VocaFree - Milestone 2

An Luu (301574874)

Parsa Ghaderi (301623337)

Jake Choi (301552446)

Caiden Merklin (301608302)

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

Video: https://youtu.be/wSqGngFJXjM?si=vV-mKzMgm1Bve_SU

VocaFree website: voca-free1.vercel.app

1. Analysis of Project’s Success

The VocaFree application successfully addresses the needs of users with busy schedules, financial difficulties, and physical or mental conditions, as highlighted by both its design and user feedback. By allowing users to enjoy karaoke from the comfort of their homes, VocaFree eliminates the logistical and financial barriers associated with traditional karaoke venues or expensive home systems. The application’s minimalist and straightforward design has received positive feedback for being easy to navigate and enjoyable to use, making it particularly accessible to individuals who might find more complex systems challenging.

Users praised the intuitive functionality of the application, particularly the ability to search for songs, play them, and view lyrics seamlessly. One tester noted, “It is straightforward. We search songs, we play them, and look at the lyrics. It’s not confusing at all.” This simplicity makes VocaFree a suitable option for users who have limited time and need a quick, hassle-free solution to enjoy karaoke. The ability to interact with lyrics by clicking on them adds a unique and engaging feature, enhancing the overall experience.
While the feedback was largely positive, users identified areas for improvement, such as the intuitiveness of the queue and progress bar. Some users found the queue unclear, noting, “I didn’t realize there’s a queue and that it was loading.” Suggestions included making the queue more intuitive and providing an option for immediate playback. Addressing these issues in future updates will further improve the app’s usability and better serve the needs of users with limited time or those seeking instant access to karaoke sessions.

Overall, VocaFree demonstrates significant success in meeting its user needs by providing an accessible, enjoyable, and engaging karaoke experience. The application has been well-received by users for its simplicity and innovative features, while also highlighting opportunities for refinement to enhance usability and user satisfaction further.

2. Analysis of SDLC Model

- Analysis of the project’s SDLC model and how it was used in the project
    - Include any changes made to the model during the project

The VocaFree project adopted an Agile Software Development Life Cycle (SDLC) model, incorporating Scrum and Kanban frameworks to manage development tasks and ensure continuous delivery of features. This approach allowed the team to iteratively design, implement, and refine the application while accommodating a flexible workflow. Each sprint focused on a specific milestone, enabling the team to meet deadlines effectively while maintaining a structured yet adaptable process. Notably, no significant changes were made to the SDLC model during the project.

How the SDLC Model Was Used

The team began with **Sprint 0**, a one-week phase dedicated to brainstorming and research. During this initial sprint, the project scope was defined, APIs were evaluated, and core features were identified. This sprint culminated in the creation of the project proposal, providing a solid foundation for subsequent phases.

In **Sprint 1**, the team focused on the planning and documentation required for Milestone 1. Key activities included finalizing the APIs, defining features, creating low- and mid-fidelity storyboards, and developing the Work Breakdown Structure (WBS) along with a project schedule. Data Flow Diagrams were also constructed to clarify the interaction between system components, setting the stage for implementation.

Sprint 2 marked the development of **Prototype 1**, where both frontend and backend functionalities were introduced. The frontend included basic song search functionality and a music player interface for playing instrumental tracks. The backend was implemented using FastAPI, enabling endpoints to handle music data, vocal separation via UVR, and Spotify API calls for metadata retrieval. Lyrics integration was also introduced during this sprint to provide timestamped lyrics for users.

Sprint 3 focused on enhancing functionality in **Prototype 2**, building upon the foundational features. The frontend saw the addition of song queue management, enabling users to add songs to a queue and process them automatically. A loading bar was implemented to improve user experience by showing progress during song processing. Corresponding backend task processing endpoints were developed to support these features.

In the final phase, **Sprint 4**, the team concentrated on deployment and report writing for Milestone 2. The frontend was hosted on a web platform, and the FastAPI server was deployed on a team member's personal computer to facilitate backend functionality. Integration between the frontend and backend was finalized during this phase, ensuring a cohesive user experience.

