

BỘ NÔNG NGHIỆP VÀ PHÁT TRIỂN NÔNG THÔN  
TRƯỜNG ĐẠI HỌC THỦY LỢI



# BÀI TẬP LỚN

HỌC PHẦN: PHÁT TRIỂN ỨNG DỤNG CHO CÁC THIẾT BỊ  
DI ĐỘNG

ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG NGHE NHẠC (ANDROID  
STUDIO)

Mã Sinh Viên	Họ và Tên	Ngày Sinh	Lớp
2051063968	Đặng Thành Nam	6/12/2002	62TH1
2051060508	Nguyễn Minh Hiếu	24/8/2002	62TH1
2051063495	Nguyễn Đăng Đông	5/5/2002	62TH1

Hà Nội, năm 2024

BỘ NÔNG NGHIỆP VÀ PHÁT TRIỂN NÔNG THÔN  
TRƯỜNG ĐẠI HỌC THỦY LỢI



# BÀI TẬP LỚN

HỌC PHẦN: PHÁT TRIỂN ỨNG DỤNG CHO CÁC THIẾT BỊ  
DI ĐỘNG

ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG NGHE NHẠC (ANDROID  
STUDIO)

Mã Sinh Viên	Họ và Tên	Ngày Sinh	Điểm	
			Bằng Số	Bằng Chữ
2051063968	Đặng Thành Nam	6/12/2002		
2051060508	Nguyễn Minh Hiếu	24/8/2002		
2051063495	Nguyễn Đăng Đông	5/5/2002		

CÁN BỘ CHẤM THI 1

CÁN BỘ CHẤM THI 2

Hà Nội, năm 2024

## LỜI NÓI ĐẦU

Trong thời đại công nghệ phát triển nhanh chóng hiện nay, các ứng dụng di động đã trở thành một phần không thể thiếu trong cuộc sống hàng ngày, giúp người dùng dễ dàng truy cập và sử dụng các dịch vụ từ bất kỳ đâu. Đặc biệt, nhu cầu giải trí qua các ứng dụng nghe nhạc đang ngày càng gia tăng. Đề tài "Xây dựng ứng dụng nghe nhạc Musium" là một phần của học phần "Phát triển ứng dụng cho các thiết bị di động," do giảng viên Kiều Tuấn Dũng hướng dẫn. Đây là cơ hội để sinh viên tìm hiểu, nghiên cứu và thực hành các kỹ thuật lập trình trên nền tảng Android Studio, đồng thời phát triển kỹ năng thiết kế giao diện người dùng và xử lý dữ liệu âm nhạc một cách tối ưu.

Trong quá trình thực hiện, nhóm đã áp dụng các kiến thức cơ bản về lập trình di động, nghiên cứu các thư viện và công nghệ hỗ trợ xử lý âm thanh nhằm xây dựng một ứng dụng nghe nhạc đáp ứng nhu cầu của người dùng, với giao diện thân thiện và tính năng phong phú. Hy vọng rằng, ứng dụng Musium này sẽ mang lại trải nghiệm sử dụng dễ dàng và tiện ích cho người dùng.

## MỤC LỤC

### CHƯƠNG 1 : MÔ TẢ BÀI TOÁN

<b>1.1 Giới thiệu :</b>	<b>5</b>
<b>1.2 Chức năng chính :</b>	<b>5</b>
1. <i>Tìm kiếm và phát nhạc :</i>	5
2. <i>Danh sách phát (Playlist) :</i>	5
3. <i>Khám phá và gợi ý :</i>	5
4. <i>Nghe offline :</i>	5
5. <i>Chia sẻ âm nhạc :</i>	6
6. <i>Thông báo và cập nhật :</i>	6
7. <i>Giao diện người dùng thân thiện :</i>	6
8. <i>Tùy chỉnh trải nghiệm :</i>	6
9. <i>Tương tác với nghệ sĩ :</i>	6
10. <i>Chế độ nghe nhạc thông minh :</i>	6
<b>1.3 Yêu cầu phi chức năng :</b>	<b>6</b>
1. <i>Hiệu suất :</i>	6
2. <i>Khả năng mở rộng :</i>	6
3. <i>Tính bảo mật :</i>	6
4. <i>Tính khả dụng :</i>	7
5. <i>Độ tin cậy :</i>	7
6. <i>Tính thân thiện với người dùng :</i>	7
7. <i>Khả năng tương tác :</i>	7
8. <i>Bảo trì và hỗ trợ :</i>	7
9. <i>Khả năng kết nối :</i>	7
10. <i>Tương thích với thiết bị ngoại vi :</i>	7

## **CHƯƠNG 2 : PHÂN TÍCH YÊU CẦU VÀ THIẾT KẾ HỆ THỐNG**

<b>2.1 Phân tích yêu cầu :</b>	<b>8</b>
<b>1. Yêu cầu chức năng :</b>	<b>8</b>
<b>2. Yêu cầu phi chức năng :</b>	<b>9</b>
<b>2.2 Thiết kế lên ý tưởng bài toán :</b>	<b>10</b>
<b>2.3 Thiết kế hệ thống :</b>	<b>11</b>
<b>1. Quan hệ giữa các lớp trong Model :</b>	<b>17</b>
<b>2. Quan hệ giữa Model, View và Controller :</b>	<b>18</b>
<b>3. Luồng tương tác MVC :</b>	<b>18</b>
<b>2.4 Triển khai :</b>	<b>20</b>
<b>2.5 Vận hành và bảo trì :</b>	<b>20</b>

## **CHƯƠNG 3 : KẾT QUẢ THỰC HIỆN**

<b>3.1 Công nghệ đã sử dụng :</b>	<b>21</b>
<b>3.2 Tiến độ thực hiện :</b>	<b>21</b>
<b>3.3 Hình ảnh sản phẩm :</b>	<b>22</b>

## **KẾT LUẬN**

## **TÀI LIỆU THAM KHẢO**

## CHƯƠNG 1. MÔ TẢ BÀI TOÁN

### 1.1. Giới thiệu

Trong thời đại công nghệ số hiện nay, nhu cầu thưởng thức âm nhạc của người dùng ngày càng gia tăng. Với sự phát triển nhanh chóng của thiết bị di động, việc truy cập và trải nghiệm âm nhạc đã trở nên dễ dàng hơn bao giờ hết. Tuy nhiên, nhiều ứng dụng nghe nhạc hiện tại thường yêu cầu người dùng phải trả phí hoặc đăng ký dịch vụ, điều này hạn chế khả năng tiếp cận của một bộ phận lớn người dùng, đặc biệt là sinh viên và người có thu nhập thấp.

Để đáp ứng nhu cầu này, chúng tôi đặt ra bài toán phát triển một ứng dụng nghe nhạc miễn phí, nhằm mang đến cho người dùng một nền tảng phong phú, đa dạng về thể loại âm nhạc, cùng với trải nghiệm thân thiện và tiện ích. Ứng dụng không chỉ giúp người dùng thưởng thức âm nhạc yêu thích mà còn khuyến khích sự sáng tạo và khám phá những nghệ sĩ mới. Qua đó, bài toán của chúng tôi không chỉ dừng lại ở việc phát triển công nghệ, mà còn góp phần xây dựng một cộng đồng âm nhạc rộng lớn và đa dạng.

### 1.2. Chức năng chính

Ứng dụng Musium có các chức năng sau:

- Đăng nhập, đăng ký: Cho phép người dùng tạo một tài khoản riêng để có thể

#### *1. Tìm kiếm và phát nhạc*

- Cho phép người dùng tìm kiếm bài hát, album hoặc nghệ sĩ thông qua thanh tìm kiếm.
- Cung cấp danh sách kết quả với khả năng phát ngay lập tức.

#### *2. Danh sách phát (Playlist)*

- Người dùng có thể tạo và quản lý danh sách phát riêng.
- Cho phép thêm, xóa và sắp xếp các bài hát theo ý thích.

#### *3. Khám phá và gợi ý*

- Gợi ý bài hát, album và nghệ sĩ dựa trên sở thích và thói quen nghe nhạc của người dùng.
- Cung cấp danh sách nhạc mới phát hành và các bảng xếp hạng âm nhạc.

#### *4. Nghe offline*

- Cho phép người dùng tải xuống bài hát để nghe offline, giúp tiết kiệm dữ liệu di động.

### **5. Thông báo và cập nhật**

- Cung cấp thông báo về các bài hát mới, nghệ sĩ nổi bật, và sự kiện âm nhạc sắp tới.

