VocaFree - Milestone 1

An Luu (301574874)

Parsa Ghaderi (301623337)

Jake Choi (301552446)

Caiden Merklin (301608302)

https://github.com/CMPT-276-FALL-2024/project-07-hills

Project overview

Karaoke can be a fantastic way for friends and family to bond and have fun together, but it often comes with significant limitations that make it less accessible for many. Several barriers hinder participation, including busy schedules that leave little time for leisure activities, financial difficulties, as the costs associated with going to karaoke venues or purchasing home karaoke systems can be prohibitive, and the needs of individuals with physical or mental conditions, who may find it challenging to travel to venues. Additionally, teams in work settings may struggle to engage in team-bonding activities like karaoke due to tight schedules. Singing enthusiasts often feel limited by the narrow selection of songs available in traditional karaoke catalogs, while language learners also face challenges, as understanding and pronouncing foreign lyrics can be difficult without proper support.

Hence, our group came up with the idea of VocaFree, a free-to-use web application that uses YouTube song video links to create an authentic karaoke experience by separating vocals from the music of any song, and playing just the instrumental track, allowing users to sing any songs of their choice without the original song's vocals. VocaFree addresses challenges by offering a free, convenient, and accessible karaoke solution that can be enjoyed from home, regardless of time constraints, financial constraints, or travel limitations. Moreover, for teams and workplaces, VocaFree provides an engaging way to boost team spirit without taking up too much time or requiring employees to leave the office. For singing enthusiasts, the app opens up a wider range of song choices by allowing users to convert any YouTube song videos into karaoke tracks, overcoming the limitations of karaoke song catalogs that only include well-known songs. Language learners and cultural enthusiasts also benefit, as VocaFree offers translation and romanization/phonetic assistance for foreign language songs, helping users practice pronunciation of those languages while enjoying a fun activity.

VocaFree is versatile and accessible, making it an ideal solution for various potential users, including:

- **Individuals with Busy Schedules:** Those who may not have time to go out can easily enjoy karaoke at the convenience of their home.

- **Individuals Facing Financial Difficulties:** Karaoke can often be an expensive activity, with both the cost of going out to karaoke venues and the high price of home karaoke systems being barriers. VocaFree offers a cost-free solution, allowing users to enjoy a full karaoke experience from the comfort of their home without needing to invest in expensive equipment.
- **Individuals with Mental or Physical Conditions:** For those who struggle to leave their homes, VocaFree provides an inclusive activity that can be enjoyed in a safe environment.
- **Teams and Work Settings:** With tight schedules and heavy workloads, employees often struggle to find time for team-building activities. VocaFree allows teams to engage in karaoke sessions from their offices or homes, enhancing team spirit without requiring significant time away from their responsibilities.
- **Singing Enthusiasts Who Want Access to a Wider Range of Songs**: For those who love singing and want to challenge themselves, VocaFree provides a solution when the songs they wish to sing aren't available in the catalogs of traditional karaoke places or apps. This allows users to sing along to any song by extracting the instrumental only from the music of YouTube song video links, making it ideal for both casual singers and those serious about honing their vocal skills.
- **Language Learners:** For songs in foreign languages, the app provides English pronunciations (romanization and phonetics) of the lyrics, enabling users to sing along even if they don't understand the original language, making it easier for language learners to practice speaking in that language.

API description

Genius API: The Genius API allows access to a database of song lyrics, artist information, and song metadata from the Genius platform. It allows you to search songs by title or lyrics, retrieve artist profiles, and get links to Genius pages where full lyrics can be found and web scraped. This API is popular for building music-related applications, helping users discover songs and learn about artists.

Feature 1 - Get Lyrics: Retrieve the lyrics for a specific song, allowing users to sing along accurately.

Feature 2 - Get Song Based on Lyrics: Search for songs by providing a snippet of lyrics, helping users find the correct song they want to sing.

Feature 3 - Get Artist Information and metadata: Fetch detailed information about artists and the song, including their biography, song metadata.

OpenAI API: This is the most commonly used LLM (large language model) api, developed by OpenAI and used for ChatGPT. It can be used to generate text from a prompt, and can return almost any kind of response.

