         self.con.commit()
42         cursor.close()
43         if records:
44             return True
45         else:
46             return False
47
48     def update(self, user_dto):
49         cursor = self.con.cursor()
50         cursor.execute("update user set username = %s, email = %s, password = %s where userID = %s", (user_dto.username,
51                                                                                                     user_dto.email, user_dto.password, user_dto.userID))
52         self.con.commit()
53         cursor.close()
54
55     def checkUsername(self, username):
56         cursor = self.con.cursor()
57         cursor.execute("select * from user where username = %s", (username,))
58         records = cursor.fetchall()
59         self.con.commit()
60         cursor.close()
61         if records:
62             return True
63         else:
64             return False
65
66     def resetPassWord(self, username, password):
67         cursor = self.con.cursor()
68         cursor.execute("update user set password = %s where username = %s", (password, username))
69         self.con.commit()
70         cursor.close()
```

Hình 13: Code cho class UserDAL gồm các phương thức CRUD (Create, Read, Update, Delete)



-Giải thích:

- Phương thức **getAllData(self)** trả về danh sách tất cả các user trong cơ sở dữ liệu.
- Phương thức **generateUserID(self)** tạo một ID duy nhất cho user mới.
- Phương thức **checkUsernameAndPass(self, user_dto)** kiểm tra username và password của user.
- Phương thức **insert(self, user_dto)** thêm một user mới vào cơ sở dữ liệu.
- Phương thức **update(self, user_dto)** sửa thông tin của một user.
- Phương thức **checkUsername(self, userID)** kiểm tra username của user.
- Phương thức **resetPassword(self, username, password)** reset password của user.



Tài liệu

- [1] CVX Introduction “**link:** <http://cvxr.com/cvx/doc/intro.html>”, *What is CVX*, lần truy cập cuối: 15/04/2017.