Final Project Presentation

Kigo

December 3 2024

TEAM

Kelly Lowrance: Back-End Developer and Quality Assurance Specialist

Noah Kabel: Back-End Developer and API Specialist

Sam Allen: Front-End Developer and Project Coordinator

PROJECT INTRO

Kigo - lyric-driven haiku generator

Our program utilizes several APIs to generate haikus using song lyrics from the user's most-listened songs on Spotify. Kigo pulls the user's listening history and identifies the most-listened to songs, fetches the lyrics and processes them, it then parses lyrics to fit the specifications of a haiku, and then displays the haiku in an artistic manner.

PROBLEM STATEMENT

We set out to create a fun and interactive program that generates personalized haikus from a user's most-listened songs. This project provides value and entertainment for music lovers by offering a playful and unique way to engage with their favorite songs. With visually engaging presentations of the haikus, our project will combine both technology and art to create a unique, personalized experience for any fan of music.

PROJECT SCHEDULE

Planned schedule:

(Weeks 1-2) Research & Planning
- Research APIs (Spotify, Lyrics.ovh)
- Define project scope and features

(Weeks 3-6) Development
- Integrate listening history retrieval
- Integrate song lyrics retrieval
- Develop an algorithm to generate the haikus

(Weeks 7-8) User Interface Design
- Design GUI layout and features
- Implement haiku display

(Weeks 9-10) Testing
- Conduct testing
- Gather user feedback for improvements

(Weeks 11-12) Documentation & Finalization
- Prepare user documentation and specifications
- Finalize project for delivery

PROJECT SCHEDULE

Actual Schedule:

(Sep 1-3) Project Ideation
- Brainstorm ideas for the project

(Sep 4-5) Research APIs: Kelly, Noah
- Research APIs (Spotify, Lyrics.ovh)
- Define project scope, features, and key functionalities

(Sep 16–20) Set Up Repositories: Sam
- Create GitHub Repository and necessary branches
- Configure repository to use GitHub Pages for static webpage hosting

(Sep 20-Oct 10) Pulling user songs using Spotify API: Noah
- Setup environment with dependencies for Spotify API.
- Develop code to pull top songs from a user.

(Sep 21–Oct 5) Spotify API Integration: Sam
- Integrate Spotify authentication functionality written by Noah into existing GUI
- Test the authentication flow and refine the process

PROJECT SCHEDULE

Actual Schedule:

(Oct 6–20) UI Development: Sam

- Design user interface, including decorative elements such as the background, logo, and cursor
- Develop and implement core GUI functionality for navigation and interactivity

(Oct 20-Nov 10) Pull Song lyrics using lyrics.ovh: Noah

- Integrate lyrics.ovh API into environment.
- Develop code to retrieve lyrics for a given song.

(Oct 21–Nov 20) Syllable Counter Development: Kelly

- Write the initial algorithm for syllable counting
- Test and optimize algorithm for accuracy

(Nov 11–20) Haiku Generator: Kelly

- Develop logic to match processed lyric lines to a 5-7-5 syllable pattern and generate haikus
- Test haiku generation for consistency and variability

PROJECT SCHEDULE

Actual Schedule:

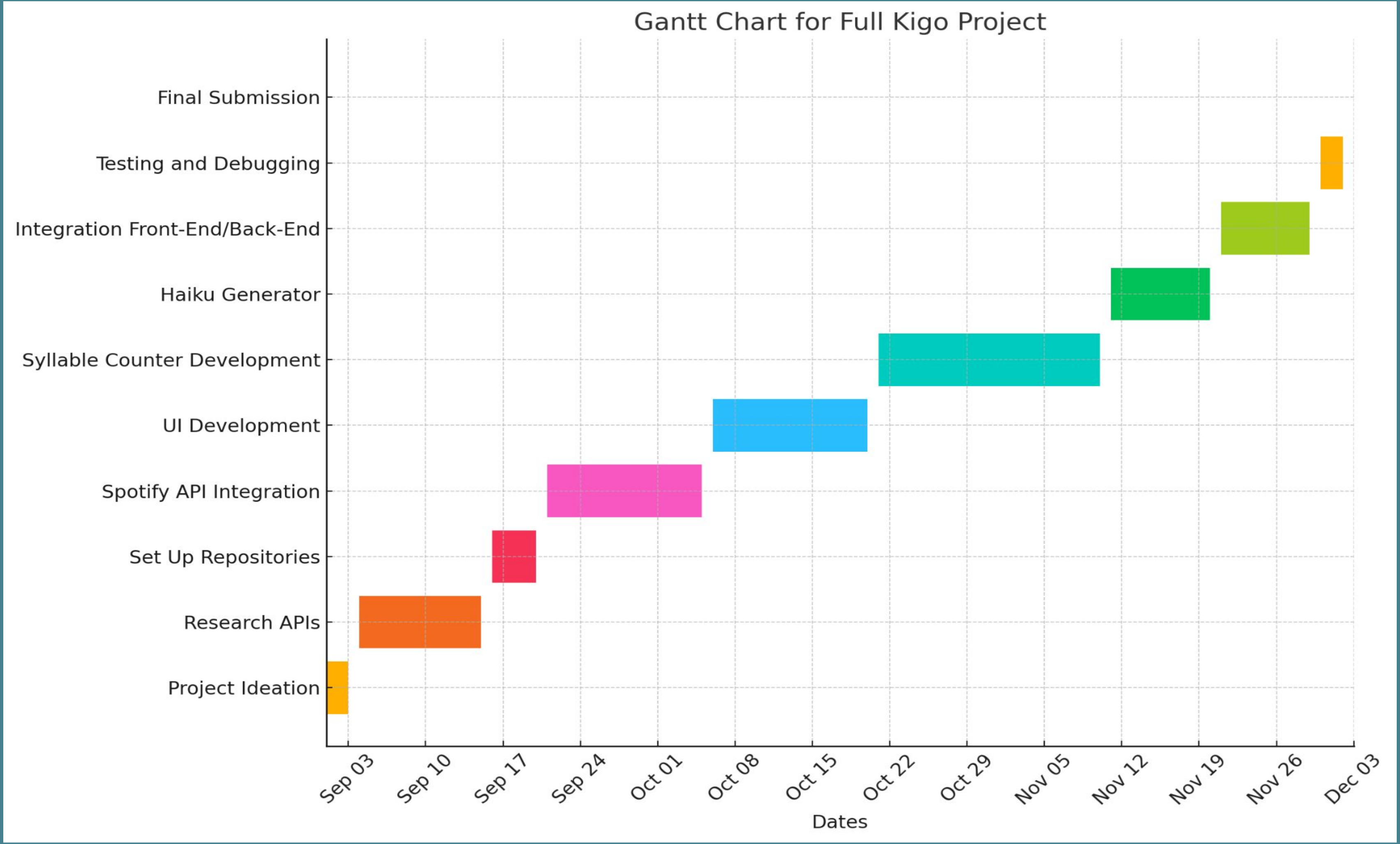
(Nov 21–29) Integration Front-End/Back-End: Sam