### **6. Giao diện người dùng thân thiện**

- Thiết kế giao diện dễ sử dụng, hỗ trợ nhiều thiết bị và kích thước màn hình khác nhau.

### **7. Tùy chỉnh trải nghiệm**

- Cho phép người dùng tùy chỉnh giao diện và cài đặt nghe nhạc, như chất lượng âm thanh và chế độ phát lại.

### **8. Tương tác với nghệ sĩ**

- Tích hợp tính năng để người dùng theo dõi nghệ sĩ yêu thích và nhận thông báo về hoạt động của họ.

### **9. Chế độ nghe nhạc thông minh**

- Hỗ trợ chế độ phát ngẫu nhiên, lặp lại và chế độ phát theo thể loại.

Những chức năng này không chỉ mang lại trải nghiệm phong phú cho người dùng mà còn thúc đẩy sự kết nối và tương tác trong cộng đồng yêu nhạc.

## **1.3. Yêu cầu phi chức năng**

### **1. Hiệu suất**

- Ứng dụng cần tải nhanh, đảm bảo thời gian khởi động dưới 2 giây.
- Thời gian phát nhạc không bị gián đoạn, ngay cả khi chuyển đổi giữa các bài hát.

### **2. Khả năng mở rộng**

- Hệ thống phải có khả năng xử lý số lượng người dùng lớn mà không ảnh hưởng đến hiệu suất.

### **3. Tính bảo mật**

- Bảo vệ thông tin cá nhân của người dùng và dữ liệu nghe nhạc thông qua mã hóa.
- Đảm bảo an toàn khi chia sẻ nội dung qua mạng xã hội.

#### **4. Tính khả dụng**

- Ứng dụng cần hoạt động trên nhiều hệ điều hành di động (iOS, Android) và các phiên bản khác nhau.
- Cung cấp hỗ trợ cho các thiết bị với độ phân giải màn hình khác nhau.

#### **5. Độ tin cậy**

- Đảm bảo thời gian hoạt động cao, với tỉ lệ uptime tối thiểu 99%.
- Cung cấp cơ chế phục hồi dữ liệu khi có sự cố xảy ra.

#### **6. Tính thân thiện với người dùng**

- Giao diện trực quan, dễ sử dụng cho người dùng ở mọi lứa tuổi.
- Hỗ trợ nhiều ngôn ngữ khác nhau để phục vụ người dùng đa dạng.

#### **7. Khả năng tương tác**

- Tương tác mượt mà giữa các chức năng mà không gây ra độ trễ cho người dùng.

#### **8. Bảo trì và hỗ trợ**

- Cung cấp hướng dẫn sử dụng và hỗ trợ khách hàng thông qua các kênh khác nhau (chat, email, diễn đàn).
- Đảm bảo dễ dàng cập nhật và bảo trì ứng dụng để sửa lỗi và cải tiến tính năng.

#### **9. Khả năng kết nối**

- Ứng dụng cần hoạt động hiệu quả với cả kết nối Wi-Fi và dữ liệu di động, bao gồm chế độ offline.

#### **10. Tương thích với thiết bị ngoại vi**

- Hỗ trợ kết nối với tai nghe, loa Bluetooth và các thiết bị âm thanh khác.

Những yêu cầu phi chức năng này sẽ giúp đảm bảo ứng dụng không chỉ hoạt động hiệu quả mà còn mang lại trải nghiệm tốt nhất cho người dùng.

## CHƯƠNG 2. PHÂN TÍCH YÊU CẦU VÀ THIẾT KẾ HỆ THỐNG

### 2.1. Phân tích yêu cầu:

#### \*Xác định người dùng:

- Sinh viên và học sinh:** Thường có ngân sách hạn chế và tìm kiếm các nguồn giải trí miễn phí.
- Người yêu thích âm nhạc:** Những người muốn khám phá nhiều thể loại và nghệ sĩ mới mà không phải trả phí.
- Người dùng di động:** Những người thường xuyên sử dụng điện thoại để nghe nhạc khi di chuyển.

#### \*Phân tích yêu cầu:

##### 1. Yêu cầu chức năng

###### a. Tìm kiếm và phát nhạc

- Yêu cầu:** Người dùng có thể tìm kiếm bài hát, nghệ sĩ, hoặc album thông qua thanh tìm kiếm.
- Phân tích:** Cần thiết lập một thuật toán tìm kiếm hiệu quả, cho phép người dùng nhập từ khóa và nhận được kết quả ngay lập tức. Hệ thống cũng cần hỗ trợ tìm kiếm theo thể loại.

###### b. Danh sách phát (Playlist)

- Yêu cầu:** Người dùng có thể tạo, chỉnh sửa và quản lý danh sách phát.
- Phân tích:** Cần có giao diện dễ sử dụng cho phép người dùng thêm, xóa và sắp xếp bài hát. Ngoài ra, có thể cung cấp danh sách phát mẫu để người dùng tham khảo.

###### c. Khám phá và gợi ý

- Yêu cầu:** Ứng dụng cung cấp gợi ý bài hát và nghệ sĩ dựa trên thói quen nghe nhạc.
- Phân tích:** Cần phát triển thuật toán học máy để cá nhân hóa trải nghiệm nghe nhạc cho người dùng, có thể dựa trên lịch sử nghe nhạc và các bài hát yêu thích.

###### d. Nghe offline

- Yêu cầu:** Người dùng có thể tải xuống bài hát để nghe khi không có kết nối internet.
- Phân tích:** Cần xây dựng một hệ thống quản lý tải xuống và lưu trữ, đảm bảo người dùng có thể dễ dàng truy cập các bài hát đã tải.

###### e. Chia sẻ âm nhạc

- **Yêu cầu:** Người dùng có thể chia sẻ bài hát hoặc danh sách phát qua mạng xã hội.
- **Phân tích:** Tích hợp API từ các mạng xã hội để người dùng dễ dàng chia sẻ nội dung mà không gặp khó khăn.

## 2. *Yêu cầu phi chức năng*

### a. **Hiệu suất**

- **Yêu cầu:** Ứng dụng cần tải nhanh và không bị gián đoạn trong quá trình phát nhạc.
- **Phân tích:** Cần tối ưu hóa mã nguồn và kiến trúc hệ thống để đảm bảo thời gian khởi động dưới 2 giây và phát nhạc mượt mà.

### b. **Tính bảo mật**

- **Yêu cầu:** Bảo vệ thông tin cá nhân và dữ liệu người dùng.
- **Phân tích:** Cần sử dụng mã hóa và các biện pháp bảo mật tốt để bảo vệ dữ liệu người dùng, đặc biệt khi chia sẻ thông tin qua mạng.

### c. **Độ tin cậy**

- **Yêu cầu:** Ứng dụng cần có thời gian hoạt động cao và khôi phục nhanh khi gặp sự cố.
- **Phân tích:** Thiết lập cơ chế kiểm tra lỗi và phục hồi dữ liệu để đảm bảo trải nghiệm liền mạch cho người dùng.

### d. **Tính thân thiện với người dùng**

- **Yêu cầu:** Giao diện dễ sử dụng, hỗ trợ nhiều ngôn ngữ.
- **Phân tích:** Cần thiết kế giao diện người dùng trực quan và cung cấp hướng dẫn chi tiết cho những người dùng mới.