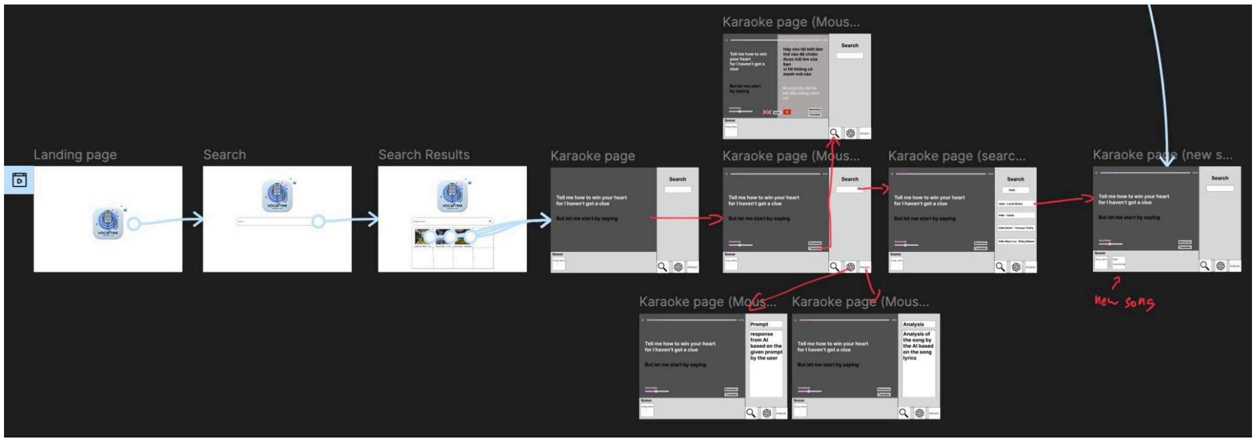
Feature 1 - Translate Lyrics: Use the API to translate song lyrics into different languages, allowing users to understand the meaning of non-native songs in real time, allowing them to improve on their foreign language skills.

Feature 2 - Romanization/Phonetics: Use an additional prompt to convert the characters into their phonetic (roman) equivalents so it can be easily read by someone used to Roman script. This allows someone the opportunity to sing along in a new language even if they can’t read the characters.

Feature 3 - Basic Lyric Analysis: Analyze song lyrics to provide insights or themes, enhancing users' understanding and engagement with the music.

Mid-fidelity storyboard

(Demo video in the Repository).



SDLC Model

Our team will be using/uses the following SDLC processes for our group project:

- Agile
- Kanban
- Scrum

All of our processes will be among various **agile** frameworks. First, we are using a **Kanban** board to break down our different tasks, assign them to each other, identify blockers, send them for testing, etc. We are currently utilizing the built-in "Project Board" in Notion, but we may convert this to GitHub issues, or Jira in the future if necessary. Importantly we will integrate the project into the "**Scrum**" standard, our team has chosen this standard as it is the most frequently used process for tech companies, and further, the one that many of us have experience in using. Scrum is a nice way to manage our productivity and what process we are in during the development, which will be crucial given our short time frame. It will allow us to better understand our project through "Sprints" in that we can produce a certain amount of features in a certain timeframe, along with various meetings to discuss the **Kanban** board.

WBS and Project Schedule

WBS in Git Repo or from this link.

Project Schedule:

Milestone 0: Project Setup and Proposal (Due Friday, Oct 18)

1. **Requirement Gathering** (Oct 5 - Oct 10)
   - **Oct 5**: Initial brainstorming session to outline main features and user needs.
   - **Oct 7**: Finalize a draft of primary features and expected functionalities.
   - **Oct 10**: Review and document the finalized requirements.
2. **Project Proposal Report** (Oct 11 - Oct 18)
   - **Oct 11**: Outline project goals, API choices, and initial scope.
   - **Oct 13**: Write a draft for the proposal report, including details on features and functionalities.
   - **Oct 16**: Review and refine the report.
   - **Oct 18**: Submit the final project proposal report.

Milestone 1: Project Planning (Due Friday, Nov 1)

1. **Technical Specifications** (Oct 19 - Oct 21)
   - **Oct 19**: Identify and document front-end, back-end, and API choices.
   - **Oct 20**: Determine technology requirements for integration.
   - **Oct 21**: Finalize and document technical specifications.
2. **Choose Tech Stack** (Oct 21 - Oct 22)
   - **Oct 21**: Research potential tech stacks and finalize decisions.
   - **Oct 22**: Confirm tech stack choices and begin setting up development environments.

