Out Of Tune

by Nadia Safadi, Sagi Kayat, Yonatan Diga, Dan Ben-Dror

Abstract

The primary goal of the "Out Of Tune" project is to assist musicians, including both singers and instrument players, in enhancing and perfecting their musical abilities. This will be achieved through monitoring of the musician's performances and comparison with the original performance, providing feedback and grading based on accuracy.

Contents

Chapter 1 - Introduction

1.1 The Problem Domain

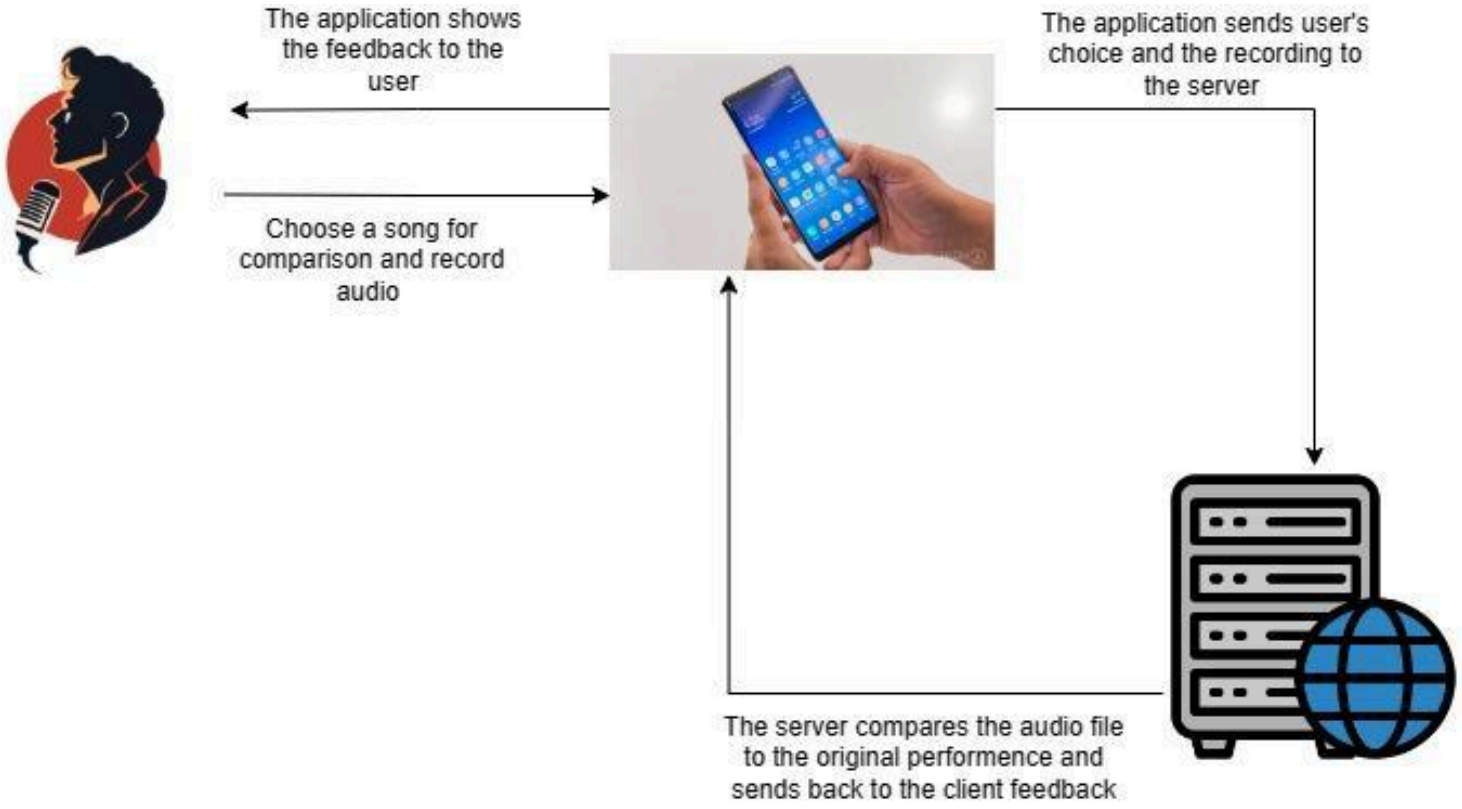
Many people want to practice their musical abilities, from amateur singers to professional instrument players, and even Bar-Mitzva boys. The problem is that professional mentors are not always available and are costly.

As the main subject of our problem domain, we chose to make an application to replace those teachers and help people to improve their abilities in comparison to the original performances.