- Connect the Node.js backend to the React frontend
- Set up routes for various tasks such as Spotify authentication, lyric fetching, and haiku generation

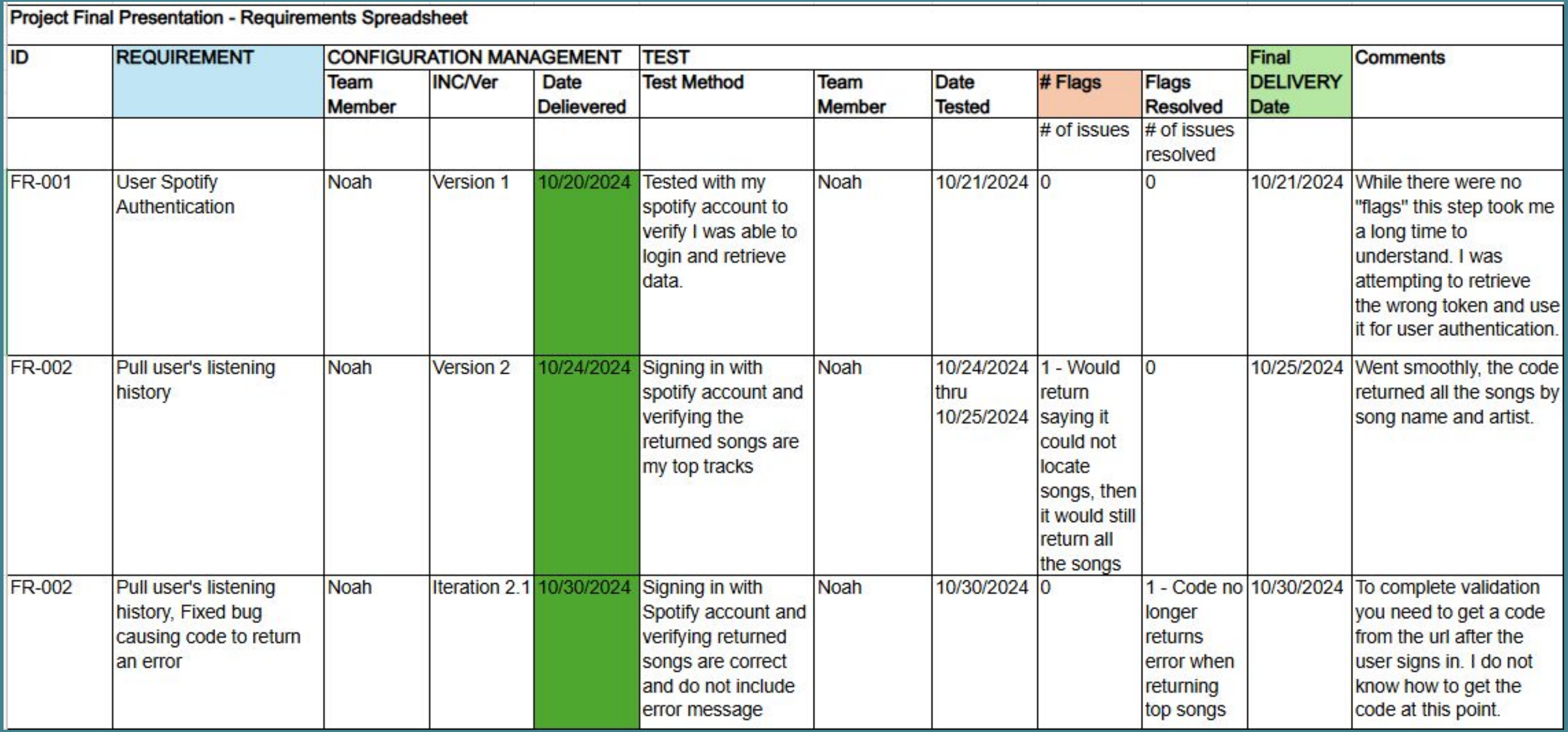
(Nov 30–Dec 2) Testing and Debugging: Team

- Conduct unit, integration, and system testing across all components
- Address any remaining issues

