NOTE WORTHY

Ana Alvarez, Sneha Bista, Emma Hockett, Katrina Lee, Nuvia Mata, Lauren Nicolas, and Sarah Yakum

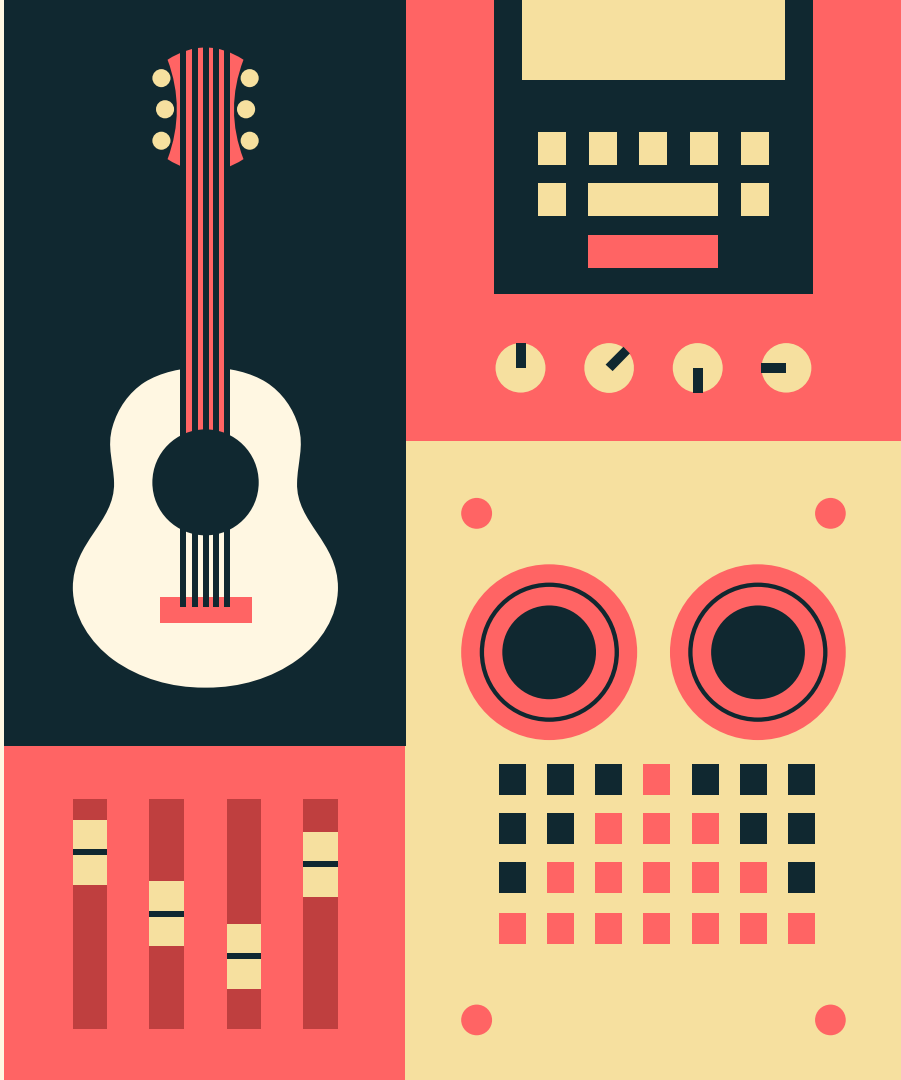


TABLE OF CONTENTS

01

OUR PURPOSE

NOTEWORTHY

Our **application** is a **music hub** that combines Rotten Tomatoes, Letterboxd, and music listening platforms all into one place. Users can:

- Read critics' and audiences reviews
- Create comments and discuss music opinions through general forums
- Play fun song games based on user's interest

The goal is give users a platform to share and read other people's opinions on music for songs they listen to.

KEY FEATURES ★

SONG CRITIQUE

Add comments or post reviews on specific songs or on discussion boards

SONG GAMES

Play fun games, such as Karaoke, Guess the Lyric, and Finish the Lyric

PERSONALIZATION

Provide listening weekly, monthly, and yearly statistics; Recommend genres

COMMUNITY

Find other similar listeners to share opinions with and discover new artists

02

PLANNING

★ COST ESTIMATION - FUNCTION POINT ALGORITHM

We used **FPA** to determine the cost. Below is a breakdown of our function categories.