Implementing an application over our problem domain faces some complex problems. We will suggest and implement solutions to those problems in the development process.

- Performance Comparison and Grading: Evaluating a musician's performance against the original recording requires complex algorithms for pitch and rhythm analysis, as well as a grading system that objectively assess accuracy.
- Continuous Improvement: Implementing mechanisms for users to track their progress over time, reflect on past performances, and receive constructive feedback is essential for continuous improvement in musical skills.
- Database Management: Maintaining an extensive and updated database of original musical performances is crucial for accurate comparison and analysis. This involves considerations of data storage, retrieval, and organization.
- Real-time Performance Monitoring: The challenge is developing a system that can accurately monitor live musical performances in real time, capturing nuances and providing immediate feedback.

1.2 Context


The application shows the feedback to the user
The application sends user's choice and the recording to the server
Choose a song for comparison and record audio
The server compares the audio file to the original performence and sends back to the client feedback

1.3 Vision

The main goal of our project is to empower amateur musicians (or anyone else who likes to play and sing), to achieve mastery and excellence in their craft. The project intends to become an innovative tool that not only monitors and evaluates performances but also serves as a supportive and personalized mentor on the journey towards musical perfection.

We are about to deal with audio analysis and comparison, in order to provide reliable feedback regarding our users' advancement.

In addition, we intend to create a nice and intuitive user interface suitable for all audiences and compatible with both Android and IOS.

Our platform supports comparison with audio files from the server's database or uploaded files given by the user.

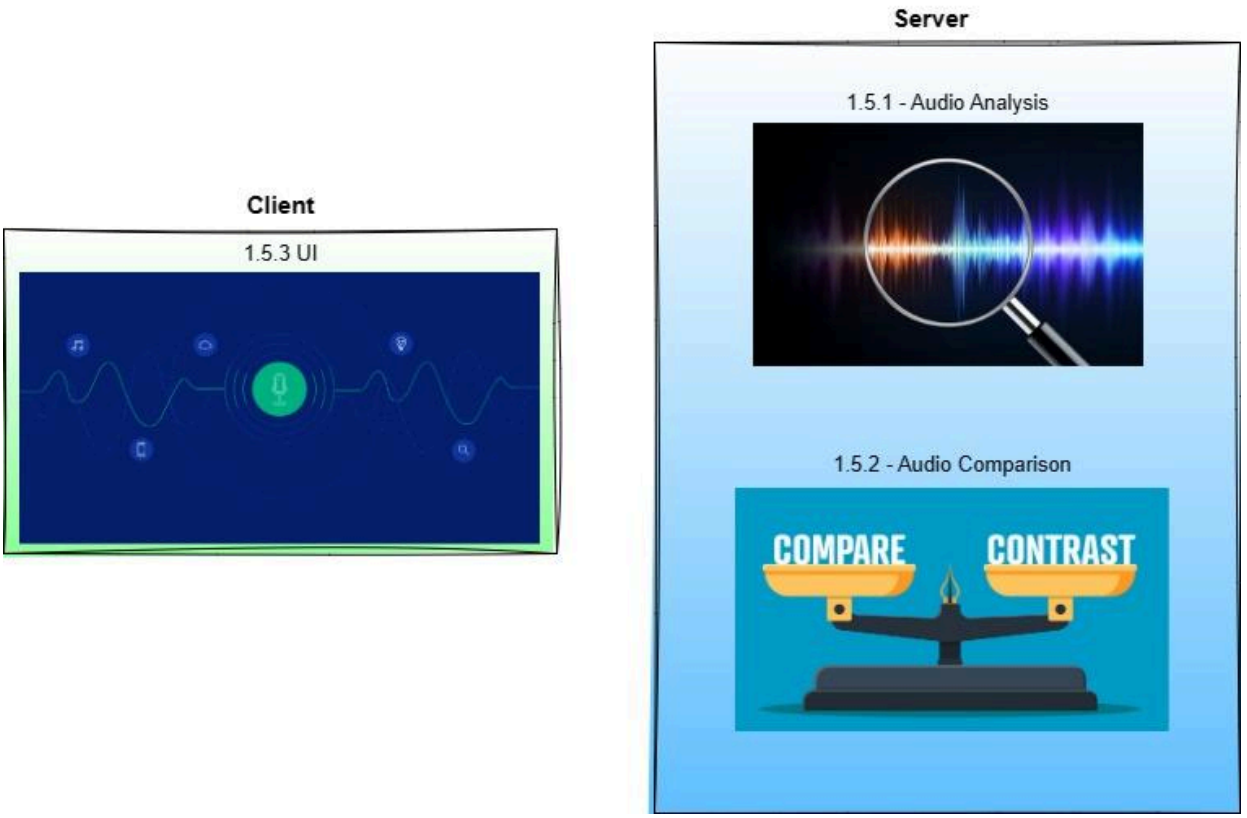
The user can watch his previous performances' grade and graph.