(Dec 3) Final Submission: Team

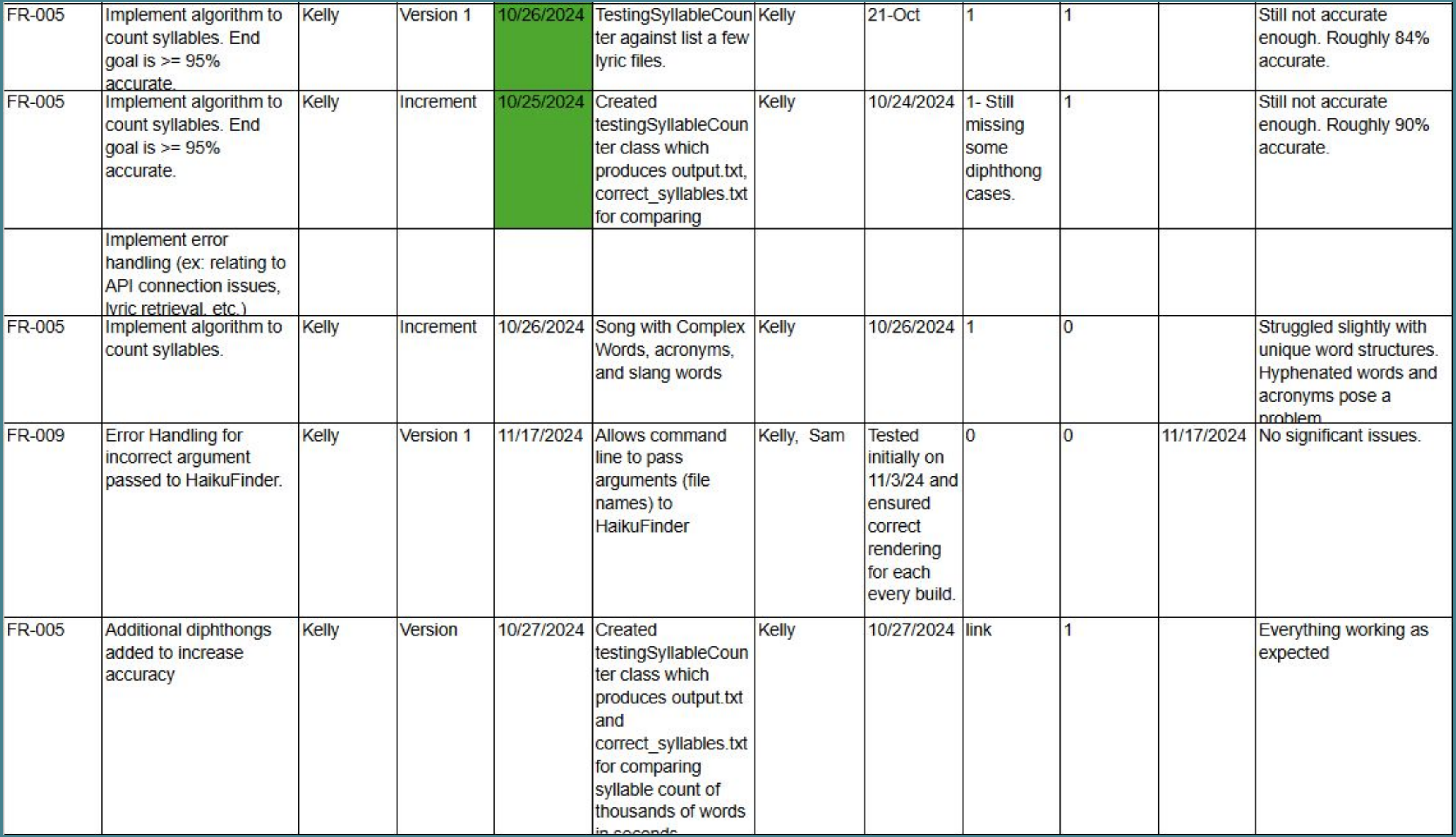

Gantt Chart for Full Kigo Project
Final Submission
Testing and Debugging
Integration Front-End/Back-End
Haiku Generator
Syllable Counter Development
UI Development
Spotify API Integration
Set Up Repositories
Research APIs
Project Ideation
Sep 03
Sep 10
Sep 17
Sep 24
Oct 01
Oct 08
Oct 15
Oct 22
Oct 29
Nov 05
Nov 12
Nov 19
Nov 26
Dec 03
Dates