3. **Learn Tech Stack** (Oct 22 - Oct 30)
   - **Oct 22 - Oct 25**: Each member completes tutorials and training for chosen tech stack components (React, APIs, UVR, Flask, FastAPI).
   - **Oct 28**: Group session to discuss any issues and ensure everyone understands each component.
   - **Oct 30**: Conduct a quick quiz or discussion to confirm proficiency.
4. **Determine Project Scope** (Oct 23 - Oct 24)
   - **Oct 23**: Outline core functionality and prioritize tasks for initial release.
   - **Oct 24**: Finalize scope, clarifying primary tasks and optional future enhancements.
5. **Determine Test Cases** (Oct 24 - Oct 25)
   - **Oct 24**: Brainstorm test scenarios, covering UI, data flow, and API functionality.
   - **Oct 25**: Document initial test cases.
6. **Complete Technical Specifications** (Oct 25 - Oct 26)
   - **Oct 25**: Draft the complete technical document with finalized tech stack, system architecture, and integration details.
   - **Oct 26**: Review and finalize technical specifications.
7. **Document Requirements** (Oct 26 - Oct 27)
   - **Oct 26**: Outline specific requirements for features, integrations, and interactions.
   - **Oct 27**: Finalize and document detailed requirements.
8. **Determine Git Flow** (Oct 27)
   - **Oct 27**: Define Git workflow for branch naming, commits, and pull requests.
9. **Determine Team Communications** (Oct 28)
   - **Oct 28**: Set up and confirm all communication channels (Notion, Git, Discord).
10. **Develop Design Documents** (Oct 28 - Oct 29)
    - **Oct 28**: Draft wireframes and initial layout ideas.
    - **Oct 29**: Finalize the design document with primary UI elements and guidelines.
11. **Develop Mid-Fidelity Storyboard** (Oct 29 - Oct 30)
    - **Oct 29**: Begin creating mid-fidelity storyboard using Figma.
    - **Oct 30**: Complete storyboard with clickable elements for user flow.
12. **Develop MVC Model** (Oct 30 - Oct 31)
    - **Oct 30**: Outline MVC structure for the application.
    - **Oct 31**: Document the model with roles and interactions.
13. **Dataflow Diagrams** (Oct 31 - Nov 1)
    - **Oct 31**: Create DFDs illustrating data flow between components and APIs.

- **Nov 1**: Finalize DFDs and incorporate them into the planning report.

14. **Risk Assessment** (Oct 29 - Nov 1)
    - **Oct 29**: Identify and document potential project risks.
    - **Nov 1**: Add mitigation strategies for each identified risk.
15. **Determine Project Schedule and Deadlines** (Oct 31 - Nov 1)
    - **Oct 31**: Outline project timeline based on task priorities and dependencies.
    - **Nov 1**: Complete and add schedule to project planning report.

Milestone 1.5: Project Check-In (Due Week of Nov 11-15)

1. **Complete Design Requirements** (Nov 1 - Nov 5)
    - **Nov 1 - Nov 3**: Finalize design elements and styles.
    - **Nov 4**: Review mid-fidelity prototype and storyboard.
    - **Nov 5**: Complete design requirements and integrate into project planning.
2. **Submit Progress Report** (Nov 6 - Nov 11)
    - **Nov 6 - Nov 8**: Draft check-in report covering completed tasks, changes, and upcoming milestones.
    - **Nov 11**: Submit the check-in report.
3. **Meeting with TA** (Week of Nov 11-15)
    - **Nov 11 - Nov 15**: Meet with TA for project feedback and adjust plans as needed.