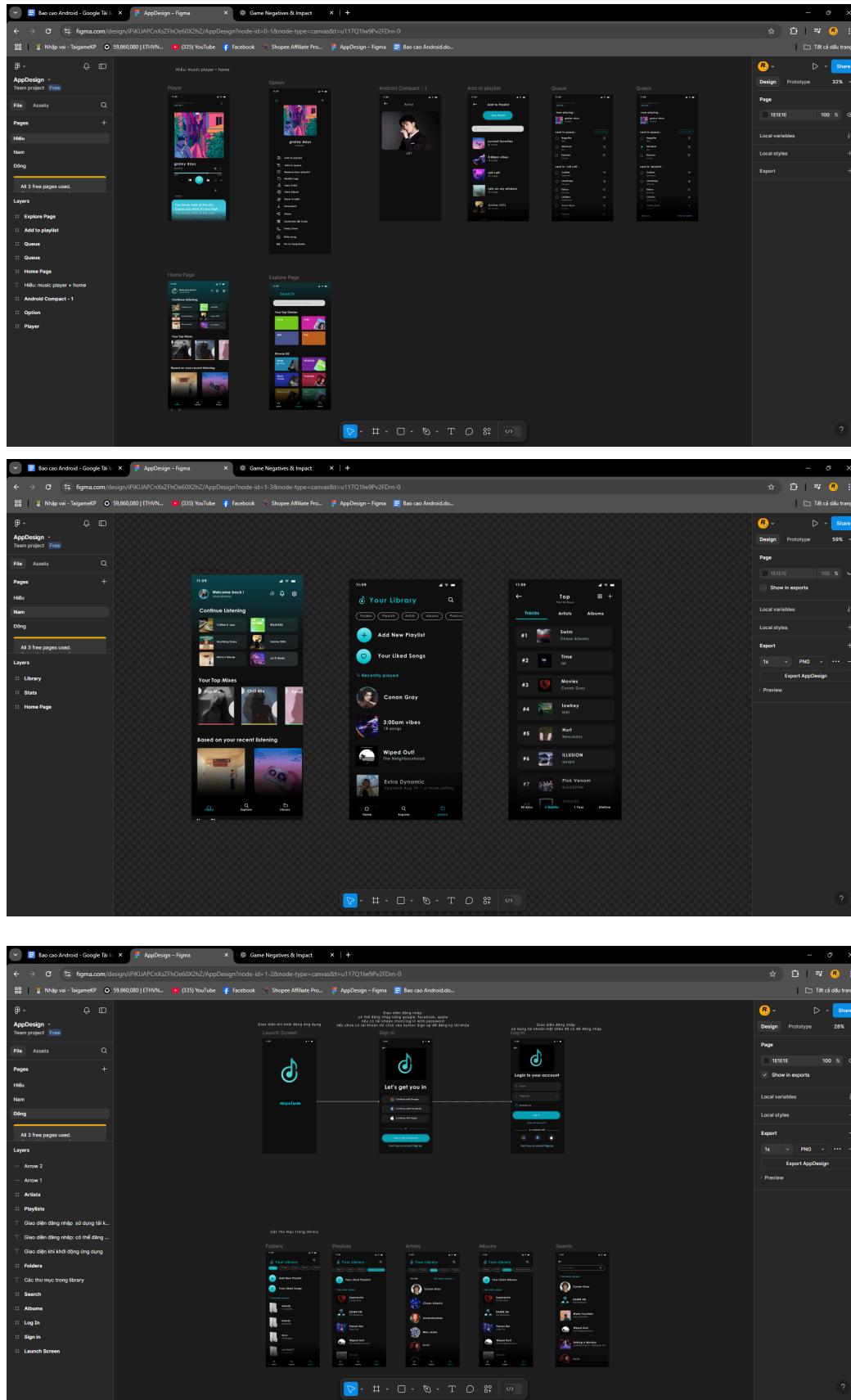
### e. **Khả năng mở rộng**

- **Yêu cầu:** Hệ thống cần có khả năng mở rộng để phục vụ số lượng người dùng lớn.
- **Phân tích:** Cần dự kiến trước khả năng gia tăng người dùng và thiết kế kiến trúc phù hợp để xử lý lượng truy cập cao mà không làm giảm hiệu suất.

## Tóm tắt

Việc phân tích yêu cầu dựa trên mô tả bài toán giúp xác định rõ ràng các tính năng cần thiết, cũng như các yếu tố kỹ thuật quan trọng để phát triển ứng dụng nghe nhạc miễn phí hiệu quả và đáp ứng nhu cầu của người dùng.

## 2.2. Thiết kế lên ý tưởng bài toán:



### 2.3. Thiết kế hệ thống:

#### Biểu đồ lớp (Class Diagram)

##### 1. Lớp User

- Thuộc tính:
  - userId: char(10)
  - username: String
  - password: String
  - FullName: String
  - playlistsID: List
- Phương thức:
  - SignUp
  - login()
  - LogOut
  - getPlaylist()

##### 2. Lớp Song

- Thuộc tính:
  - SongId: char(10)
  - SongName: String
  - ArtistID: char(10)
  - AlbumID: char(10)
  - GenreID: char(10)
  - AlbumID: char(10)
- Phương thức:
  - playSong: MediaPlayer
  - addToPlaylist
  - get Artist
  - get Album
  - get Genre

##### 3. Lớp Artist

- Thuộc tính:
  - ArtistId: char(10)
  - ArtistName: String
  - Description: String
- Phương thức:
  - getAlbums()
  - gétong()

#### 4. Lớp **Album**

- **Thuộc tính:**
  - `albumId: char(10)`
  - `AlbumName: String`
  - `releaseDate: Date`
  - `SongID: List`
  - `ArtistID: char(10)`
- **Phương thức:**
  - `getSong`
  - `getArtist`

#### 5. Lớp **Playlist**

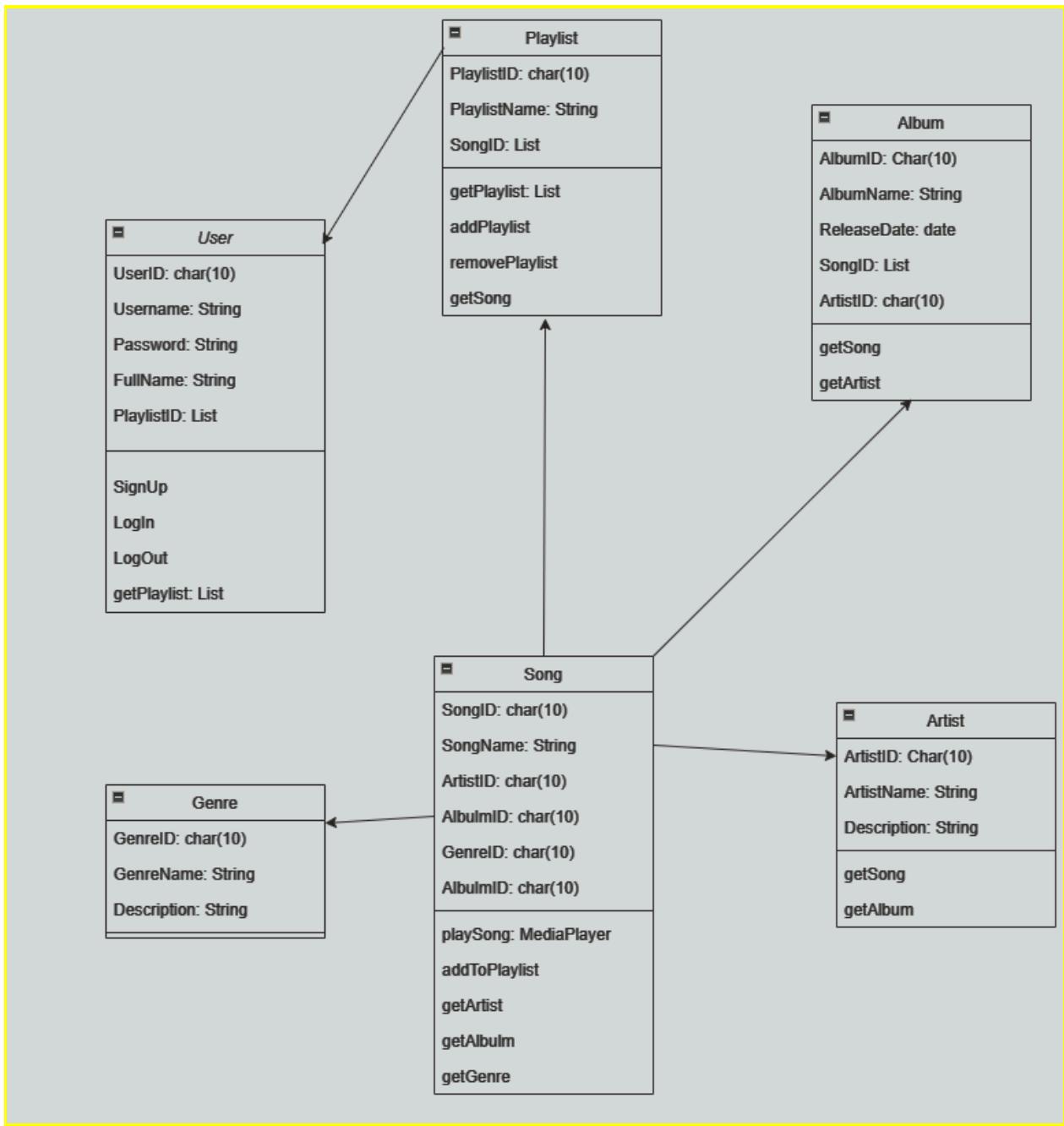
- **Thuộc tính:**
  - `PlaylistId: char(10)`
  - `PlaylistName: String`
  - `SongID: List<Music>`
- **Phương thức:**
  - `getPlaylist: List`
  - `addPlaylist`
  - `removePlaylist`
  - `getSong`

#### 6. Lớp **Genre**

- **Thuộc tính:**
  - `GenreID: char(10)`
  - `GenreName: String`
  - `Description: String`

#### Quan hệ giữa các lớp:

- **User** có thể có nhiều **Playlist**.
- **Playlist** chứa nhiều **Song**.
- **Music** thuộc về một **Artist** và một **Album**.
- **Album** chứa nhiều **Music**.
- **Song** thuộc nhiều **Genre**



### Biểu đồ lớp cho MVC:

- Mô tả kiến trúc của ứng dụng, bao gồm các lớp trong Model, View và Controller.
- Thể hiện mối quan hệ giữa các lớp trong MVC và cách chúng tương tác với nhau.