	USER INPUTS	USER OUTPUTS	USER QUERIES	DATA FILES	EXTERNAL INTERFACES
EXAMPLES (NOT ALL):	Username	Trending music	Find song title	User account information	Other streaming platforms
	Password	New releases	Find artist name	User listening data	Youtube for music videos
	Song ratings	Discussion board feed	Find reviews	Rating table	API for application data
	Discussion posts	User profile information	Find friends/users	Song games table	Mobile application
	Liking comments	Playing music	Find discussion forums	Playlist table	Billboard statistics for tending and new music
TOTAL:	16	20	9	13	7

★ COST ESTIMATION - GROSS FUNCTION POINT

	Function Category	Count	Complexity			Count x Complexity
			Simple	Average	Complex	
1	# of User Input	16	3	**4**	6	= 16 x 4= 64
2	# of User Output	20	**4**	5	7	= 20 x 4 = 80
3	# of User Queries	9	**3**	4	6	= 9 x 3 = 27
4	# of Data Files and Relational Tables	13	7	**10**	15	= 13 x 10 = 130
5	# of External Interfaces	7	5	7	**10**	= 7 x 10 = 70
					GFP	**= 371**

After processing the complexity adjustment, the estimated effort based on a team of **15 people** working on **6 functions points** per week would will around **5 weeks.**

PROJECT TIMELINE

PLANNING
Select resources
Research

REQUIREMENTS ANALYSIS
Review
Sign Off

DESIGN
Proof of concept
Technical specifications
Design review

DEVELOPMENT
Test planning
Documentation
Deployment Plan

IMPLEMENTATION & REPORTING
Integration
Quality control
Budget Control

TEST & QUALITY ASSURANCE
Demo
Issues resolution
Improvement plan

MAINTENANCE & COMPLETION
Documentation
Finalize model

LAUNCH

03

REQUIREMENTS

FUNCTIONAL REQUIREMENTS

DISPLAY TRENDING MUSIC

The system will display popular music on the user's home page.

LOG IN USERS

The system will allow users to log into their account

CONNECT TO OTHER STREAMING PLATFORMS

Signed-in users must be able to connect other platforms to play music

VIEW AND POST REVIEWS

Users must be able to view and post comments to a community forum

VIEW PERSONALIZED STATISTICS

Users must be able to view personalized listening history and minutes.

SEARCH & SORT THROUGH SONGS

User must be able to go through database of songs by genre, title, or artist

NON FUNCTIONAL REQUIREMENTS ★

PRODUCT REQUIREMENTS

- Usability
  - … shall provide a new user navigation training that takes at most 1 minute.
- Dependability
  - … shall be able to load the user's requested data within 2 seconds for 95% of requests like searching for songs, creating playlists, and posting/editing forum posts

EFFICIENCY REQUIREMENTS

- Performance
  - … shall load the user's library and playlist within 2 seconds.
- Operational
  - … shall be constructed in about 5-6 months with an additional 2 months of testing.

*Not all non functional requirements are included in this list.

NON FUNCTIONAL REQUIREMENTS ★

EXTERNAL REQUIREMENTS

- Regulatory
  - … shall adhere to industry standards for music streaming and distribution.
- Ethical
  - … shall have a tagging system that allows artists to be credited on songs

LEGISLATIVE REQUIREMENTS

- Accounting
  - … shall compensate the artists for plays, downloads, and ad revenue.
- Safety
  - … shall have a real-time monitor for system performance and security breaches.

*Not all non functional requirements are included in this list.