Reflection on the SDLC Model

The combination of Agile, Scrum, and Kanban proved to be highly effective for this project. By dividing the development into distinct sprints, the team was able to prioritize tasks, address dependencies, and ensure iterative progress. The use of a Kanban board for task management and Scrum's sprint cycles allowed for transparent tracking of tasks and milestones.
Despite the challenges associated with tight deadlines, the team adhered to the planned SDLC model without making any changes. The model's flexibility and focus on incremental delivery ensured that all project requirements were met within the allocated timeframe. Overall, the SDLC model was instrumental in the successful completion of the VocaFree project, providing a structured yet adaptable framework for development.

3. Features Implemented for Each API

The VocaFree project incorporates various APIs and tools to deliver its features. Each API was carefully selected or replaced during development to optimize functionality, efficiency, and integration. Below is a detailed description of the features implemented for each API, including changes made during the project.

Spotify API

The Spotify API was used to search for Spotify tracks and retrieve their metadata, including the title, artist, album art, Spotify ID, and Spotify URL. This data was essential for the karaoke application as it facilitated song identification and processing.

Features Implemented:

- Track Search: Allows users to search for specific songs based on song title, artist name, or lyrics on Spotify, providing quick access to their details.

Spotify-Lyrics-API

The Spotify-Lyrics-API was added during development to meet the need for timestamped song lyrics. It offers RESTful endpoints for retrieving lyrics synchronized with Spotify tracks.
Features Implemented:

- Lyrics Display: Users can view synchronized lyrics for songs, enhancing engagement during karaoke sessions.
- Timestamped Lyrics: Lyrics are synced to playback, ensuring users can follow along accurately.

SpotDL (Spotify Downloader)

SpotDL was incorporated as a Python package to download songs and provide instrumental tracks for karaoke sessions. It replaced YouTube-DL to create a more cohesive integration with Spotify.
Features Implemented:

- Music Downloads: Automates song downloads based on Spotify links provided by users.
- Instrumental Creation: Serves as the foundation for processing downloaded tracks into instrumental versions.

Ultimate Vocal Remover (UVR)

UVR was used to separate vocals from instrumentals, addressing the core functionality of the application. A Singleton class was developed to handle this process, ensuring consistency and reusability across the backend.
Features Implemented:

- Instrumental and Vocal Separation: Automatically processes every song added to the queue for karaoke preparation.
- Singleton Class Integration: A centralized system for managing audio separation tasks within the FastAPI backend.

Changes Made to Features During Development
Genius API to Spotify API:

- Reason: Spotify API provided a more seamless integration with the overall workflow, simplifying song search and metadata retrieval.

YouTube-DL to SpotDL:

- Reason: SpotDL streamlined the process of searching tracks using Spotify API and downloading them efficiently.

Removal of Romanization and Translation:

- Reason: These features were deprioritized due to time constraints but remain potential future enhancements.

4. CI/CD Pipeline

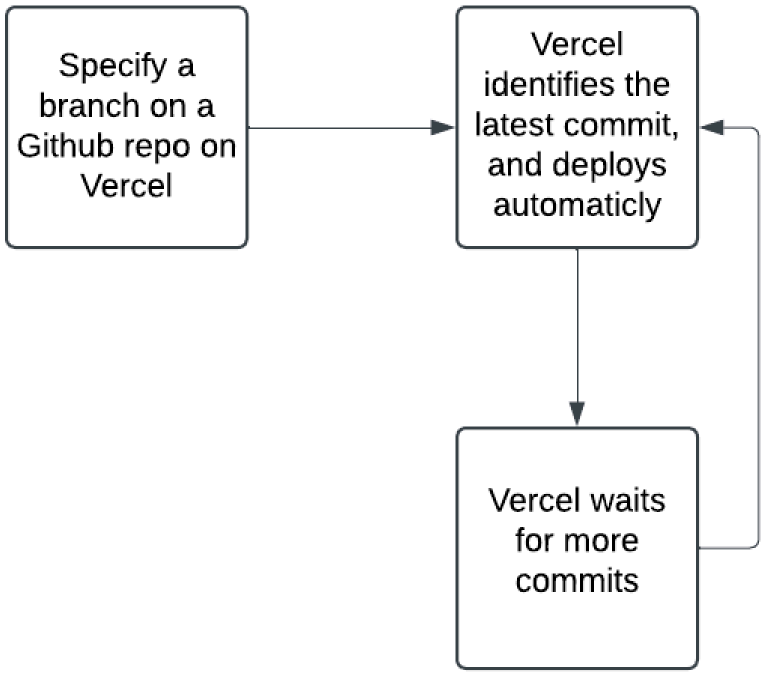
Overview:

- We utilized Vercel's quick and user-friendly built-in Continuous Deployment (CD) capabilities.