PROJECT REQUIREMENTS

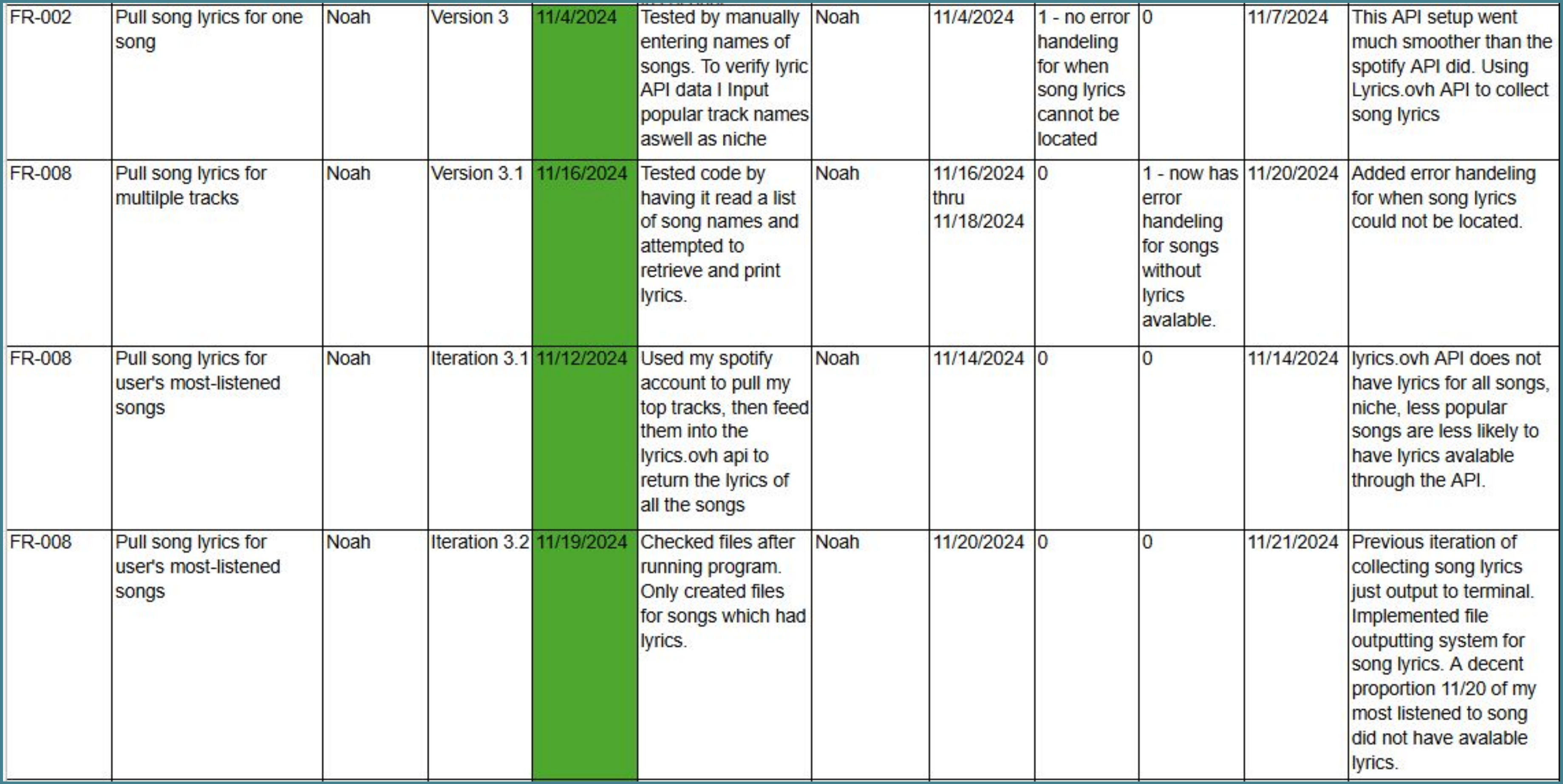


Project Final Presentation - Requirements Spreadsheet

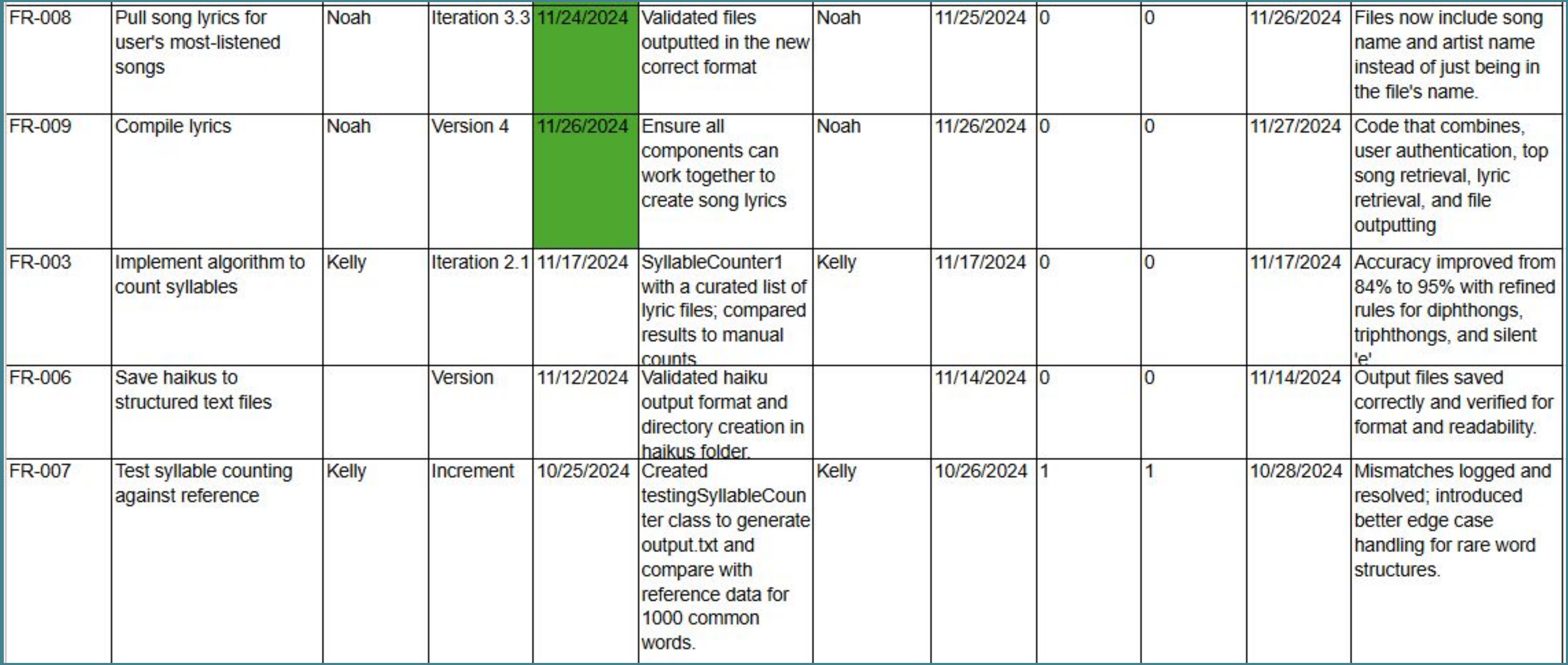
ID	REQUIREMENT	CONFIGURATION MANAGEMENT			TEST					Final DELIVERY Date	Comments
		Team Member	INC/Ver	Date Delievered	Test Method	Team Member	Date Tested	# Flags	Flags Resolved		
								# of issues	# of issues resolved		
FR-001	User Spotify Authentication	Noah	Version 1	10/20/2024	Tested with my spotify account to verify I was able to login and retrieve data.	Noah	10/21/2024	0	0	10/21/2024	While there were no "flags" this step took me a long time to understand. I was attempting to retrieve the wrong token and use it for user authentication.
FR-002	Pull user's listening history	Noah	Version 2	10/24/2024	Signing in with spotify account and verifying the returned songs are my top tracks	Noah	10/24/2024 thru 10/25/2024	1 - Would return saying it could not locate songs, then it would still return all the songs	0	10/25/2024	Went smoothly, the code returned all the songs by song name and artist.
FR-002	Pull user's listening history, Fixed bug causing code to return an error	Noah	Iteration 2.1	10/30/2024	Signing in with Spotify account and verifying returned songs are correct and do not include error message	Noah	10/30/2024	0	1 - Code no longer returns error when returning top songs	10/30/2024	To complete validation you need to get a code from the url after the user signs in. I do not know how to get the code at this point.