## Thiết kế kiến trúc (MVC):

- **Model:**

**Model** đại diện cho dữ liệu và logic nghiệp vụ. Trong trường hợp này, Model tương ứng với các cấu trúc dữ liệu và phương thức từ sơ đồ lớp.

- **User (Người dùng):** Xử lý dữ liệu và các hành động của người dùng như **SignUp** (Đăng ký), **LogIn** (Đăng nhập), **Logout** (Đăng xuất), và quản lý các playlist.
  - Thuộc tính: **UserID**, **Username** (Tên người dùng), **Password** (Mật khẩu), **FullName** (Tên đầy đủ), **PlaylistID** (Danh sách playlist).
  - Phương thức: **SignUp()**, **LogIn()**, **Logout()**, **getPlaylist()**.
- **Playlist (Danh sách phát nhạc):** Quản lý danh sách phát nhạc và các bài hát trong đó.
  - Thuộc tính: **PlaylistID**, **PlaylistName** (Tên playlist), **SongID** (Danh sách các bài hát).
  - Phương thức: **getPlaylist()**, **addPlaylist()** (Thêm playlist), **removePlaylist()** (Xóa playlist), **getSong()** (Lấy bài hát).
- **Song (Bài hát):** Chứa thông tin bài hát và các tương tác như thêm vào danh sách phát hoặc phát bài hát.
  - Thuộc tính: **SongID**, **SongName** (Tên bài hát), **ArtistID** (Mã nghệ sĩ), **AlbumID** (Mã album), **GenreID** (Mã thể loại).
  - Phương thức: **playSong()** (Phát bài hát), **addToPlaylist()** (Thêm vào danh sách phát), **getArtist()** (Lấy thông tin nghệ sĩ), **getAlbum()** (Lấy thông tin album).
- **Album (Album nhạc):** Quản lý thông tin về các album nhạc.
  - Thuộc tính: **AlbumID**, **AlbumName** (Tên album), **ReleaseDate** (Ngày phát hành), **SongID** (Danh sách các bài hát), **ArtistID** (Mã nghệ sĩ).
  - Phương thức: **getSong()** (Lấy danh sách bài hát trong album), **getArtist()** (Lấy thông tin nghệ sĩ của album).
- **Artist (Nghệ sĩ):** Chứa thông tin về các nghệ sĩ.
  - Thuộc tính: **ArtistID**, **ArtistName** (Tên nghệ sĩ), **Description** (Mô tả về nghệ sĩ).
  - Phương thức: **getSong()** (Lấy bài hát của nghệ sĩ), **getAlbum()** (Lấy album của nghệ sĩ).

- **Genre (Thể loại):** Lưu trữ thông tin về các thể loại âm nhạc.
  - Thuộc tính: **GenreID**, **GenreName** (Tên thể loại), **Description** (Mô tả thể loại).

- **View:**

Lớp **View** chịu trách nhiệm về giao diện người dùng (UI), hiển thị dữ liệu từ Model và gửi các thao tác của người dùng đến Controller. Các màn hình giao diện cụ thể có thể được phát triển cho từng đối tượng:

- **User View:** Hiển thị thông tin người dùng, giao diện đăng ký, đăng nhập, và danh sách phát của người dùng.
  - Giao diện đăng ký và đăng nhập.
  - Hiển thị danh sách phát và bài hát của người dùng.
- **Playlist View:** Hiển thị các playlist của người dùng, thêm và xóa bài hát khỏi playlist.
  - Hiển thị danh sách phát và chi tiết các bài hát bên trong.
  - Giao diện thêm bài hát vào playlist hoặc xóa bài hát.
- **Song View:** Hiển thị thông tin bài hát, cho phép người dùng phát bài hát.
  - Giao diện để phát bài hát, thêm bài hát vào playlist.
- **Album View:** Hiển thị các album nhạc với thông tin chi tiết và danh sách các bài hát thuộc album.
  - Giao diện hiển thị các album của nghệ sĩ, và chi tiết bài hát trong album.
- **Artist View:** Hiển thị thông tin về nghệ sĩ, bao gồm album và các bài hát của họ.
  - Giao diện để duyệt danh sách bài hát và album của nghệ sĩ.
  -

- **Controller:**

Lớp **Controller** chịu trách nhiệm xử lý các yêu cầu của người dùng, thực hiện các thao tác trên dữ liệu (Model), và trả kết quả về View. Mỗi controller sẽ tương tác với từng phần của hệ thống:

- **User Controller:** Quản lý việc đăng ký, đăng nhập, đăng xuất và quản lý playlist của người dùng.
  - Phương thức: `handleSignUp()`, `handleLogIn()`, `handleLogOut()`, `managePlaylist()`.
- **Playlist Controller:** Xử lý các thao tác liên quan đến playlist, như thêm hoặc xóa bài hát khỏi playlist.

- Phương thức: `handleAddToPlaylist()`,  
`handleRemoveFromPlaylist()`,  
`getPlaylistDetails()`.
  - **Song Controller:** Điều khiển việc phát nhạc, thêm bài hát vào playlist, hoặc lấy thông tin bài hát.
    - Phương thức: `handlePlaySong()`,  
`handleAddToPlaylist()`, `getSongDetails()`.
  - **Album Controller:** Xử lý việc lấy thông tin album và bài hát trong album.
    - Phương thức: `getAlbumDetails()`.
  - **Artist Controller:** Xử lý việc lấy thông tin nghệ sĩ, album và bài hát của nghệ sĩ.
    - Phương thức: `getArtistDetails()`.
- 

Tóm lại, kiến trúc MVC sẽ chia ứng dụng của bạn thành các phần Model, View và Controller rõ ràng, giúp quản lý dữ liệu, giao diện và tương tác người dùng một cách mạch lạc và dễ bảo trì.

### **Dựa trên kiến trúc MVC đã chọn, ta có thể xác định các lớp sau:**

Các lớp Controller sẽ điều khiển luồng dữ liệu giữa View và Model, xử lý các yêu cầu từ người dùng.

- **UserController:** Quản lý các hành động của người dùng như đăng ký, đăng nhập, và quản lý danh sách phát.
  - Phương thức:
    - `handleSignUp()`
    - `handleLogIn()`
    - `handleLogOut()`
    - `managePlaylist()`
- **PlaylistController:** Xử lý các thao tác thêm, xóa bài hát từ playlist, hoặc xem danh sách bài hát.
  - Phương thức:
    - `handleAddToPlaylist()`
    - `handleRemoveFromPlaylist()`
    - `getPlaylistDetails()`
- **SongController:** Quản lý phát nhạc, thêm bài hát vào playlist, và lấy thông tin chi tiết của bài hát.
  - Phương thức:

- handlePlaySong()
  - handleAddToPlaylist()
  - getSongDetails()
  - **AlbumController:** Xử lý việc lấy thông tin album và bài hát trong album.
    - Phương thức:
      - getAlbumDetails()
  - **ArtistController:** Xử lý các yêu cầu về nghệ sĩ, bao gồm thông tin album và bài hát.
    - Phương thức:
      - getArtistDetails()
- 

Tóm lại, với kiến trúc MVC này, các lớp sẽ được chia thành ba phần chính: **Model** để quản lý dữ liệu, **View** để quản lý giao diện người dùng, và **Controller** để điều khiển luồng dữ liệu và các thao tác của người dùng.