Components:

- We configured Vercel to monitor a specific branch in a GitHub repository. With every commit to that branch, Vercel automatically deployed the latest changes. Each deployment generated a detailed build log, which provided insights into the process and highlighted any errors encountered.

Pipeline Workflow:


Specify a branch on a Github repo on Vercel
Vercel identifies the latest commit, and deploys automaticly
Vercel waits for more commits

5. Testing Strategy

Testing Approach

Our testing approach included a combination of unit tests, integration tests, and user testing to ensure a robust and user-friendly application.

Unit Tests with Jest:

We used Jest and `@testing-library/react` to write unit tests for individual components like `SearchBar` and `PlayButton`. These tests focused on validating specific functionalities, such as rendering elements correctly, handling user input, and triggering callback functions.

Integration Tests with Jest:

Integration tests were implemented to ensure components interacted as expected. For example, we verified that submitting a search query successfully triggered the appropriate actions, such as updating the UI and sending the necessary API requests.

In-Class User Testing:

During a class session, we had peers interact with the application. This provided direct feedback on usability, highlighting areas for improvement, such as button labeling and input field clarity.

External User Testing:

We shared the deployed application link with a friend and asked them to explore its features. Their feedback, which included suggestions for UI improvements and functionality enhancements, was instrumental in refining the final product.

Implementation

Tools and Frameworks:

Jest with `@testing-library/react` was used for unit and integration testing. Manual user testing sessions were conducted for peer feedback.

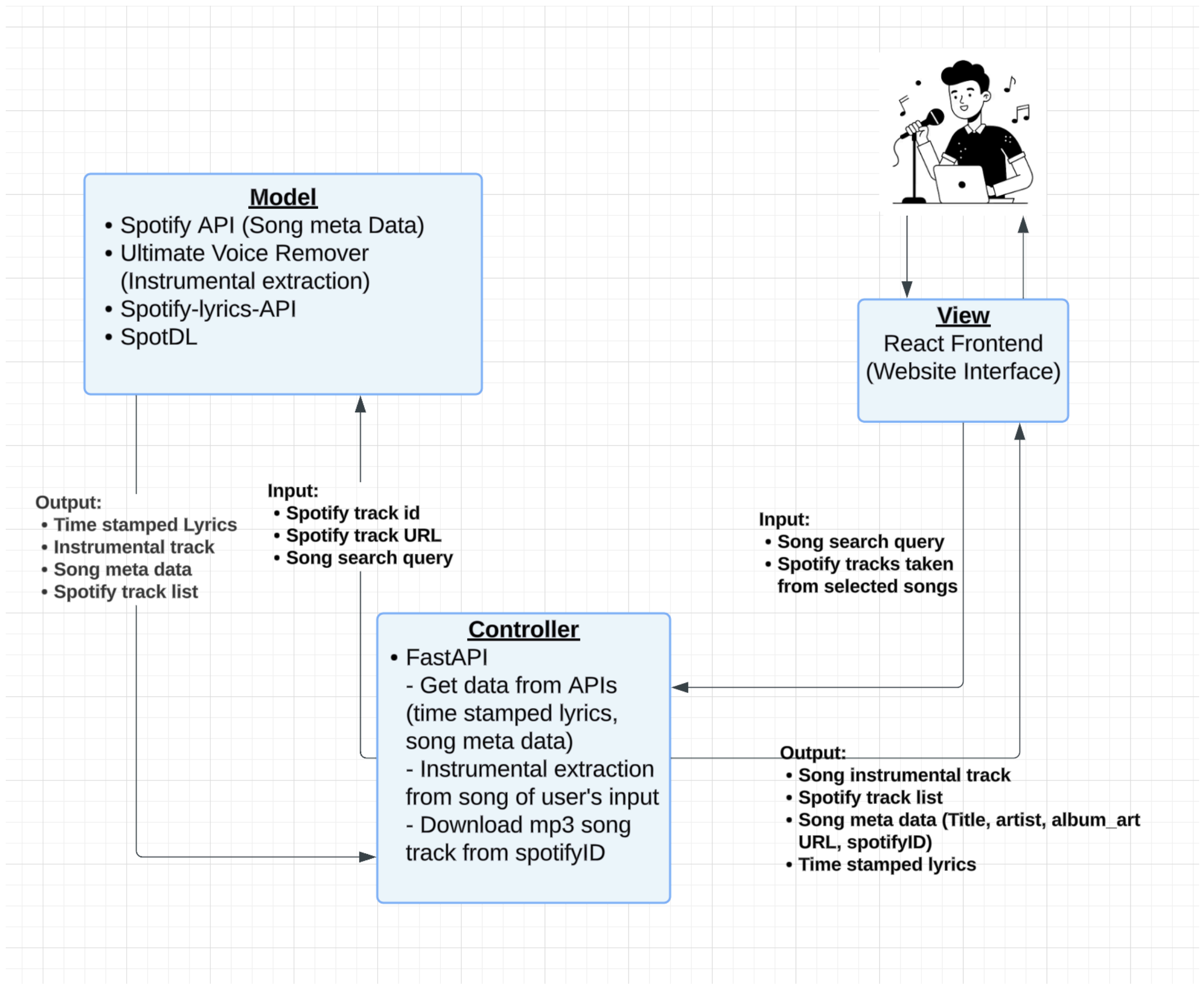
Coverage and Findings:

Tests covered ~85% of the codebase, focusing on critical components like `SearchBar` and `PlayButton`. We did not use it on the backend as the backend was not a required bit of this project. Key bugs were fixed, such as API request failures and input validation issues.

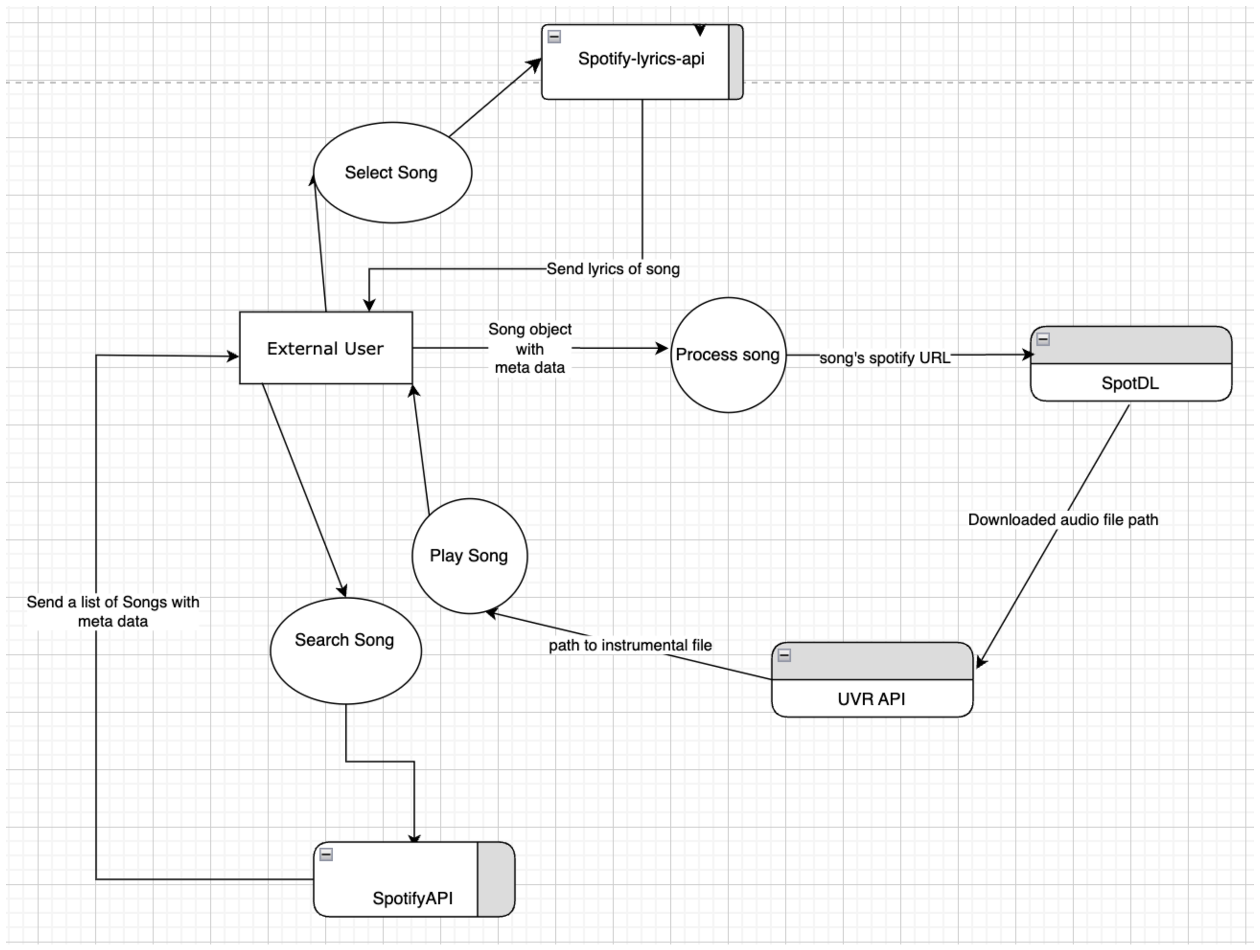
- User Testing:
  - Summary of the peer testing session and its results.