04

SOFTWARE DIAGRAMS

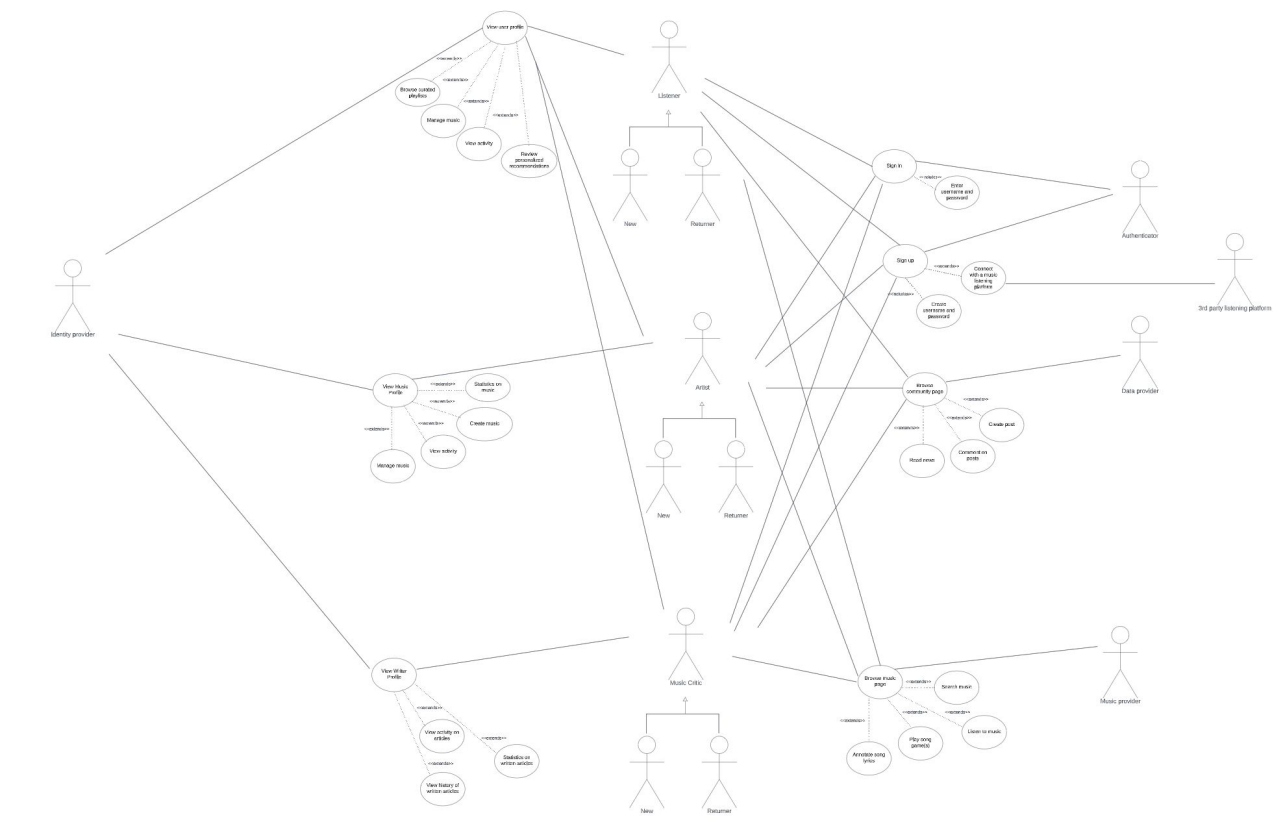
★ USE CASE DIAGRAM - OVERVIEW

ACTORS:

- Identity Provider
- Listeners
- Artists
- Music Critics
- Data Provider

ACTIONS:

- View User/Music/Writer Page
- Browse forums and music



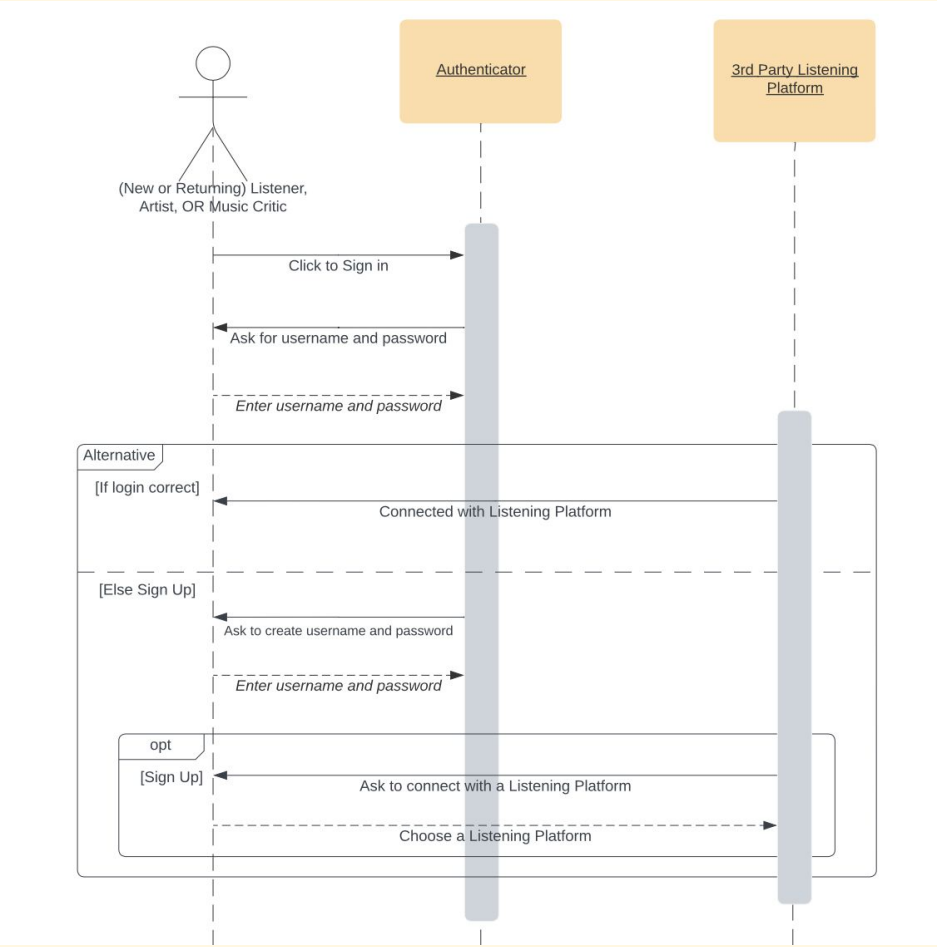
★ SEQUENCE DIAGRAM - SIGNING UP

Authenticator:

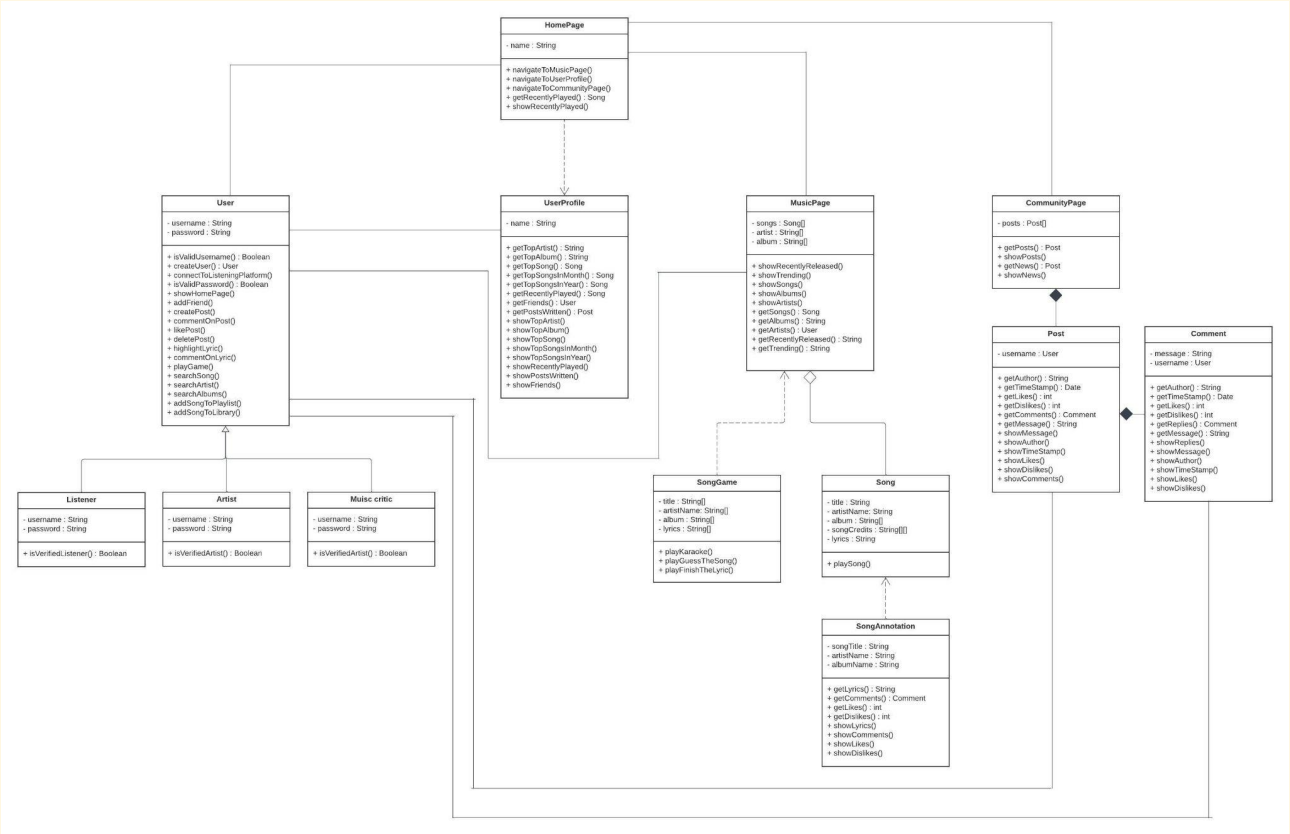
- If user has an account enters Username and password
- If not user can create an account