Milestone 2: Project Delivery (Due Week of Nov 26 - Dec 3)

Implementation

1. **GUI Implementation**
    - **Setup React** (Nov 11 - Nov 12): Initialize and set up React environment.
    - **Code Landing Page** (Nov 13): Develop and style the homepage layout.
    - **Code Search Bar** (Nov 14): Implement search bar functionality.
    - **Display Logo** (Nov 14): Integrate and display logo in the header/main page.
    - **Finish GUI Implementation** (Nov 15): Finalize front-end UI components.
2. **Feature Development**

    - **Code Suggestions Page** (Nov 16 - Nov 17): Implement suggestions feature.
    - **Code Karaoke Page** (Nov 18 - Nov 20): Develop main karaoke feature with lyrics.
    - **Integrate UVR API** (Nov 21): Set up UVR API integration.
    - **Integrate Genius API** (Nov 22): Set up Genius API integration for lyrics.
    - **Integrate LLM API** (Nov 23): Configure and test LLM API for content generation.
3. **Backend Setup**
    - **Setup FastAPI Backend** (Nov 24): Initialize FastAPI for backend processing.
    - **Setup Node.js Backend** (Nov 25): Configure Node.js server as needed.
    - **Complete Implementation** (Nov 26): Finalize all code and test integrations.

Testing

1. **Test GUI Functionality** (Nov 27 - Nov 28): UAT and regression testing on all front-end components.
2. **Test UVR Functionality** (Nov 28): Confirm UVR integration.
3. **Test Genius Integration** (Nov 29): Verify lyrics retrieval and display.
4. **Test LLM Integration** (Nov 30): Test dynamic content generation.
5. **Test Server Functionality** (Dec 1): Ensure back-end API responses and data handling work smoothly.
6. **Complete Testing** (Dec 2): Review test cases and make necessary fixes.

Deployment

1. **Configure GitHub Actions** (Dec 2): Set up CI/CD pipeline for automated testing and deployment.
2. **Finish Deployment** (Dec 3): Deploy the app and confirm it's ready for presentation.

Documentation

1. **Edit README** (Nov 30 - Dec 1): Update README with setup, usage, and project details.
2. **Write Project Report** (Dec 2 - Dec 3): Summarize project goals, features, and development insights.
3. **Complete All Documentation** (Dec 3): Ensure all documents are finalized and ready for submission.

Risk Assessment

Low-risk:

1. Reliability of UVR:

Since we are using an external application to handle our audio separation, we need to be careful of its reliability. There is a risk of the UVR not always working as expected while users try our app.

Mitigation plan: Test its reliability to the best of our power (as many testing scenarios as possible) and research for a secondary application for backup.

2. Reliability of Genius API:

We are using this API for three of our core features. We have to make sure it is responsive and reliable. There is a risk of Genius API not always working as expected while users try our app, and therefore obstructing our features related to it.

Mitigation plan: Test Genius API and familiarize ourselves with Spotify's API and have it ready as a backup.

3. Reliability of OpenAI API:

This API is also being used for three of our core features. Its reliability and responsiveness should be tested. There is a risk of OpenAI API not always working as expected while users try our app, and therefore obstructing our features related to it.

Mitigation plan: Test OpenAI API and have Llama API as backup, and understand how it works if we need to switch.

4. Two or more songs that share lyric parts.

The case where the user inputs a lyrics snippet, excepting one specific song, but more songs have the same snippet.

Mitigation plan: Inform users that there is a chance of returning the wrong song and ask them to give larger snippets so there is less chance of a mistake. Since we do not have a database, we might not be able to show users different options hence we might return the first song that has the snippet. This might increase error so we have to make sure we communicate with the user (as mentioned above) clearly.

5. Two or more songs share the same name.

The case where there is more than one song with the same name.

Mitigation plan: Inform users of the possibility that the wrong song is returned, and ask them to give as much info on the song as possible(e.g. Artist, year, album) so there is less chance of mistakes. Since we do not have a database, we might not be able to show users different options hence we might return the first song that has the title. This might increase error so we have to make sure we communicate with the user (as mentioned above) clearly.

Mid-risk:

1. No YouTube video for the song, or the video is not of good quality.

It will be troublesome if the YouTube video does not exist altogether or if it is not good enough for our purposes. This will go against our promise of having a wide selection of songs.

Mitigation plan: We can upload our own YouTube videos for popular songs that do not have any decent YouTube videos attached to them. It is important to remember that the video does not have to be officially uploaded by that artist, but it can be any video that a user has uploaded. So while it is not likely that this will arise, we can inform our users of the possibility.

2. Speed of the vocal separation.

While we advertise VocaFree to those with busy schedules, we need to make sure that the separation happens promptly. Even if UVR is reliable, it is important how fast it works.