6. Project Architecture

VocaFree MVC


Model
• Spotify API (Song meta Data)
• Ultimate Voice Remover (Instrumental extraction)
• Spotify-lyrics-API
• SpotDL
View
React Frontend (Website Interface)
Output:
• Time stamped Lyrics
• Instrumental track
• Song meta data
• Spotify track list
Input:
• Spotify track id
• Spotify track URL
• Song search query
Input:
• Song search query
• Spotify tracks taken from selected songs
Controller
• FastAPI
- Get data from APIs (time stamped lyrics, song meta data)
- Instrumental extraction from song of user's input
- Download mp3 song track from spotifyID
Output:
• Song instrumental track
• Spotify track list
• Song meta data (Title, artist, album_art URL, spotifyID)
• Time stamped lyrics

VocaFree DFD



7. Known Bugs and Issues

Bug Description	Steps to reproduce	Severity Level	Link	Status
Deleting a song from the queue while it has not been finished downloading and processing, can cause other songs inside the queue to duplicate	1. Add multiple songs into the queue. 2. Delete songs that are still processing. There is a chance of a song inside the queue duplicating.	HIGH	Link	RESOLVED (Fixed the bug by not letting the user delete songs while it is still processing)
Attempting to play a song before it is finished downloading and processing the separation of vocals will prevent the song from playing. (It is not really a bug, but to users who don't know how the system works might think the web application is faulty)	1. Add a song to the queue. 2. Try to play that song immediately by clicking on the play button. The song won't play because the user does not give the web application time to process the song.	MODERATE	Link	RESOLVED (Fixed the problem by displaying a progress bar that shows the current status of the newly added song in the queue, with the following states: Initializing, Downloading, Separating, Complete, so that the user knows that it needs some time to prepare the song)
Scrubbing or clicking on the lyrics does not work for Chromium based browsers. Instead, it causes the song to replay from the start.	1. Access the web application through a chromium based browser. 2. Add a song to the queue. 3. Click the play button and let the song play for a few seconds. 4. Click on the scrub bar or the lyrics. It will cause the song to replay instead of going to the desired part of the song.	MODERATE (works properly on Safari or Firefox. Only reason it is moderate is due to how popular Chromium based browsers are among the population. Would have been LOW otherwise.)	Link	UNRESOLVED

Backend after being left unattended for some period of time will start to throw 500 internal server errors on the downloading from the get so ng.	After some time it will occur after leaving the server unattended.	HIGH	Link	Workaround: When the server receives a 500 internal server error, it will restart. The user will have to wait for the server to restart and then retry downloading. At some point we will need to check the stack trace and see if we can get it running.