3rd Party Listening Platform:

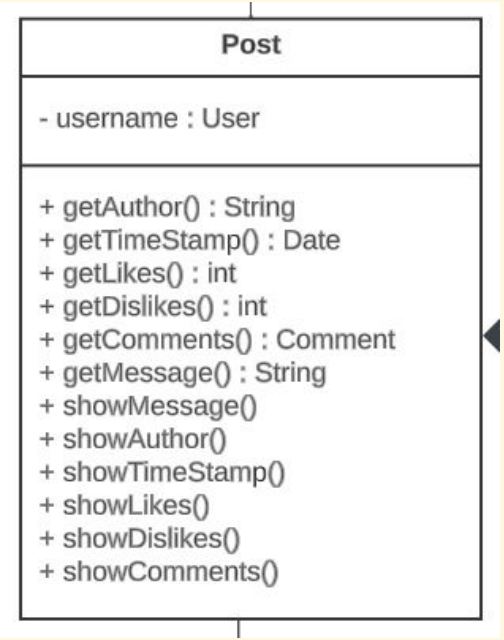
- Allows user to connect to existing account from third party



CLASS DIAGRAM - OVERALL VIEW ★



CLASS DIAGRAM - DETAILS ★

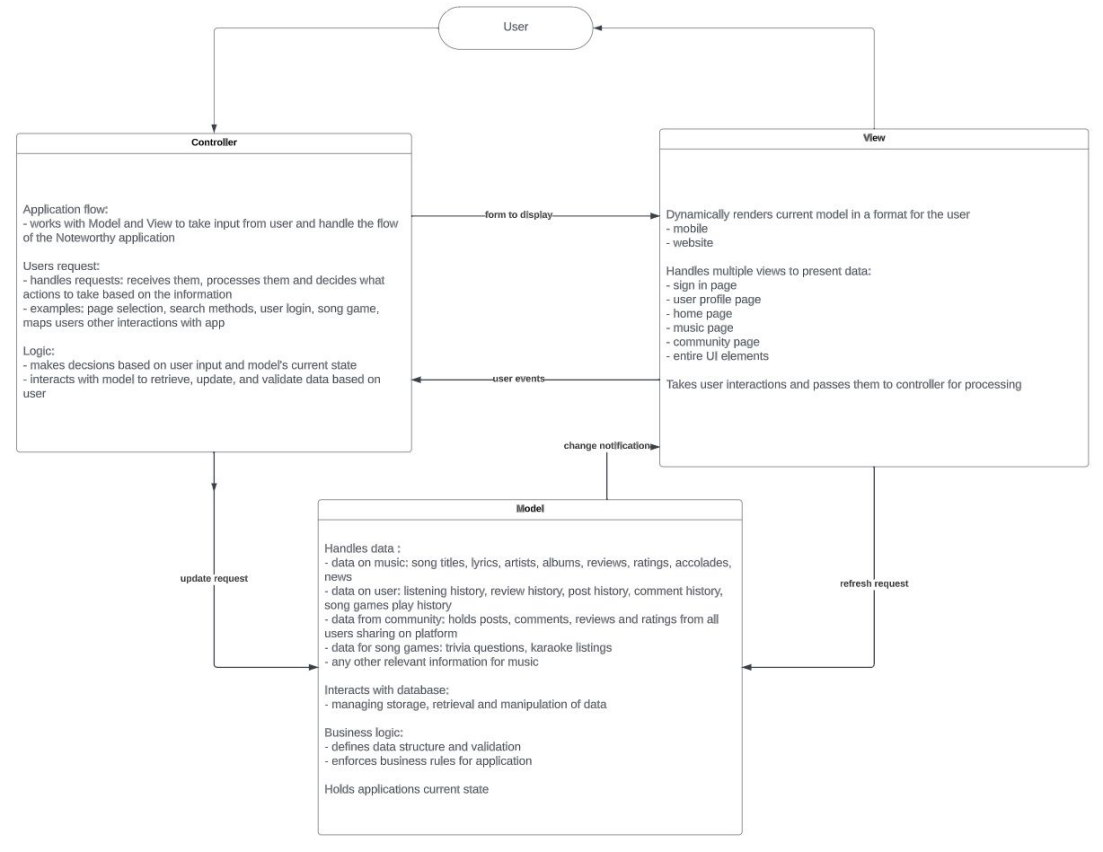


Examples of the three different classes and their methods

★ ARCHITECTURAL DESIGN: MVC PATTERN

WHY:

- Displays the interactions between the user and the data
- Helps organize the application into 3 main levels



★ ARCHITECTURAL DESIGN: MODEL AND CONTROLLER

Model

Handles data :
- data on music: song titles, lyrics, artists, albums, reviews, ratings, accolades, news
- data on user: listening history, review history, post history, comment history, song games play history
- data from community: holds posts, comments, reviews and ratings from all users sharing on platform
- data for song games: trivia questions, karaoke listings
- any other relevant information for music

Interacts with database:
- managing storage, retrieval and manipulation of data

Business logic:
- defines data structure and validation
- enforces business rules for application

Holds applications current state

Controller

Application flow:
- works with Model and View to take input from user and handle the flow of the Noteworthy application

Users request:
- handles requests: receives them, processes them and decides what actions to take based on the information
- examples: page selection, search methods, user login, song game, maps users other interactions with app

Logic:
- makes decsions based on user input and model's current state
- interacts with model to retrieve, update, and validate data based on user

★ ARCHITECTURAL DESIGN: VIEW

View

Dynamically renders current model in a format for the user
- mobile
- website

Handles multiple views to present data:
- sign in page
- user profile page
- home page
- music page
- community page
- entire UI elements

Takes user interactions and passes them to controller for processing

05

COMPETITORS

OUR COMPETITORS

Spotify

Key Features

- Spotify wrapped
- Generated playlists
- Personality games

Receiptify

Key Features

- Top songs/albums/ artists based on time period
- UI is based on a receipt

Letterboxd

Key Features

- Reviews: post, react, and comment on reviews
- Playlists for movies
- News

Genius

Key Features

- Song annotations

HOW NOTEWORTHY STANDS OUT

Noteworthy incorporates all of our competitors' key features into one location:

- **Song critique:** for each song, users can add comments to lyrics
- **Song games:** include various song games such as karaoke, guess the lyric, finish the lyric, etc.
    - Songs will be randomized but be based on the user's data or whatever is trending.
- **Personalization:**
    - Provide personalized songs, genres, artists, and playlists
    - Provide statistics on a weekly, monthly, and yearly basis.
        - Statistics would range from top song(s), artist(s), album(s), etc to a personality based statistic (i.e. what type of aura do they have).
- **Community:**
    - Allow critics, fans, and artists to post announcements, opinions, etc and allows other users to comment on them
    - Get updates on the musical world
    - Share music to other users

06

TESTING

UNIT CODE ★

Dataset credit: https://www.kaggle.com/datasets/nelgiriyewithana/top-spotify-songs-2023

```python
import pandas as pd

class Chart:
    def __init__(self, filename):
        self.filename = filename
        df = pd.read_csv(filename, encoding="ISO-8859-1")

        df["streams"] = pd.to_numeric(df["streams"], errors="coerce", downcast="integer")
        self.df = df

    def get_top(self, n, category):
        df = self.df
        if category == "songs":
            topn = df.nlargest(n, "streams")[["track_name", "artist(s)_name", "streams"]]
            return topn.set_index(pd.Index(range(1, n+1)))
        elif category == "artists":
            counts = df["artist(s)_name"].str.split(", ").explode().value_counts()
            topn = counts.reset_index(name="songs").nlargest(n, "songs")
            return topn
        else:
            pass
```

TEST CASE 1 - TOP 5 SONGS (SUCCESS) ★

```
def test_top5songs_success(self):
    expected_df = pd.DataFrame.from_dict({"track_name": {1: "Blinding Lights", 2: "
        Shape of You", 3: "Someone You Loved", 4: "Dance Monkey", 5: "Sunflower -
        Spider-Man: Into the Spider-Verse"}, "artist(s)_name": {1: "The Weeknd", 2: "
        Ed Sheeran", 3: "Lewis Capaldi", 4: "Tones and I", 5: "Post Malone, Swae Lee"
        }, "streams": {1: 3703895074.0, 2: 3562543890.0, 3: 2887241814.0, 4:
        2864791672.0, 5: 2808096550.0}})
    try:
        pd.testing.assert_frame_equal(self.chart.get_top(5, "songs"), expected_df)
    except AssertionError:
        raise AssertionError("FAILED: query for top 5 songs")
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.008s

OK
```