### 1. Quan hệ giữa các lớp trong Model:

Trong phần **Model**, các lớp có mối quan hệ trực tiếp với nhau thông qua các thuộc tính đại diện cho các liên kết giữa các đối tượng:

- **User - Playlist:**
  - Một User (người dùng) có thể có nhiều Playlist (danh sách phát). Mỗi người dùng sẽ có danh sách các playlist riêng thông qua thuộc tính **PlaylistID** (danh sách phát).
  - Mối quan hệ: 1 - N (một người dùng có nhiều danh sách phát).
- **Playlist - Song:**
  - Mỗi Playlist có thể chứa nhiều Song (bài hát). Mỗi playlist sẽ chứa các **SongID** đại diện cho các bài hát trong danh sách phát.
  - Mối quan hệ: 1 - N (một playlist có nhiều bài hát).
- **Song - Album:**
  - Một Song thuộc về một Album, thông qua thuộc tính **AlbumID**. Một album chứa nhiều bài hát.
  - Mối quan hệ: 1 - N (một album có nhiều bài hát, một bài hát chỉ thuộc về một album).
- **Song - Artist:**
  - Mỗi bài hát (Song) thuộc về một Artist (nghệ sĩ) thông qua thuộc tính **ArtistID**. Một nghệ sĩ có thể có nhiều bài hát.
  - Mối quan hệ: 1 - N (một nghệ sĩ có nhiều bài hát, một bài hát chỉ có một nghệ sĩ).
- **Song - Genre:**

- Mỗi bài hát thuộc về một **Genre** (thể loại), thông qua **GenreID**. Một thể loại có thể có nhiều bài hát.
- Mỗi quan hệ: 1 - N (một thể loại có nhiều bài hát, một bài hát thuộc một thể loại).
- **Album - Artist:**
  - Một **Album** thuộc về một **Artist** thông qua **ArtistID**. Một nghệ sĩ có thể có nhiều album.
  - Mỗi quan hệ: 1 - N (một nghệ sĩ có nhiều album, một album chỉ có một nghệ sĩ).

## 2. Quan hệ giữa Model, View và Controller:

- **View - Model:**
  - **View** chịu trách nhiệm hiển thị dữ liệu từ **Model**. Khi **Controller** yêu cầu **Model**, dữ liệu từ **Model** sẽ được truyền về và **View** sẽ hiển thị dữ liệu này cho người dùng. Ví dụ:
    - **UserView** hiển thị danh sách playlist của người dùng (lấy từ **Playlist Model**).
    - **PlaylistView** hiển thị các bài hát trong một playlist (lấy từ **Song Model**).
- **Controller - Model:**
  - **Controller** tương tác với **Model** để thực hiện các thao tác như thêm, xóa, cập nhật dữ liệu. **Controller** nhận yêu cầu từ **View** và điều khiển **Model** thực hiện các thay đổi cần thiết. Ví dụ:
    - **PlaylistController** có thể yêu cầu **Model** thêm một bài hát vào playlist.
    - **UserController** yêu cầu **Model** xác thực thông tin đăng nhập từ người dùng.
- **View - Controller:**
  - **View** gửi các hành động của người dùng đến **Controller**. Khi người dùng thực hiện các thao tác (như nhấn nút "Đăng nhập", "Thêm bài hát"), **View** sẽ gửi yêu cầu đến **Controller** xử lý. Ví dụ:
    - Khi người dùng đăng nhập, **UserView** sẽ gửi thông tin đến **UserController** để thực hiện xác thực.
    - Khi người dùng muốn thêm bài hát vào playlist, **PlaylistView** gửi yêu cầu đến **PlaylistController**.

## 3. Luồng tương tác MVC:

- **Người dùng** tương tác với **View** thông qua giao diện người dùng.
- **View** nhận yêu cầu từ người dùng và gửi tới **Controller**.
- **Controller** xử lý yêu cầu, tương tác với **Model** để thực hiện các thay đổi hoặc lấy dữ liệu.
- **Model** thực hiện các thao tác nghiệp vụ và gửi dữ liệu về **Controller**.

- **Controller** gửi dữ liệu tới **View** để hiển thị kết quả cho người dùng.

### Ví dụ cụ thể về tương tác giữa các lớp:

#### 1. Người dùng đăng nhập:

- Người dùng nhập thông tin vào **UserView** (Giao diện đăng nhập).
- **UserView** gửi thông tin đến **UserController**.
- **UserController** gọi **User Model** để kiểm tra thông tin người dùng.
- **User Model** xác nhận thông tin và gửi kết quả lại cho **UserController**.
- **UserController** truyền kết quả đăng nhập cho **UserView** để hiển thị thông báo thành công hoặc thất bại.

#### 2. Người dùng thêm bài hát vào playlist:

- Người dùng chọn bài hát và nhấn "Thêm vào playlist" trên **PlaylistView**.
- **PlaylistView** gửi yêu cầu đến **PlaylistController**.
- **PlaylistController** gọi **Song Model** để lấy thông tin bài hát và cập nhật vào **Playlist Model**.
- **PlaylistController** sau đó gửi kết quả về **PlaylistView** để hiển thị danh sách playlist đã cập nhật cho người dùng.

### Thiết kế cơ sở dữ liệu:

- Tạo một Google Sheet mới để lưu trữ dữ liệu.
- Xác định tên sheet và các cột tương ứng với các thuộc tính của công việc (ID, tiêu đề, mô tả, deadline, mức độ ưu tiên, trạng thái).

### Thiết kế giao diện:

- Hiển thị menu chính với các lựa chọn:
  1. Hiển thị danh sách ca sĩ
  2. Hiển thị các thể loại nhạc
  3. Hiển thị các album
  4. Tìm kiếm
  5. Thư viện
  6. Thoát

### 2.4. Triển khai:

- **Viết code:** Sử dụng Java để cài đặt các class trong mô hình MVC, đọc/ghi file, xử lý dữ liệu và hiển thị giao diện console.

- Xử lý xác thực để ứng dụng có quyền truy cập vào Google Sheet của bạn.
- Thay đổi code trong TaskController để sử dụng GoogleSheetHandler cho việc đọc/ghi dữ liệu.
- **Kiểm thử:** Viết các test case để kiểm tra các chức năng của ứng dụng, đảm bảo ứng dụng hoạt động đúng theo yêu cầu.

## 2.5. Vận hành và bảo trì:

- Cài đặt và triển khai:
  - Hướng dẫn người dùng cách chạy ứng dụng từ console (ví dụ: java -jar Musium.jar).
  - Ngoài hướng dẫn chạy ứng dụng, cần hướng dẫn người dùng cách tạo Google Sheet và chia sẻ quyền truy cập cho ứng dụng.
  - Có thể cần cài đặt thêm các thư viện cần thiết cho việc kết nối với Google Sheets.
- Bảo trì: Sửa lỗi phát sinh, cập nhật chức năng mới (nếu có) và cải thiện hiệu năng của ứng dụng.

## CHƯƠNG 3. KẾT QUẢ THỰC HIỆN

### 3.1. Công nghệ đã sử dụng

- Ngôn ngữ lập trình: Java
- Công cụ: Android Studio
- Thư viện: firebase

### 3.2. Tiến độ thực hiện

Link github tới dự án: [https://github.com/nemngonnnhi/CSE441\\_PROJECT22](https://github.com/nemngonnnhi/CSE441_PROJECT22)

#### Hướng dẫn các bước đã thực hiện:

##### B1. Tạo dự án mới:

- Mở Android Studio
- Chọn "New Project".
- Chọn Empty Views Activity
- Chọn "Java" làm ngôn ngữ lập trình.
- Chọn JDK phù hợp (ví dụ: JDK 11 hoặc mới hơn).
- Nhập tên dự án .
- Chọn vị trí lưu trữ dự án.
- Nhấn "Finish" để tạo dự án.

##### B2. Tạo các lớp:

- Trong mỗi package, click chuột phải và chọn "New" -> "Java Class" để tạo các lớp tương ứng.
- Cài đặt các thuộc tính và phương thức cho từng lớp dựa trên thiết kế đã phân tích.

##### B4. Cài đặt thư viện Firebase:

- Mở file pom.xml (nếu bạn đang sử dụng Maven) hoặc build.gradle (nếu bạn đang sử dụng Gradle).
- Thêm các dependency cần thiết cho Firebase. Ví dụ:  

```
implementation 'com.google.firebaseio:firebase-admin:9.0.0'
```
- Lưu file build.gradle. IntelliJ IDEA sẽ tự động tải về và thêm thư viện vào dự án của bạn..

##### B5. Viết code:

- Bắt đầu viết code cho từng lớp, thực hiện các chức năng của ứng dụng:
  - **Task:** cài đặt các thuộc tính và phương thức cơ bản.
  - **TaskList:** cài đặt các phương thức để quản lý danh sách công việc.

- **TaskView:** cài đặt các phương thức để hiển thị giao diện console và tương tác với người dùng.
- **TaskController:** cài đặt logic xử lý yêu cầu của người dùng, điều phối các lớp khác để hoàn thành các chức năng.