8. Future Work and Potential Improvements

- Features Not Implemented:

1. **Integration of OpenAI API for Translation and Romanization of Lyrics**:
    - While the karaoke app supports displaying lyrics in real-time, integrating the OpenAI API for translations, romanizations, and lyric analysis was a planned feature that was not implemented. This feature would allow users to translate lyrics to their preferred language, romanize non-Latin scripts, or analyze lyrics for themes, sentiment, or musicality.
2. **Option to Display Song's Metadata and Artist Information**:
    - Initially, the app only focused on the core karaoke experience. Adding the ability to show detailed metadata for each song (e.g., release date, genre, popularity) and artist information (biography, related songs) was planned but not integrated. These features would enhance the user experience by providing context and background on the music being played.

- Ideas for How These Features Could Be Implemented:

1. **React Component for Integrated "Pages"**:
    - The sidebar could be further enhanced to include tabs or buttons that allow easy switching between pages inside the sidebar for specific functionalities. This would streamline the user experience by avoiding multiple separate pages or complex navigation. Each page **inside** the sidebar would host specific tools:
        - **"Search" Page:** Default page, already integrated.
        - **"OpenAI" Page**: Here, users could select from options such as translation, romanization, or lyric analysis. The karaoke lyrics display would then be split into two sections—one showing the original lyrics and the other presenting the translated, romanized, or analyzed lyrics. This separation allows users to easily switch between versions without disrupting the viewing experience.
        - **"Song Metadata" Section**: On the bottom right of the sidebar, a dynamic component could display current song details, such as title, artist, album cover,

and related artist information. This would provide users with a quick overview of the song and its context.

- Enhancements:

1. **Main Page**:
   - Initially, the main page of the app was a simple page featuring a search bar and minimal instructions—just a prompt to search for a song, which would then lead the user to the karaoke page. Based on user feedback and project evolution, we updated the main page to include instructions on how to use the app and what to expect. This helped guide users through the process and made the app more accessible.
   - A prominent button was added to transition directly to the main karaoke page, enhancing usability and ensuring that users could quickly start their karaoke experience.

2. **Interactive Progress Bar and Lyrics**:
   - Originally, the progress (scrub) bar was non-interactive, meaning users could not skip to different parts of the song manually. We also didn't have synchronized lyrics that changed with the song's progression. This forced users to manually scroll through the lyrics, which was not ideal for a seamless karaoke experience.
   - The feedback highlighted the need for an interactive scrub bar that synced with the song's playback, allowing users to jump to specific parts of the song effortlessly. We also incorporated a play/pause button and replay controls to improve the experience by giving users more control over the playback.
   - By adding manual and auto-scroll modes for lyrics, we addressed the limitation of only allowing manual scrolling. This feature not only improved usability but also ensured a more engaging and intuitive karaoke experience by enabling lyrics to automatically scroll in sync with the song or allow for free scrolling when the user desired.

3. **Advanced Features**:
   - **Lyrics Interaction**: The ability to directly **click on lyrics** and jump to specific song parts was not initially considered. This feature was introduced later based on feedback, allowing users to engage more deeply with the music.
   - **Auto and Manual Scroll**: Previously, the automatic scrolling feature was absent. Now, users can choose between auto scroll—where the app moves through the lyrics based on the song's timing—and manual scroll—enabling users to freely explore the lyrics. After 3 seconds of inactivity in manual scroll mode, the app automatically switches back to auto scroll, ensuring a smooth user experience.

By integrating these features, the app transformed from a basic karaoke search tool into a comprehensive karaoke experience, providing users with powerful tools to sing along with their favorite songs in a more personalized and interactive way. These enhancements demonstrate the app's evolution and commitment to improving the karaoke experience based on user feedback and project insights.