Mitigation plan: Test UVR's speed and have research for a quicker app in case it is slow. Research clever ways to handle the waiting time (e.g. having an interactive or fun loading screen).

3. Genius API does not know the song.

We are advertising our app for having a large selection, so it'll be quite a shame if a user asks for a song that does not exist in Genius's library.

Mitigation plan: Test Genius library's diversity. Have SpotifyAPI ready if it is not diverse enough

High-risk:

1. Song title is different from the YouTube title.

We are relying on YouTube videos for our app. However, there is an issue where the song title is different from the YouTube video title, which will be troublesome.

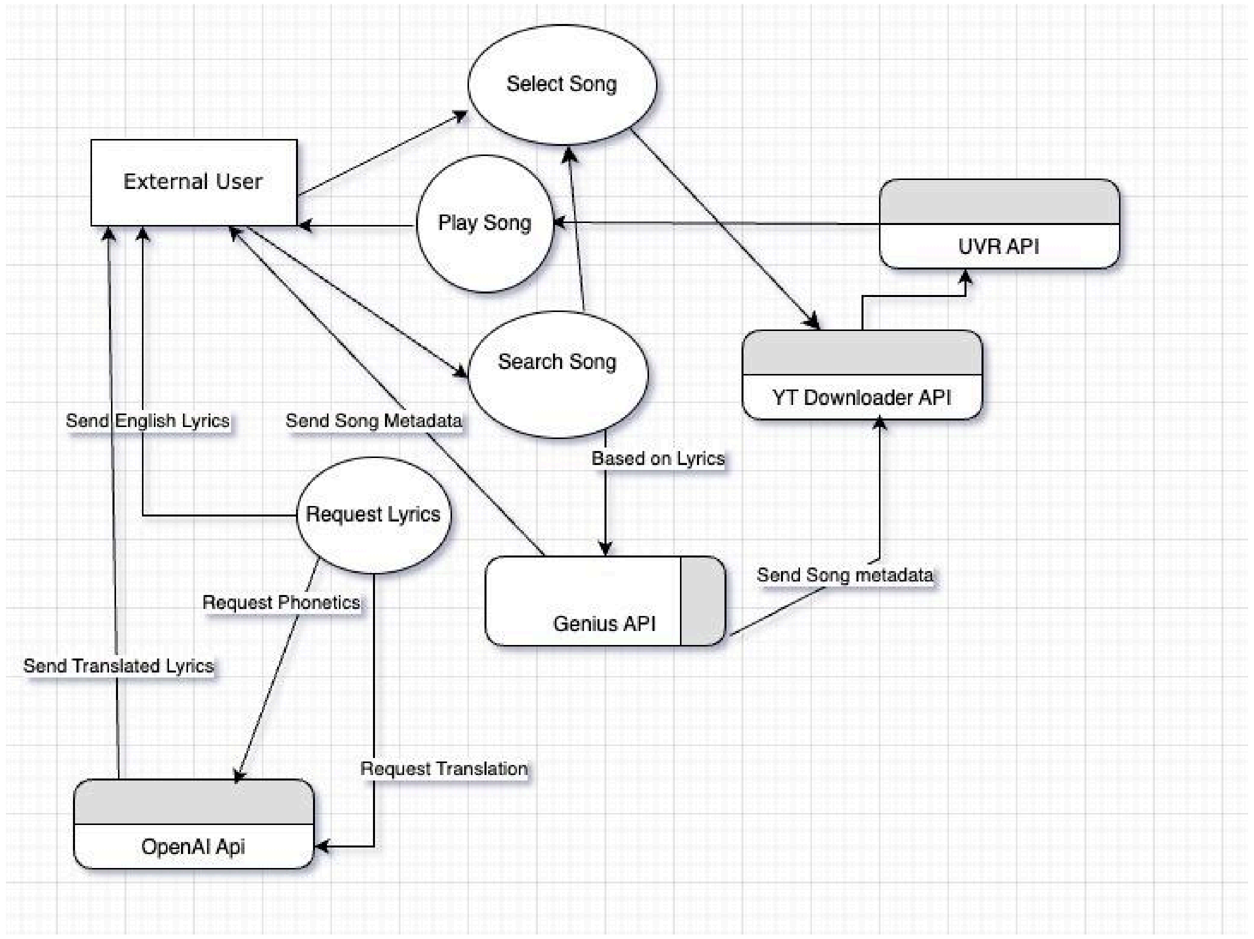
Mitigation plan: Rely on OpenAI API or Llama API for finding the Youtube video based on a song title.