#### **B6. Xử lý xác thực :**

- Tạo tài khoản và đăng nhập vào Firebass.
- Tạo một dự án mới hoặc chọn dự án hiện có mà bạn muốn sử dụng với Firebase.
- Sử dụng file JSON để xác thực với Firebase
- Cài đặt mã xử lý xác thực
- Gọi hàm `initializeFirebase`

#### **B7. Chạy và kiểm thử:**

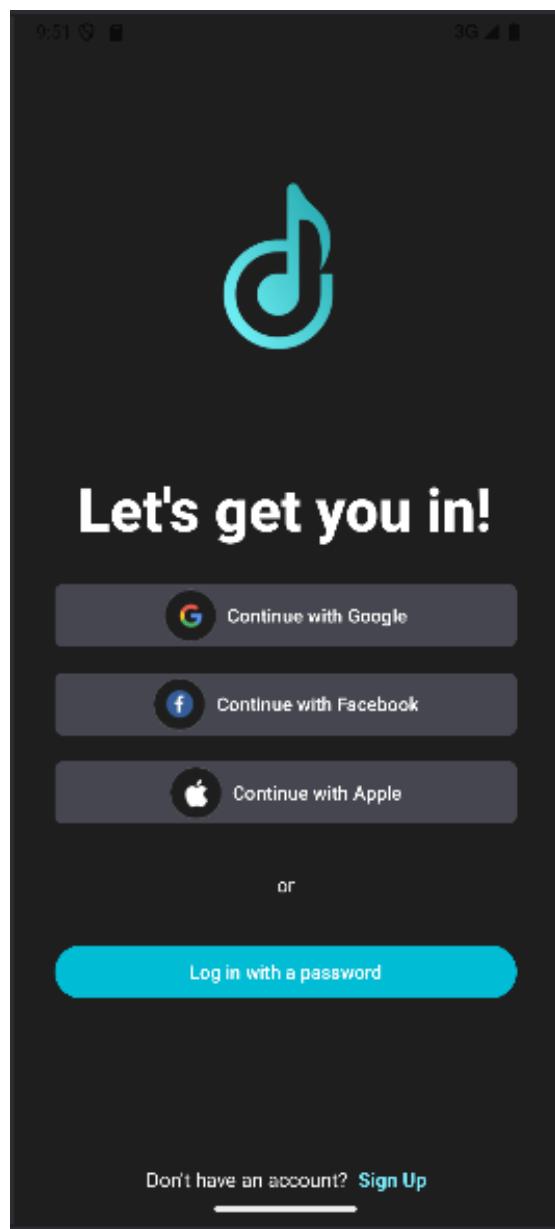
- Chạy ứng dụng từ máy ảo bằng cách click chuột phải vào lớp Main (hoặc lớp chứa phương thức main) và chọn "**Run 'Main.main()'**".
- Kiểm tra các chức năng của ứng dụng, sửa lỗi và hoàn thiện code.

#### **B8. Triển khai (tùy chọn):**

- Nếu muốn đóng gói ứng dụng thành file JAR để dễ dàng chia sẻ và chạy trên các máy khác, bạn có thể sử dụng chức năng "Build Artifacts" của Android Studio

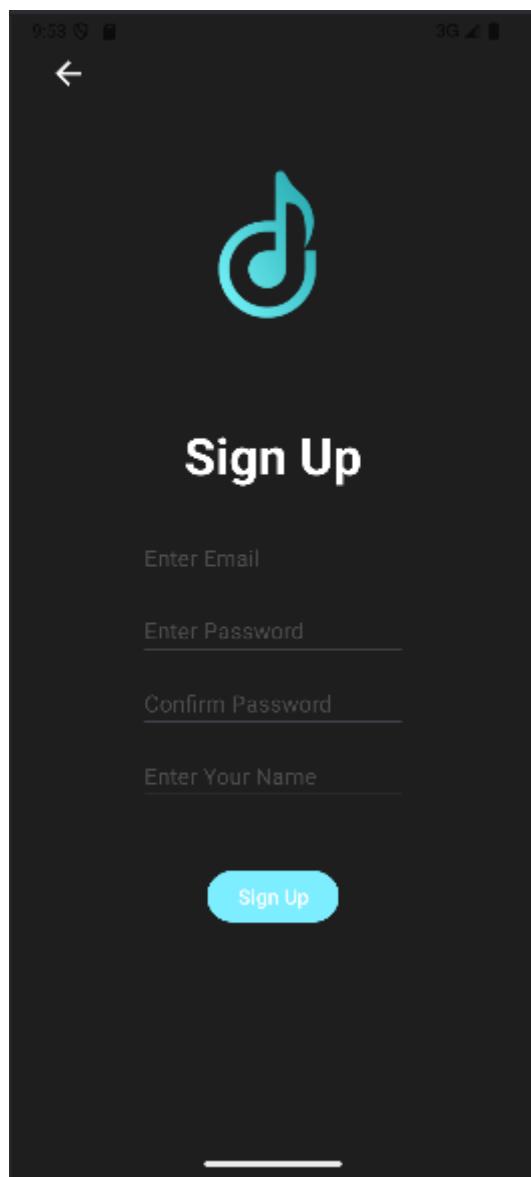
### **3.3. Hình ảnh sản phẩm**

1. Giao diện bắt đầu: Người dùng có thể lựa chọn đăng nhập bằng tài khoản sẵn có hoặc đăng ký nếu chưa có tài khoản.

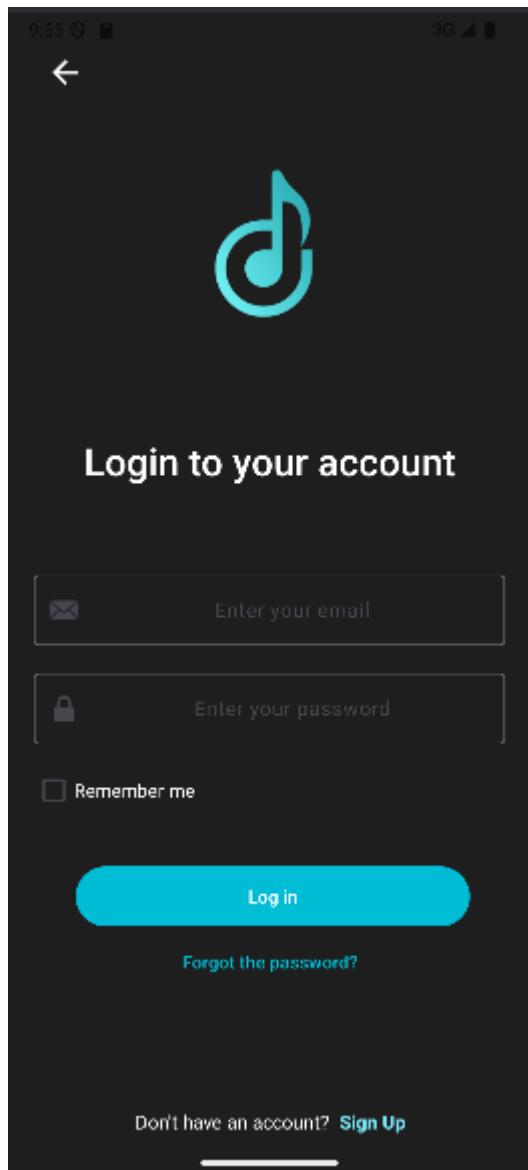


## 2. Giao diện đăng ký:

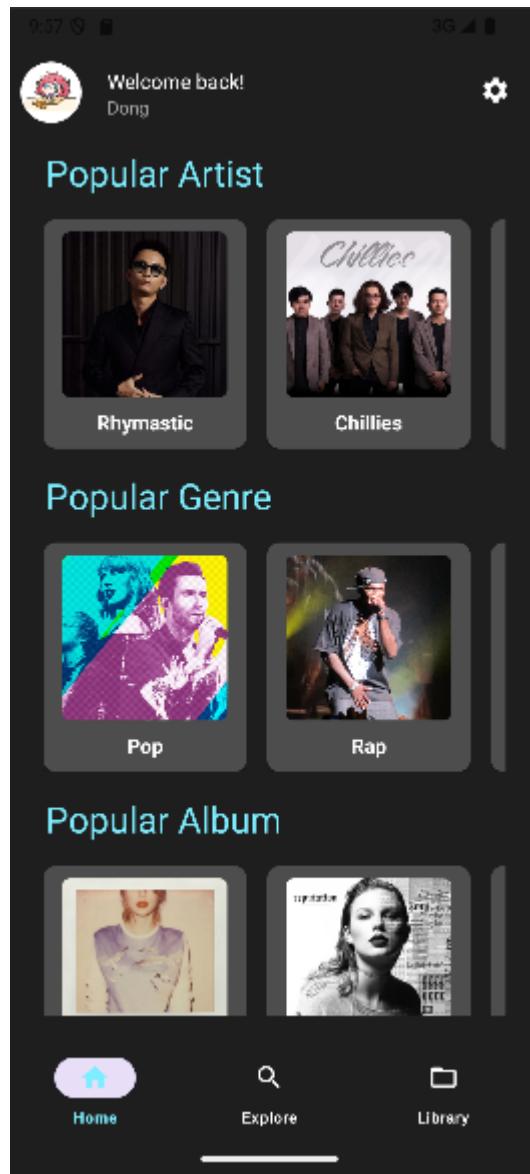
Người dùng đăng ký bằng email, password và Tên.