Finally, after the problem is handled, future versions of the project may handle songs with music and singing, and will be able to separate between them, allowing the users to compare themselves to the official version of the songs. Moreover, the application may be able to get a recording of singing and playing and grade each of them separately in comparison to the original version.

1.4 Stakeholders

- Users:
  The users of our application will be of a wide range - professional singers and instrument players, amateurs, and anyone else who is passionate about music. (add Bar-Mitzva boys?)
- Customers:
  The main customer of our project is Roni Stren, who came up with the idea of an application to help his Bar-Mitzva boy to train reading in the Torah.
- Experts:
  We can consult audio experts on how to analyze and compare audio files.
  In addition, we can ask music experts to try the application and give us feedback for future development.

1.5 Software Context



1.5.1 Audio Analysis

This component receives an audio file from the user and analyzes it. Its main purpose is to convert the file to a more suitable format for comparison in the next phase. It does so by using audio analysis tools and methods that take the audio input and transform it into a vector of numbers representing the original file. Therefore, this phase has a major significance on the process and must be as accurate as possible. In the end of the analysis, the audio file is discarded and the vector is passed on to the Audio Comparison (1.5.2). The Audio Analysis will be a part of the server functionality.

1.5.2 Audio Comparison

This component is the core component of the system, responsible for comparing the user's performance to the original. In order to do that, the algorithm must be able to ignore disturbances like time shifts or

background noises, while accurately comparing the two audio files and returning reliable feedback to the user. This can be achieved by using several popular audio vectors comparison methods. The input of the Audio comparison is the vector produced by the Audio Analysis (1.5.1), and the output is a vector representing the grade for each time frame of the performance, as well as a global grade for it. The Audio Comparison component will be a part of the server functionality.

1.5.3 UI

The UI will be a mobile application, responsible for getting the audio input from the user, passing it to the server, and getting the server's output to show to the user. The UI will be compelling yet simple, to allow all kinds of users to use the application, not just professional musicians. In order to get to a large audience, we will make a multi-platform app compatible with Android and IOS devices, using popular technologies to do so. Those technologies will help us with client-server communication as well, allowing the application to maintain short waiting times between the recording and the returned grade.

2 Chapter 2 - Usage Scenarios

2.1 User Profiles - The Actors

2.1.1 The End User

This actor uses the system to record audio and compare the recording to the original performance of the actor's choice. In addition, the actor can watch previous performances' summaries, where there is more specific information about each performance.

2.2 Use-cases

Register

- **Actor:** not registered user
- **Preconditions:** the user does not exist
- **Postconditions :** the user is registered, the user is not logged in
- **Parameters:** email, password
- **Actions:**
  1) check if email is unique
  2) check if password corresponds with the requirements
  3) create a new user in the system
  4) if all conditions are met - the user can log in, else - send an error message

Login

- **Actor:** user
- **Preconditions:** the user exists, the user not logged in
- **Postconditions :** the user is logged in
- **Parameters:** email, password
- **Actions:**
  1) check if email is recognized by the system
  2) check if password is correct
  3) change the user's status to "logged in"
  4) if all conditions are met - the user continues to the home screen, else - send an error message

Logout

- **Actor:** user
- **Preconditions:** user is logged in
- **Postconditions :** user is logged out
- **Parameters:** none
- **Actions:**
  1) check if user is logged in
  2) change user's status to "logged out"
  3) if all conditions are met - the user is logged out and returns to login page, else - send an error message

Delete User

- **Actor:** user
- **Preconditions:** user is logged in
- **Postconditions :** user deleted
- **Parameters:** password
- **Actions:**
  1) check if user is logged in
  2) check if password is the correct password
  3) delete user from system
  4) if all conditions are met - the user continues to login page, else - send an error message

Change Password

- **Actor:** user
- **Preconditions:** user is logged in
- **Postconditions :** user.password is changed
- **Parameters:** password, newPassword
- **Actions:**
  1) check if user logged in
  2) check if password is correct
  3) check if newPassword corresponds with the requirements
  4) change user.password to newPassword
  5) if all conditions are met - continue, else - send an error message

Change Email

- **Actor:** user
- **Preconditions:** user is logged in
- **Postconditions :** user.email is changed
- **Parameters:** password, newEmail
- **Actions:**
  1) check if user logged in
  2) check if password is correct
  3) check if newEmail is unique
  4) change user.email to newEmail
  5) if all conditions are met - continue, else - send an error message