FR-005	Implement algorithm to count syllables. End goal is >= 95% accurate.	Kelly	Version 1	10/26/2024	TestingSyllableCounter against list a few lyric files.	Kelly	21-Oct	1	1		Still not accurate enough. Roughly 84% accurate.
FR-005	Implement algorithm to count syllables. End goal is >= 95% accurate.	Kelly	Increment	10/25/2024	Created testingSyllableCounter class which produces output.txt, correct_syllables.txt for comparing	Kelly	10/24/2024	1- Still missing some diphthong cases.	1		Still not accurate enough. Roughly 90% accurate.
	Implement error handling (ex: relating to API connection issues, lyric retrieval, etc.)										
FR-005	Implement algorithm to count syllables.	Kelly	Increment	10/26/2024	Song with Complex Words, acronyms, and slang words	Kelly	10/26/2024	1	0		Struggled slightly with unique word structures. Hyphenated words and acronyms pose a problem
FR-009	Error Handling for incorrect argument passed to HaikuFinder.	Kelly	Version 1	11/17/2024	Allows command line to pass arguments (file names) to HaikuFinder	Kelly, Sam	Tested initially on 11/3/24 and ensured correct rendering for each every build.	0	0	11/17/2024	No significant issues.
FR-005	Additional diphthongs added to increase accuracy	Kelly	Version	10/27/2024	Created testingSyllableCounter class which produces output.txt and correct_syllables.txt for comparing syllable count of thousands of words	Kelly	10/27/2024	link	1		Everything working as expected