### 3. Giao diện đăng nhập



#### 4. Menu chính

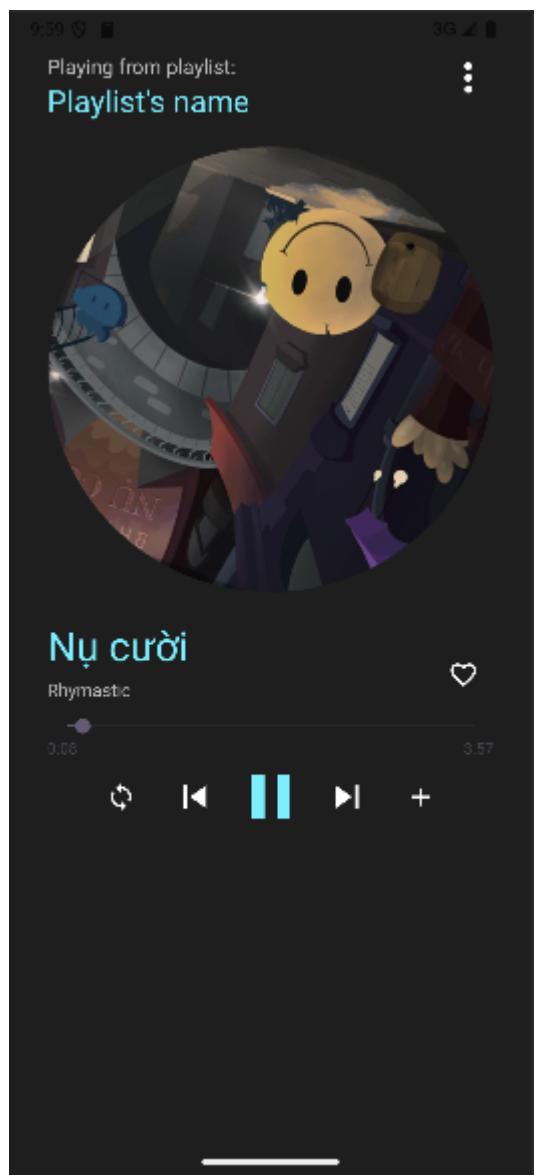


##### 5. Giao diện về ca sĩ:

Ở đây bao gồm thông tin và những bài hát nổi bật của ca sĩ đó.



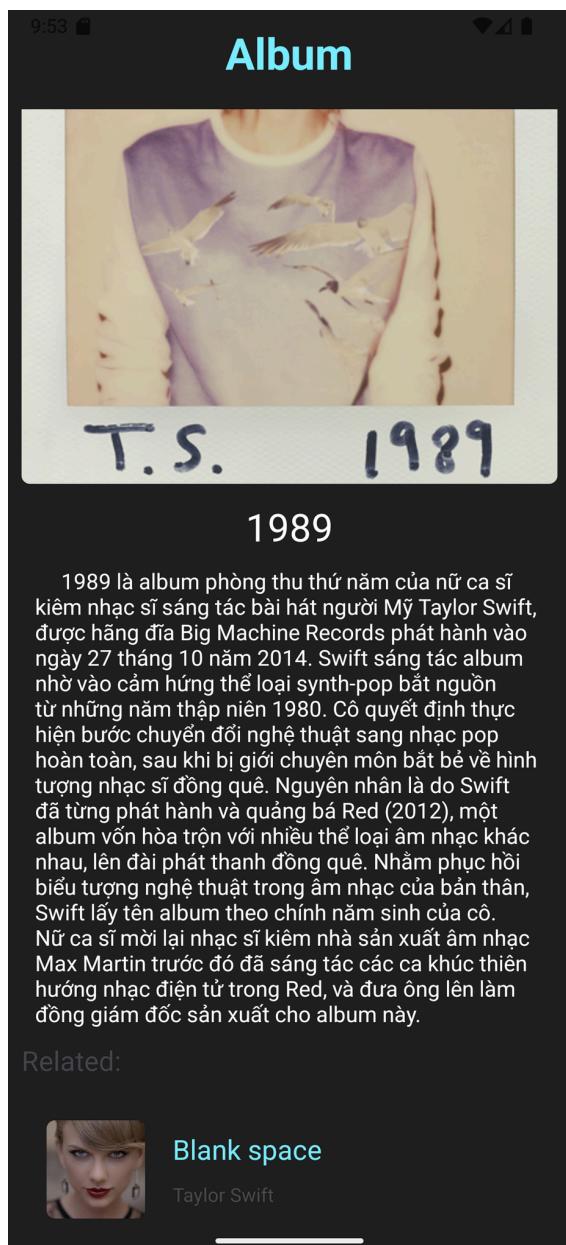
## 6. Giao diện phát nhạc



7. Giao diện thẻ loại nhạc



## 8. Giao diện Album



## KẾT LUẬN

(Trình bày thành 3 đoạn văn nêu **Ưu điểm**, **nhược điểm** và **hướng phát triển chủ đề**)

### Ưu điểm:

Ứng dụng Musium mang đến cho người dùng trải nghiệm âm nhạc toàn diện với các tính năng phong phú như tìm kiếm và phát nhạc nhanh chóng, tạo và quản lý danh sách phát cá nhân, và gợi ý nhạc dựa trên sở thích cá nhân. Chức năng nghe offline hỗ trợ người dùng tiết kiệm dữ liệu di động và duy trì trải nghiệm âm nhạc liền mạch. Giao diện thân thiện, dễ sử dụng, giúp người dùng có thể tìm kiếm và tận hưởng âm nhạc một cách dễ dàng. Tính năng chia sẻ âm nhạc qua mạng xã hội còn giúp người dùng kết nối với cộng đồng và chia sẻ sở thích âm nhạc của mình.

### **Nhược điểm:**

Mặc dù Musium sở hữu nhiều tính năng hữu ích, nhưng yêu cầu phi chức năng như khả năng mở rộng và độ tin cậy cao đòi hỏi hệ thống phải được tối ưu hóa kỹ lưỡng để đảm bảo hiệu suất ổn định khi số lượng người dùng tăng cao. Vấn đề bảo mật cũng là một thách thức lớn vì ứng dụng phải đảm bảo an toàn thông tin cá nhân của người dùng, nhất là khi chia sẻ qua mạng xã hội. Bên cạnh đó, việc duy trì tính thân thiện với người dùng đòi hỏi giao diện và khả năng tương tác phải được cải tiến liên tục để đáp ứng nhu cầu đa dạng, đặc biệt khi người dùng có thể gặp khó khăn trong việc thao tác với các thiết lập tùy chỉnh phức tạp.

### **Hướng phát triển:**

Trong tương lai, ứng dụng có thể tập trung vào cải tiến thuật toán gọi ý âm nhạc để mang lại trải nghiệm cá nhân hóa cao hơn, đồng thời cải thiện khả năng tương tác giữa người dùng và nghệ sĩ thông qua các tính năng cập nhật hoạt động nghệ sĩ và các sự kiện âm nhạc. Về mặt phi chức năng, cần chú trọng nâng cấp hệ thống để đáp ứng khả năng mở rộng, đảm bảo độ tin cậy và khả năng bảo mật tốt hơn. Tính năng hỗ trợ đa ngôn ngữ cũng nên được mở rộng để thu hút người dùng từ các thị trường khác nhau, mang đến trải nghiệm mượt mà cho người dùng trên toàn cầu.