2. Storing song suggestions with no database.

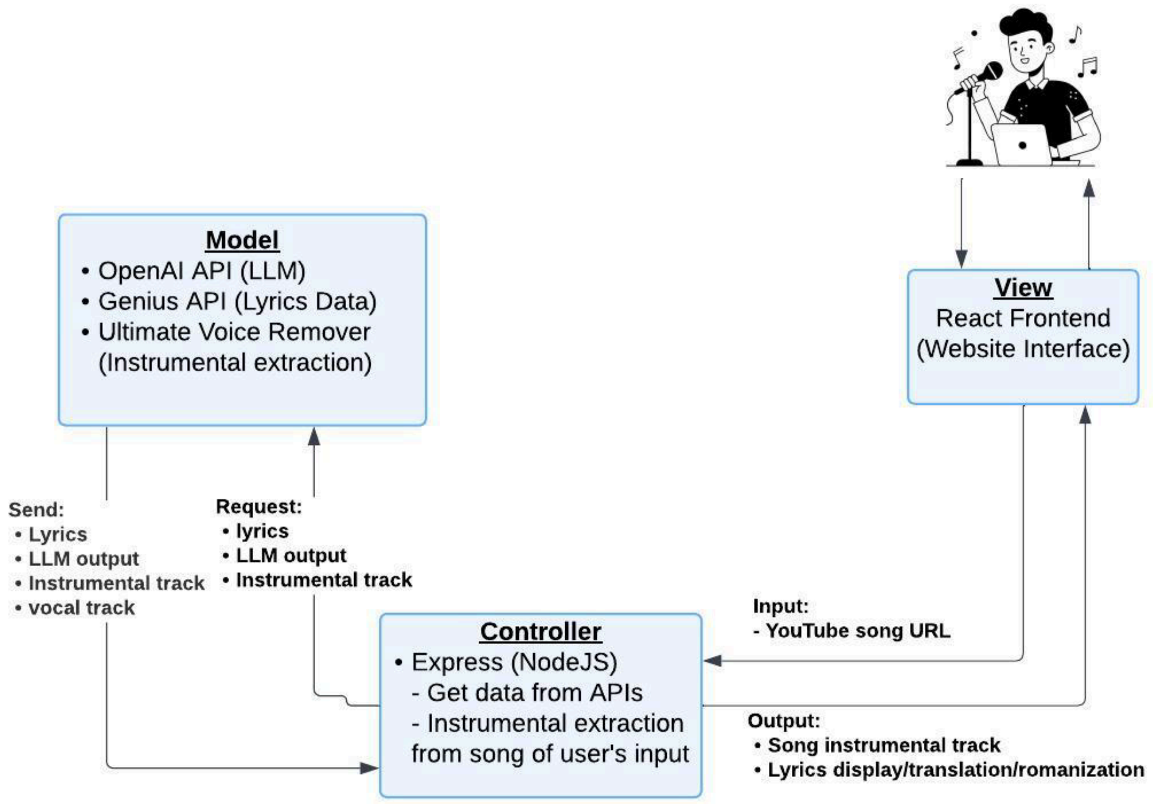
There is a risk that we won't be able to display a list of songs since we are not using a database. This will affect many features.

Mitigation plan: Experiment with OpenAI API for displaying a list of songs. In the last case scenario, remove this feature or add a database.

Data Flow Diagram



MVC Model



Model
• OpenAI API (LLM)
• Genius API (Lyrics Data)
• Ultimate Voice Remover (Instrumental extraction)
View
React Frontend (Website Interface)
Send:
• Lyrics
• LLM output
• Instrumental track
• vocal track
Request:
• lyrics
• LLM output
• Instrumental track
Input:
- YouTube song URL
Controller
• Express (NodeJS)
- Get data from APIs
- Instrumental extraction from song of user's input
Output:
• Song instrumental track
• Lyrics display/translation/romanization