9. Lessons Learned and Takeaways

Challenges:

- Backend Deployment: Deploying a FastAPI backend requiring GPU power for the separation process on the cloud proved to be more challenging than expected. We underestimated the time and effort needed for this task. Initially, we tried using platforms like Google Cloud and Render but encountered significant issues. Google Cloud lacked affordable GPUs, while Render's free plan didn't offer sufficient processing power. Eventually, we decided to deploy the backend on a personal computer, which turned out to be much simpler. By exposing port 8000 on the local network and using ngrok to obtain a public URL, we were able to make the server accessible. We updated the frontend code to address this new URL, allowing it to communicate with the FastAPI server whenever the personal computer was running. This solution enabled us to process and return the instrumentals successfully.

Key Learnings:

- Research and Organization: We realized the importance of having a well-structured approach to each task. While initial research can be time-consuming, it significantly streamlines the workflow later on. Proper research provides a deeper understanding of the project and minimizes redundant personal exploration, saving the team valuable time.
- Mastering the Tech Stack: Being proficient with every framework and language in the tech stack proved crucial. By ensuring that team members were familiar with the tools at hand, we were better equipped to collaborate and contribute across all aspects of the project as needed.

10. Appendix

Group members and contributions

An

- Designed FastAPI backend and API endpoints
- Designed architecture of React frontend and FastAPI backend
- Implemented and automated song processing workflow by integrating chosen APIs and packages
- Managed data serialization between the backend and frontend and streamlined song queue management.

Caiden

- Designed FastAPI backend and API endpoints
- Managed backend deployment on personal computer
- Test and research necessary APIs and packages
- Created React project file, logo, and the Queue UI

Jake

- Designed, styled, and polished the user interface to provide an intuitive experience for users.
- Synchronized the display of lyrics with the music on the user interface.
- Developed play, pause, replay, and skip buttons to enhance user control.
- Implemented an interactive scrubbing bar and lyrics display, allowing users to skip to specific parts of songs by clicking on the scrub bar or lyrics.
- Introduced both autoscroll and manual scroll options for lyrics to cater to different user preferences.
- Implemented backend endpoint to retrieve lyrics of a song based on artist name and song title from the Genius platform, but was scrapped after deciding to use the Spotify-Lyrics-API and Spotify API instead.

Parsa

- Managed Frontend deployment on Vercel
- Managed backend deployment on personal computer
- Implemented backend endpoint to retrieve data of a Youtube video based on the url from the YoutubeAPI and an Express server to work with React frontend which were both scrapped.
- Assisted in developing play, pause, and volume control buttons.

Peer Testing Feedback Form

Link to form:
https://docs.google.com/spreadsheets/d/17GP_bcsPLyy_7I-NSDUfBN5oVIB72IG3ANWtWUX9ERU/edit?usp=sharing

Survey questions used for collecting feedback.

General Impressions

What are your overall thoughts about the application?

Did you find the application enjoyable or intuitive to use? Why or why not?

Task-Specific Feedback

For each task you performed, please answer the following:

1. How easy or difficult was it to complete this task?
2. Was anything unclear or confusing during this task? If so, what?
3. Did you encounter any bugs or issues while completing this task? Please describe.

Task 1: Searching Songs

Task 2: Queueing up songs

Task 3: Singing along to the lyrics

Usability Suggestions

Do you have any suggestions for improving the application’s navigation?

Were there any design elements that you found unclear or distracting? If so, what?

What features or improvements would you suggest to enhance the overall user experience?

Summary of survey responses from user testing.

User testing feedback highlighted VocaFree’s creative and engaging design, with users praising its minimalist interface and enjoyable karaoke features. The application was generally found intuitive and easy to use, with straightforward song searching and lyric synchronization. However, some users noted areas for improvement, such as making the queue and progress bar more intuitive and adding features like auto-playing songs in the queue. Suggestions included expanding the search panel, providing a clear loading or progress bar, and incorporating a local database for faster song retrieval. Overall, users found the app impressive, with clear navigation and no major distractions, while offering constructive feedback to refine its functionality and enhance the user experience.