TEST CASE 2 - TOP 5 SONGS (FAILURE) ★

```
F
======================================================================
FAIL: test_top5songs_failure (__main__.TestGetTop.test_top5songs_failure)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\kat\Downloads\test_get_top.py", line 23, in test_top5songs_failure
    pd.testing.assert_frame_equal(self.chart.get_top(5, "songs"), expected_df)
  File "C:\Python312\Lib\site-packages\pandas\_testing\asserters.py", line 1209, in assert_frame_equal
    assert_series_equal(
  File "C:\Python312\Lib\site-packages\pandas\_testing\asserters.py", line 1005, in assert_series_equal
    _testing.assert_almost_equal(
  File "testing.pyx", line 55, in pandas._libs.testing.assert_almost_equal
  File "testing.pyx", line 173, in pandas._libs.testing.assert_almost_equal
  File "C:\Python312\Lib\site-packages\pandas\_testing\asserters.py", line 598, in raise_assert_detail
    raise AssertionError(msg)
AssertionError: DataFrame.iloc[:, 0] (column name="track_name") are different

DataFrame.iloc[:, 0] (column name="track_name") values are different (20.0 %)
[index]: [1, 2, 3, 4, 5]
[left]:  [Blinding Lights, Shape of You, Someone You Loved, Dance Monkey, Sunflower - Spider-Man: Into the Spider-Verse]
[right]: [Blinding Lights, Shape of You, Someone You Loved, Hotel Room Service, Sunflower - Spider-Man: Into the Spider-Verse]
At positional index 3, first diff: Dance Monkey != Hotel Room Service

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\kat\Downloads\test_get_top.py", line 25, in test_to
    raise AssertionError("FAILED: query for top 5 songs")
AssertionError: FAILED: query for top 5 songs

----------------------------------------------------------------------
Ran 1 test in 0.010s

FAILED (failures=1)
```

```python
def test_top5songs_failure(self):
    expected_df = pd.DataFrame.from_dict({"track_name": {1: "Blinding Lights", 2: "
        Shape of You", 3: "Someone You Loved", 4: "Hotel Room Service", 5: "
        Sunflower - Spider-Man: Into the Spider-Verse"}, "artist(s)_name": {1: "The
        Weeknd", 2: "Ed Sheeran", 3: "Lewis Capaldi", 4: "Pitbull", 5: "Post Malone,
        Swae Lee"}, "streams": {1: 3703895074.0, 2: 3562543890.0, 3: 2887241814.0, 4:
         2864791672.0, 5: 2808096550.0}})
    try:
        pd.testing.assert_frame_equal(self.chart.get_top(5, "songs"), expected_df)
    except AssertionError:
        raise AssertionError("FAILED: query for top 5 songs")
```

TEST CASE 3 - TOP 3 ARTISTS (SUCCESS) ★

```
def test_top3artists_success(self):
    expected_df = pd.DataFrame.from_dict({"artist(s)_name": {0: "Bad Bunny", 1: "
        Taylor Swift", 2: "The Weeknd"}, "songs": {0: 40, 1: 38, 2: 37}})
    try:
        pd.testing.assert_frame_equal(self.chart.get_top(3, "artists"), expected_df)
    except AssertionError:
        raise AssertionError("FAILED: query for top 3 charting artists")
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.009s

OK
```