Add Song To List (Local)

- **Actor:** user
- **Preconditions:** user is logged in, song is not in list
- **Postconditions :** song is in list
- **Parameters:** audio file, song name
- **Actions:**
  1) check if user is logged in
  2) check if song name is not in list
  3) add the audio file to the list as song name
  4) if all conditions are met - continue, else - send an error message

Add Song To List (YouTube)

- **Actor:** user
- **Preconditions:** user is logged in, song(url) is not in list
- **Postconditions :** song(url) is in list
- **Parameters:** URL, song name
- **Actions:**
  1) check if user is logged in
  2) check if URL leads to YouTube video
  3) check if song name not in list
  4) add the URL to the list as song name
  5) if all conditions are met - continue, else - send an error message

Delete Song From List

- **Actor:** user
- **Preconditions:** user is logged in, song in list
- **Postconditions :** song not in list
- **Parameters:** song name
- **Actions:**
  1) check if user is logged in
  2) check if song name in list
  3) delete song name from list
  4) if all conditions are met - continue, else - send an error message

Watch Song Details

- **Actor:** user
- **Preconditions:** user is logged in, song in list
- **Postconditions :** none
- **Parameters:** songName
- **Actions:**
  1) check if user logged in
  2) check if song in list
  3) if all conditions are met - show song's details on screen, else - send an error message

Record Audio

- **Actor:** user
- **Preconditions:** user is logged in
- **Postconditions :** recording saved in system
- **Parameters:** songName
- **Actions:**
  1) check if user logged in
  2) check if songName in list
  3) begin recording, notify user
  4) when user stops the recording:
     4.1) notify user
     4.2) save recording (audio file) in system
     4.3) compare to the original (given by songName)
  5) if all conditions are met - continue to performance summary view, else - send an error message

Watch Performance Summary

- **Actor:** user
- **Preconditions:** user is logged in, song in list, performance in list
- **Postconditions :** none
- **Parameters:** songName, performanceNumber
- **Actions:**
  1) check if user is logged in
  2) check if songName in list
  3) check if performanceNumber available
  4) if all conditions are met - show performance summary view, else - send an error message

2.3 Special Usage Considerations

2.3.1 Quiet Recording Environment

In order for the audio comparison to succeed, the recording environment must be quiet, so no external sounds will get to the recording and spoil the process, making the feedback given to the user unreliable.

2.3.2 Good Recording Tools

Although the application is targeted at many kinds of audiences, every user must have good recording tools. There is no need for professional tools - an average cell phone is enough, yet it must be able to record audio in good quality. The reason for this is the same as in 2.3.1, a bad recording (static noises, inaccurate sounds, etc.) will spoil the process, making the feedback unreliable.

2.3.3 Good Original Performance

The original performance is used by the application to grade the user's performance. Therefore, it must be a high quality recording with no background noises, whether it is a local file provided by the user or a YouTube video.

Chapter 3 - Functional Requirements

User Management

1. Account Operations

- 1.1 Create Account:
  - The system shall provide a registration form for users to enter personal information, including username, password, and email.
  - Upon submission, the system shall validate the information and create a new user account.
- 1.2 Delete Account:
  - Users shall have the option to delete their accounts through account settings.
  - The system shall prompt users to confirm the deletion and permanently remove their account and associated data.

2. Authentication

- 2.1 Login:
  - The system shall provide a login interface for users to enter their username and password.
  - Upon successful validation, the system shall grant access to user-specific functionalities.
- 2.2 Logout:
  - Users shall be able to log out from any page within the system.
  - The system shall terminate the user's session, requiring reauthentication for further access.

3. Profile Management

- 3.1 Modify Details:
  - Users shall have the ability to modify their personal details, including username, password, and email.
  - The system shall validate changes and update the user's profile accordingly.
- 3.2 Song List Operations:
  - 3.2.1 Add Songs:

        - Users can add songs to their selected or performed song list.
        - The system shall validate and update the user's song list.
    - 3.2.2 Remove Songs:
        - Users can remove songs from their selected or performed song list.
        - The system shall validate and update the user's song list.

Server Operations

1. Song Management

- 1.1 Song List Storage:
    - The server shall maintain a separate song list for each user, storing selected and performed songs.
    - Song lists shall be associated with respective user accounts.
- 1.2 Global YouTube List Update:
    - The server shall periodically update a global list of songs from YouTube.
    - The global list shall be accessible to all users for song selection.
- 1.3 Song Operations:
    - 1.3.1 Add Songs:
        - The server shall support the addition of new songs to the global list.
        - Song information, including metadata and source links, shall be stored.
    - 1.3.2 Remove Songs:
        - The server shall allow the removal of songs from the global list.
        - Removal actions shall be reflected in user song lists.