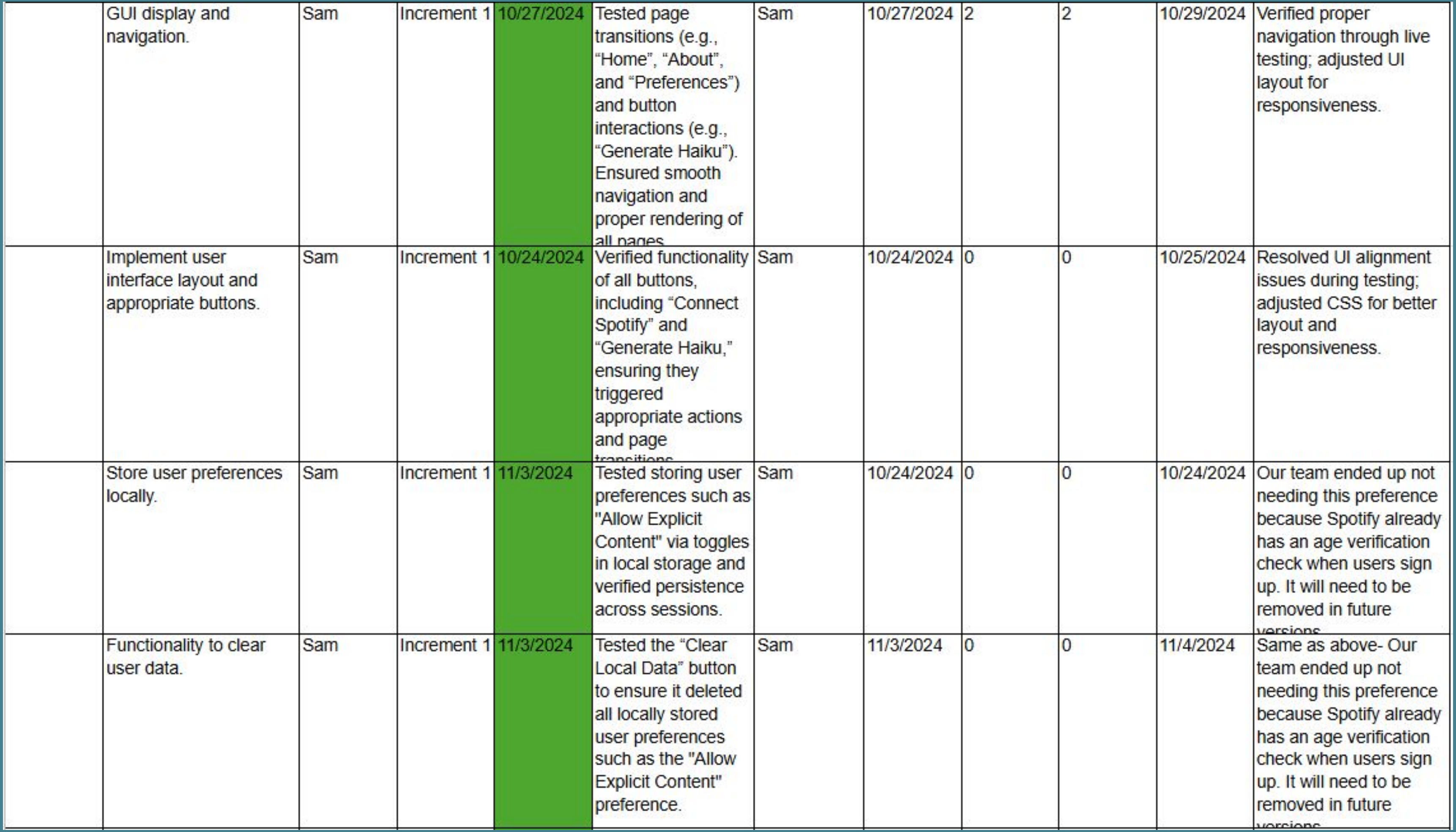


FR-002	Pull song lyrics for one song	Noah	Version 3	11/4/2024	Tested by manually entering names of songs. To verify lyric API data I Input popular track names aswell as niche	Noah	11/4/2024	1 - no error handeling for when song lyrics cannot be located	0	11/7/2024	This API setup went much smoother than the spotify API did. Using Lyrics.ovh API to collect song lyrics
FR-008	Pull song lyrics for multilple tracks	Noah	Version 3.1	11/16/2024	Tested code by having it read a list of song names and attempted to retrieve and print lyrics.	Noah	11/16/2024 thru 11/18/2024	0	1 - now has error handeling for songs without lyrics avalable.	11/20/2024	Added error handeling for when song lyrics could not be located.
FR-008	Pull song lyrics for user's most-listened songs	Noah	Iteration 3.1	11/12/2024	Used my spotify account to pull my top tracks, then feed them into the lyrics.ovh api to return the lyrics of all the songs	Noah	11/14/2024	0	0	11/14/2024	lyrics.ovh API does not have lyrics for all songs, niche, less popular songs are less likely to have lyrics avalable through the API.
FR-008	Pull song lyrics for user's most-listened songs	Noah	Iteration 3.2	11/19/2024	Checked files after running program. Only created files for songs which had lyrics.	Noah	11/20/2024	0	0	11/21/2024	Previous iteration of collecting song lyrics just output to terminal. Implemented file outputting system for song lyrics. A decent proportion 11/20 of my most listened to song did not have avalable lyrics.