TEST CASE 4 - TOP 3 ARTISTS (FAILURE) ★

```
F
======================================================================
FAIL: test_top3artists_failure (__main__.TestGetTop.test_top3artists_failure)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\kat\Downloads\test_get_top.py", line 51, in test_top3artists_failure
    pd.testing.assert_frame_equal(self.chart.get_top(3, "artists"), expected_df)
  File "C:\Python312\Lib\site-packages\pandas\_testing\asserters.py", line 1209, in assert_frame_equal
    assert_series_equal(
  File "C:\Python312\Lib\site-packages\pandas\_testing\asserters.py", line 1005, in assert_series_equal
    _testing.assert_almost_equal(
  File "testing.pyx", line 55, in pandas._libs.testing.assert_almost_equal
  File "testing.pyx", line 173, in pandas._libs.testing.assert_almost_equal
  File "C:\Python312\Lib\site-packages\pandas\_testing\asserters.py", line 598, in raise_assert_detail
    raise AssertionError(msg)
AssertionError: DataFrame.iloc[:, 0] (column name="artist(s)_name") are different

DataFrame.iloc[:, 0] (column name="artist(s)_name") values are different (66.66667 %)
[index]: [0, 1, 2]
[left]:  [Bad Bunny, Taylor Swift, The Weeknd]
[right]: [Bugs Bunny, SpongeBob SquarePants, The Weeknd]
At positional index 0, first diff: Bad Bunny != Bugs Bunny

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\kat\Downloads\test_get_top.py", line 53, in test_top3artists_failure
    raise AssertionError("FAILED: query for top 3 chartin
AssertionError: FAILED: query for top 3 charting artists

----------------------------------------------------------
Ran 1 test in 0.010s

FAILED (failures=1)
```

```python
def test_top3artists_failure(self):
    expected_df = pd.DataFrame.from_dict({"artist(s)_name": {0: "Bugs Bunny", 1: "
        SpongeBob SquarePants", 2: "The Weeknd"}, "songs": {0: 40, 1: 38, 2: 37}})
    try:
        pd.testing.assert_frame_equal(self.chart.get_top(3, "artists"), expected_df)
    except AssertionError:
        raise AssertionError("FAILED: query for top 3 charting artists")
```

07

CONCLUSION & FUTURE WORKS

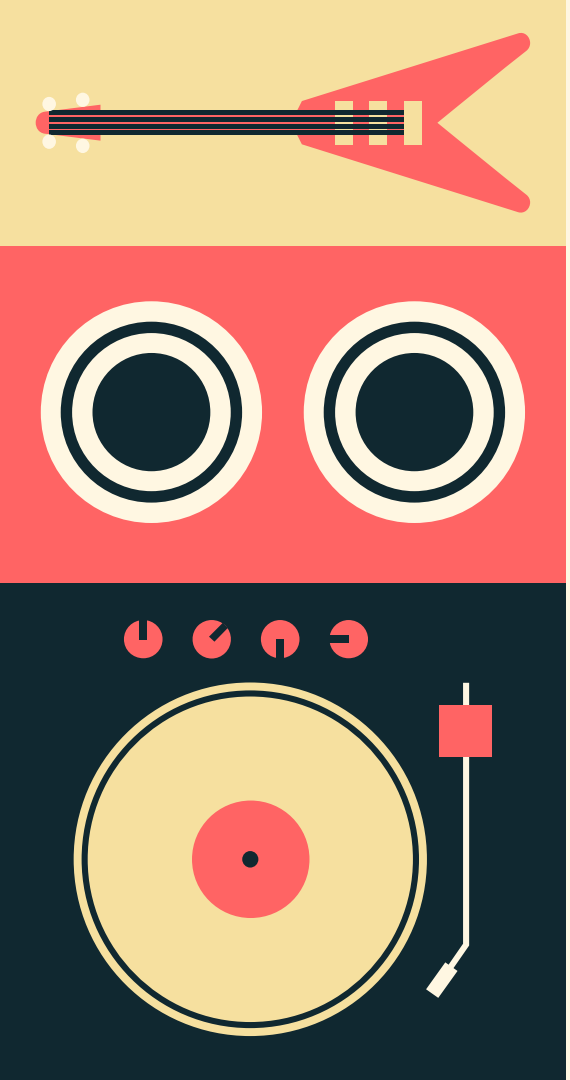
CONCLUSION & FUTURE WORK

DEVELOP & EXPAND

In the future, we hope to:

- Start developing and release the first iteration of our app
- Promote our app to gain new users

With our love for music, we were able to transform Noteworthy from a simple music app to an interactive community-based and music-sharing application.



THANK YOU!

ANY QUESTIONS OR COMMENTS?

Ana Alvarez: ana.alvarez@utdallas.edu
Sneha Bista: sneha.bista@utdallas.edu
Emma Hockett: emma.hockett@utdallas.edu
Katrina Lee: katrina@utdallas.edu
Nuvia Mata: nuvia.mata@utdallas.edu
Lauren Anne Nicolas: lauren.nicolas@utdallas.edu
Sarah Yakum: sarah.yakum@utdallas.edu