## DANH MỤC TÀI LIỆU THAM KHẢO

- [1]. *Giáo trình Phát triển ứng dụng cho các thiết bị di động.*
- [2]. <https://chatgpt.com/>
- [3]. <https://developer.android.com/design/ui?hl=vi>
- [4]. <https://viblo.asia/p/firebase-trong-android-studio-AQ3vVk1bRbOr>

## PHỤ LỤC

- Xác thực: Cần có bước xác thực để ứng dụng có thể truy cập vào Google Sheet. Bạn có thể sử dụng OAuth2 để cho phép ứng dụng truy cập vào tài khoản Google của bạn.
- API Google Sheets: Tìm hiểu và sử dụng API của Google Sheets để thực hiện các thao tác đọc, ghi, cập nhật dữ liệu trên sheet.
- Hiệu năng: Tốc độ đọc/ghi dữ liệu có thể phụ thuộc vào tốc độ mạng và API của Google Sheets.
- Bảo mật: Cần bảo mật thông tin xác thực của bạn để tránh rò rỉ thông tin.
- **Code minh họa đầy đủ đã thực hiện**

- **MainActivity.java**

```

package com.example.launchscreen.HomeScreen;

import android.os.Bundle;
import android.view.MenuItem;

import androidx.activity.EdgeToEdge;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.viewpager2.widget.ViewPager2;

import com.example.launchscreen.HomeScreen.Fragments.ExploreFragment;
import com.example.launchscreen.HomeScreen.Fragments.HomeFragment;
import com.example.launchscreen.HomeScreen.Fragments.LibraryFragment;
import com.example.launchscreen.HomeScreen.Fragments.ViewPagerAdapter;
import com.example.launchscreen.R;
import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.android.material.navigation.NavigationBarView;

import java.util.ArrayList;

public class HomeActivities extends AppCompatActivity {

    ViewPager2 mainviewPager;
    ArrayList<Fragment> FragmentArrayList = new ArrayList<>();

    BottomNavigationView bottomNavigationView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_home_activities);

        bottomNavigationView = findViewById(R.id.bottom_nav);
        mainviewPager = findViewById(R.id.main_view_pager);

        FragmentArrayList.add(new HomeFragment());
        FragmentArrayList.add(new ExploreFragment());
    }
}

```

```

FragmentArrayList.add(new LibraryFragment());

        ViewPagerAdapter viewPagerAdapter = new ViewPagerAdapter(this,
FragmentArrayList);
        mainviewPager.setAdapter(viewPagerAdapter);

        mainviewPager.registerOnPageChangeCallback(new
ViewPager2.OnPageChangeCallback() {
    @Override
    public void onPageSelected(int position) {

        switch (position){
            case 0:
                bottomNavigationView.setSelectedItemId(R.id.nav_home);
                break;
            case 1:

bottomNavigationView.setSelectedItemId(R.id.nav_explore);
                break;
            case 2:

bottomNavigationView.setSelectedItemId(R.id.nav_Library);
                break;
        }

        super.onPageSelected(position);
    }
});;

        bottomNavigationView.setOnItemSelectedListener(new
NavigationBarView.OnItemSelectedListener() {
    @Override
    public boolean onNavigationItemSelected(@NonNull MenuItem item) {

        if (item.getItemId() == R.id.nav_home){
            mainviewPager.setCurrentItem(0);
        }
        if (item.getItemId() == R.id.nav_explore){
            mainviewPager.setCurrentItem(1);
        }
        if (item.getItemId() == R.id.nav_Library){
            mainviewPager.setCurrentItem(2);
        }
        return true;
    }
});;
}
}

```

## o HomeFragment.java

```

package com.example.launchscreen.HomeScreen.Fragments;

import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;

```

```

import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.example.launchscreen.HomeScreen.Activities.ItemOnItemClickListener;
import com.example.launchscreen.HomeScreen.Activities.SettingsActivity;
import com.example.launchscreen.HomeScreen.ModelsandAdapter.HomeChildAdapter;
import com.example.launchscreen.R;
import com.example.launchscreen.HomeScreen.ModelsandAdapter.HomeChildModel;
import com.example.launchscreen.HomeScreen.ModelsandAdapter.HomeParentAdapter;
import com.example.launchscreen.HomeScreen.ModelsandAdapter.HomeParentModel;
import com.example.launchscreen.items.ArtistModel;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebaseio.database.DataSnapshot;
import com.google.firebaseio.database.DatabaseError;
import com.google.firebaseio.database.DatabaseReference;
import com.google.firebaseio.database.FirebaseDatabase;
import com.google.firebaseio.database.ValueEventListener;

import java.util.ArrayList;

public class HomeFragment extends Fragment implements
HomeParentAdapter.OnHomeChildClickListener {

    private TextView userName;

    private DatabaseReference hDatabase;
    private FirebaseAuth hAuth;

    ImageButton settingbtn;

    RecyclerView recyclerView;
    ArrayList<HomeParentModel> homeParentModelsArrayList;
    ArrayList<HomeChildModel> popularGenre;
    ArrayList<HomeChildModel> popularAlbum;
    ArrayList<HomeChildModel> popularArtist;

    //interface cho skien click
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_home, container, false);
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
    savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
    }
}

```

```

    userName = view.findViewById(R.id.user_name);
    //firebase
    hDatabase = FirebaseDatabase.getInstance().getReference();
    hAuth = FirebaseAuth.getInstance();

    recyclerView = view.findViewById(R.id.rv_parent);

    popularGenre = new ArrayList<>();
    popularAlbum = new ArrayList<>();
    popularArtist = new ArrayList<>();
    homeParentModelsArrayList = new ArrayList<>();

    HomeParentAdapter homeParentAdapter;

    settingbtn = view.findViewById(R.id.btn_home_setting);

    //Lay ten cua user dang dang nhap tu database
    FirebaseUser currentUser = hAuth.getCurrentUser();
    if (currentUser != null){
        String userID = currentUser.getUid();

    hDatabase.child("user").child(userID).child("name").addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            String name = snapshot.getValue(String.class);

            if (name !=null){
                userName.setText(name);
            }
            else{
                userName.setText("cant get name.");
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Toast.makeText(requireContext(), "Loading name error",
            Toast.LENGTH_SHORT).show();
        }
    });
}

    popularArtist.add(new HomeChildModel(R.drawable.rhymastic_img,
"Rhymastic"));
    popularArtist.add(new HomeChildModel(R.drawable.chillies_img,
"Chillies"));
    popularArtist.add(new HomeChildModel(R.drawable.sontung_img, "Son Tung
M-TP"));
    popularArtist.add(new HomeChildModel(R.drawable.buctuong_img, "Buc
Tuong"));
    popularArtist.add(new HomeChildModel(R.drawable.taylor swift img, "Taylor
Swift"));
    popularArtist.add(new HomeChildModel(R.drawable.vu img, "Vu"));

}

```

```

        homeParentModelsArrayList.add(new HomeParentModel("Popular Artist",
popularArtist));

        popularGenre.add(new HomeChildModel(R.drawable.pop_img, "Pop"));
        popularGenre.add(new HomeChildModel(R.drawable.rap_img, "Rap"));
        popularGenre.add(new HomeChildModel(R.drawable.podcast_img, "Podcast"));
        popularGenre.add(new HomeChildModel(R.drawable.usuk_img, "US-UK"));
        popularGenre.add(new HomeChildModel(R.drawable.metal_img, "Rock"));
        popularGenre.add(new HomeChildModel(R.drawable.indie_img, "Indie"));

        homeParentModelsArrayList.add(new HomeParentModel("Popular Genre",
popularGenre));

        popularAlbum.add(new HomeChildModel(R.drawable.album1989_img, "1989"));
        popularAlbum.add(new HomeChildModel(R.drawable.reputation_img,
"Reputation"));
        popularAlbum.add(new HomeChildModel(R.drawable.quakhungcuaso,
"QuaKhungCuaSo"));

        homeParentModelsArrayList.add(new HomeParentModel("Popular Album",
popularAlbum));

        homeParentAdapter = new HomeParentAdapter(homeParentModelsArrayList,
requireActivity(), this);
        recyclerView.setLayoutManager(new LinearLayoutManager(requireActivity()));
        recyclerView.setAdapter(homeParentAdapter);

        homeParentAdapter.notifyDataSetChanged();

        //nut setting
        settingbtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startActivity(new Intent(requireActivity(),
SettingActivity.class));
            }
        });
    }

    @Override
    public void onHomeChildClicked(HomeChildModel model) {
        Intent intent = new Intent(requireContext(), ItemOnClickActivity.class);
        intent.putExtra("title", model.getTitle());
        intent.putExtra("image", model.getImage());
        startActivity(intent);
    }
}

```