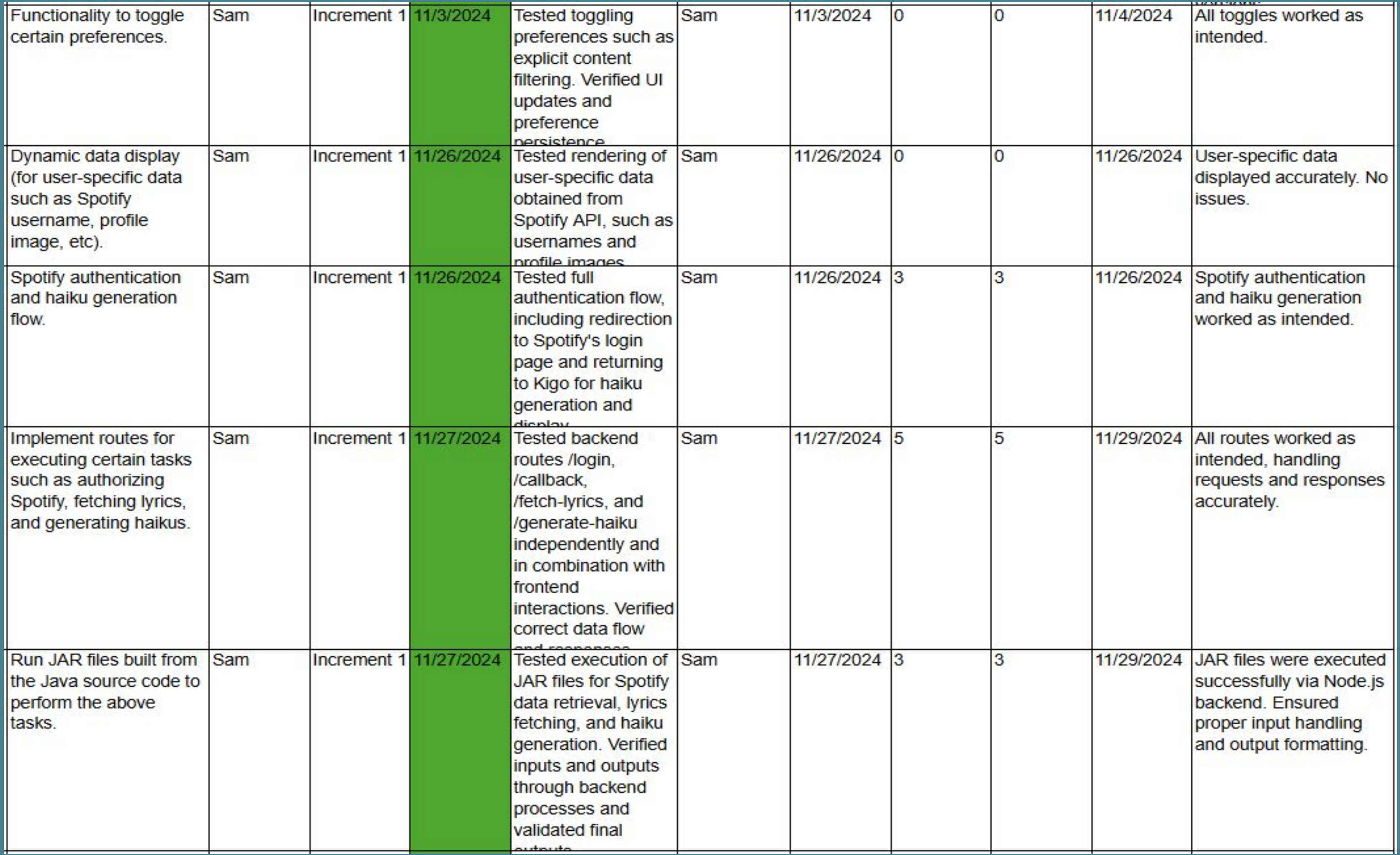


FR-008	Pull song lyrics for user's most-listened songs	Noah	Iteration 3.3	11/24/2024	Validated files outputted in the new correct format	Noah	11/25/2024	0	0	11/26/2024	Files now include song name and artist name instead of just being in the file's name.
FR-009	Compile lyrics	Noah	Version 4	11/26/2024	Ensure all components can work together to create song lyrics	Noah	11/26/2024	0	0	11/27/2024	Code that combines, user authentication, top song retrieval, lyric retrieval, and file outputting
FR-003	Implement algorithm to count syllables	Kelly	Iteration 2.1	11/17/2024	SyllableCounter1 with a curated list of lyric files; compared results to manual counts	Kelly	11/17/2024	0	0	11/17/2024	Accuracy improved from 84% to 95% with refined rules for diphthongs, triphthongs, and silent 'e'
FR-006	Save haikus to structured text files		Version	11/12/2024	Validated haiku output format and directory creation in haikus folder.		11/14/2024	0	0	11/14/2024	Output files saved correctly and verified for format and readability.
FR-007	Test syllable counting against reference	Kelly	Increment	10/25/2024	Created testingSyllableCounter class to generate output.txt and compare with reference data for 1000 common words.	Kelly	10/26/2024	1	1	10/28/2024	Mismatches logged and resolved; introduced better edge case handling for rare word structures.

NFR-001	Ensure reliable file operations	Kelly	Increment	10/20/2024	Simulated various input file issues (missing, empty, corrupt) to test error handling	Kelly	10/22/2024	1	1	10/22/2024	Error messages refined to be more user-friendly and clear.
NFR-002	Support extensibility in processing rules	Kelly	Increment	10/26/2024	Added regex patterns for vocable detection and tested for performance impacts during lyric processing.	Noah	10/26/2024	0	0	10/28/2024	Successfully extended regex rules without significant performance degradation.
FR-008	Handle incorrect arguments in HaikuFinder	Kelly	Increment	11/16/2024	Passed invalid arguments to verify error handling and fallback behavior.	Kelly	11/17/2024	0	0	11/17/2024	HaikuFinder now handles invalid inputs gracefully and logs errors appropriately.
Hai-00Y	Improve diphthong and triphthong handling	Kelly	Increment	10/27/2024	Tested additional rules for syllable counting in SyllableCounter1 Using diphthong- and triphthong-heavy	Kelly	10/28/2024	0	0	10/29/2024	Accuracy improved by 5% with added cases for less common diphthongs and triphthongs.
	Design visuals for GUI (app background, cursor design)	Sam	Version 1	11/3/2024	Visuals are stored in docs>src>visual and all relevant code points to this location. Ensured that visuals rendered correctly for every build of the webpage	Sam	11/3/2024	0	0	11/3/2024	Verified proper navigation through live testing; adjusted UI layout for responsiveness.



	GUI display and navigation.	Sam	Increment 1	10/27/2024	Tested page transitions (e.g., “Home”, “About”, and “Preferences”) and button interactions (e.g., “Generate Haiku”). Ensured smooth navigation and proper rendering of all pages	Sam	10/27/2024	2	2	10/29/2024	Verified proper navigation through live testing; adjusted UI layout for responsiveness.
	Implement user interface layout and appropriate buttons.	Sam	Increment 1	10/24/2024	Verified functionality of all buttons, including “Connect Spotify” and “Generate Haiku,” ensuring they triggered appropriate actions and page transitions	Sam	10/24/2024	0	0	10/25/2024	Resolved UI alignment issues during testing; adjusted CSS for better layout and responsiveness.
	Store user preferences locally.	Sam	Increment 1	11/3/2024	Tested storing user preferences such as "Allow Explicit Content" via toggles in local storage and verified persistence across sessions.	Sam	10/24/2024	0	0	10/24/2024	Our team ended up not needing this preference because Spotify already has an age verification check when users sign up. It will need to be removed in future versions
	Functionality to clear user data.	Sam	Increment 1	11/3/2024	Tested the “Clear Local Data” button to ensure it deleted all locally stored user preferences such as the "Allow Explicit Content" preference.	Sam	11/3/2024	0	0	11/4/2024	Same as above- Our team ended up not needing this preference because Spotify already has an age verification check when users sign up. It will need to be removed in future versions