2. Audio Comparison and Feedback

- 2.1 Accept Audio Files:
    - The server shall receive audio files from users for performance comparison.
    - File formats and size limitations shall be specified.
- 2.2 Perform Comparison:

        - The server shall utilize algorithms to compare user performances with the original audio.
        - Timing, pitch accuracy, and dynamics shall be considered in the comparison.
    - 2.3 Send Ratings and Feedback:
        - The server shall generate ratings and constructive feedback based on the comparison.
        - Feedback shall be sent to users within a reasonable timeframe.

3. Performance Logging

- 3.1 Store Ratings:
    - The server shall store performance ratings for each user and song.
    - Ratings shall include an overall score and specific metrics.
- 3.2 Maintain Accuracy List:
    - The server shall maintain a list of accuracy metrics for each user and song.
    - Accuracy metrics shall provide detailed insights into performance areas.

Chapter 4 - Non-Functional Requirements

4.1 Implementation Constraints

Performance:

- Response Time: The system shall provide a response time of less than 2 seconds for user interactions.
- Throughput: The system should support concurrent usage by at least 1000 users without significant degradation in performance.
- Speed:
  The application should support vocal analysis up to 100ms. An existing audio file analysis currently takes about 1.5-2 minutes for a 1-minute song (not including API transmission time, runtime will be shorter in the future).

Reliability:

- Availability: The system shall have an availability of 99.9% during normal operating hours.
- Fault Tolerance: The system should be resilient to potential failures, and critical functionalities should have backup mechanisms.

Security:

- Data Encryption: All user data, especially login credentials and personal information, should be encrypted during transmission and storage.
- Access Control: The system shall implement role-based access control to ensure that users can only access the functionalities relevant to their roles.

Portability:

- The system should be deployable on both iOS and Android platforms to ensure accessibility on mobile devices.
- The system should support text in English, with an option to extend to other languages in future versions.
- Unicode or other internationalization standards should be employed to handle different character sets.

Usability:

Entire Application:

- The application must be simple to manage by the common user.
- The application will have a user-friendly GUI with a few screens.

Business logic:

- The users may be a professional singer/musician who's trying to be more accurate or a trainee that wants to learn how to sing/play in tune.
- The application must provide clear corrections so the user will be able to perform better next time.

Availability:

- The application must be available and functional at any time. The only constraints are that there must be an internet connection and a working microphone.

4.2 Platform Constraints

4.2.1 SE Project Constraints

- Demo currently unavailable and will be in the POC stage.
- The app is interactive - the user will record input from his device's mic and will receive feedback accordingly. We will receive authorization from the user to access his mic. We also support premade WAV files.
- We will develop our system with our PCs, using Pycharm, VS Code & Express.
- There are no platform constraints.

4.3 Special Restrictions & Limitations

We may add an option to add a new song that is not saved in our system (for example if I record myself and later I will want to compare myself to the recorded version, or if a user will want to practice a new song that is not saved in our archive.)

Addition of recordings (wav files) will be optional only if the recording is acapella or only 1 instrument is played.

If a user tries to add wav files that won't fit in this category, a message will be displayed to him.

Chapter 5 - Risk Assessment & Proof of Concept Plan

There are 3 major risks in Out Of Tune that may cause us to change the project's nature. or may even fail it:

1) The audio analysis risk:
   The audio analysis must be accurate. This phase is critical for the proper behavior of the system, otherwise it will not be able to compare the user's performance to the original one. Our plan is to try several algorithms and understand their limitations, for example - some voices may not be recognized well, how background noises may affect the analysis and more. After that phase proved possible, this critical risk will be removed.
2) The audio comparison risk:
   Probably the greatest risk in the project, if the audio comparison fails the project will fail for certain. This phase is dangerous due to the many variables that we must consider - time shifts, different voices, different rhythms, different file lengths and many more. Our hope is to solve this problem with existing algorithms that proved to be good algorithms. After this major problem is dealt with, the risk will greatly decrease.
3) Waiting times:
   Due to the project's current structure, we may encounter long waiting times. On the server's side, the audio analysis and the audio comparison are both costly and will take time. The solution can be a powerful server, distributed calculations and sacrificing accuracy for shorter waiting times. On the client's side, the communication between the client and the server may take long, especially when the client sends the audio file to the server. The solution to that problem may be working with efficient audio files format or changing the architecture of the program to a local app instead of client-server. This risk is mostly concerning the user experience.