Functionality to toggle certain preferences.	Sam	Increment 1	11/3/2024	Tested toggling preferences such as explicit content filtering. Verified UI updates and preference persistence	Sam	11/3/2024	0	0	11/4/2024	All toggles worked as intended.
Dynamic data display (for user-specific data such as Spotify username, profile image, etc).	Sam	Increment 1	11/26/2024	Tested rendering of user-specific data obtained from Spotify API, such as usernames and profile images	Sam	11/26/2024	0	0	11/26/2024	User-specific data displayed accurately. No issues.
Spotify authentication and haiku generation flow.	Sam	Increment 1	11/26/2024	Tested full authentication flow, including redirection to Spotify's login page and returning to Kigo for haiku generation and display	Sam	11/26/2024	3	3	11/26/2024	Spotify authentication and haiku generation worked as intended.
Implement routes for executing certain tasks such as authorizing Spotify, fetching lyrics, and generating haikus.	Sam	Increment 1	11/27/2024	Tested backend routes /login, /callback, /fetch-lyrics, and /generate-haiku independently and in combination with frontend interactions. Verified correct data flow and responses	Sam	11/27/2024	5	5	11/29/2024	All routes worked as intended, handling requests and responses accurately.
Run JAR files built from the Java source code to perform the above tasks.	Sam	Increment 1	11/27/2024	Tested execution of JAR files for Spotify data retrieval, lyrics fetching, and haiku generation. Verified inputs and outputs through backend processes and validated final outputs	Sam	11/27/2024	3	3	11/29/2024	JAR files were executed successfully via Node.js backend. Ensured proper input handling and output formatting.

PROJECT CONFIGURATION MANAGEMENT

Version Control System:

- The project utilizes Git as the primary version control system to manage source code and related files efficiently.
  - We created and maintained 7 branches. Main, Kigo-source, Kigo-backend, kigo-documentation, and individual branches for all members.
- We kept track of progress through our requirements spreadsheet.

PROJECT SOFTWARE DEVELOPMENT

Primary Languages:

- Java: Used for backend logic in the JAR libraries.
- JavaScript: Used for the Node.js backend and React frontend.
- HTML and CSS: Used for frontend design and user interface styling.

Other:

- We did not use COTS
- We decided to license our code under an MIT license
- We planned to use a feature in the Eclipse IDE to generate JavaDocs from source code comments, but did not end up implementing this in our final documentation.

SOFTWARE ENGINEERING PRINCIPLES

1. **Modularity**
   - The project is divided into three distinct components: the frontend (React.js), backend (Node.js), and Java libraries (compiled into JAR files), allowing for independent development and testing.
2. **Reusability**
   - Core functionalities like the syllable counting algorithm and haiku generation are implemented in reusable Java classes, making them easily adaptable for future enhancements.
3. **Encapsulation**
   - Sensitive data (e.g., Spotify API credentials) is securely managed using environment variables, ensuring data is not hardcoded in the source files.

SOFTWARE ENGINEERING PRINCIPLES

4. **Incremental Development**
    - Features were developed and tested incrementally, such as Spotify authentication, syllable counting, and haiku generation, allowing iterative improvements and quick fixes.
5. **Maintainability**
    - Clean code practices, detailed logging, and organized project structure improve readability and make it easier for future developers to understand and modify the code.

SOFTWARE ENGINEERING PRINCIPLES

We used the Scrum framework to guide our software development. This is because:

- Works well for small teams
- Works well for fast-moving development projects
- Flexible framework that is effective for projects with requirements that may change/unknown solutions

We utilized our requirements spreadsheet for agile project management. With this we tracked our progress, dates, issues, and additional comments on what was implemented.

SOFTWARE ENGINEERING TOOLS

Tools used in Kigo development:

Spotify API - used to fetch relevant user data, such as top tracks, username, and profile image
lyrics.ovh API - used to retrieve lyrics for the user's top tracks
Eclipse 2023-003 - development environment used
Collective requirements spreadsheet - for Agile project management
GitHub - for version control management and collaboration
Node.js - to handle the backend server for execution of JAR files
React.js - frontend framework for building the user interface
Maven - build and dependency management for Java projects
cloc (count lines of code) - to measure the size of the codebase
npm (node package manager) - manages dependencies for the Node.js backend and React frontend

PROJECT DESIGN

Front-end:

Purpose: Collect user input and display haikus.

Technologies: React.js, Node.js, JavaScript, HTML, CSS

Responsibilities:

- Let users log in, and collect authentication code.
- Displaying generated haikus.
- Communicating with the backend.

Back-end:

Purpose: Make API calls, handle haiku generation, and manage data.

Technologies: Java, Maven.

Responsibilities:

- User authentication with Spotify.
- Fetching user's top songs.
- Fetching lyrics from external API.
- Parsing lyrics to generate haikus.
- External APIs

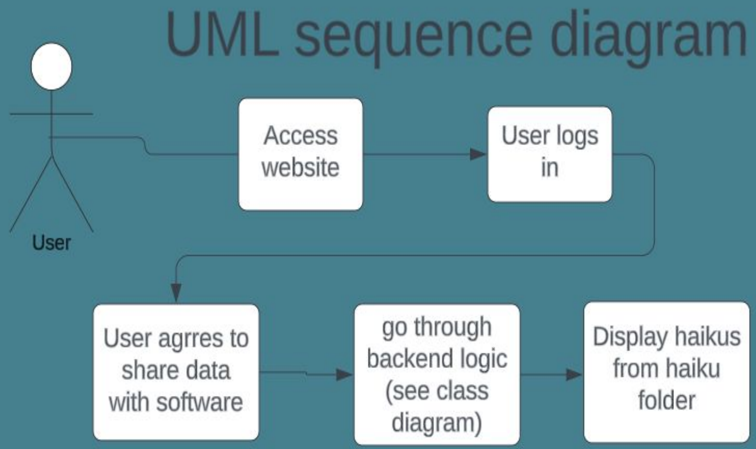
PROJECT MODEL - Agile perspective model

1. User Perspective

- User Stories:
  - A user needs to be able to access website and log into their spotify account
  - A user needs to be able to see haikus produces from song lyrics.

Acceptance Criteria:

- User can access the website
- User can log into their spotify account
- User's spotify data is collected
- Haikus are generated from song lyrics
- Haikus are displayed for the user to view


UML sequence diagram
User
Access website
User logs in
User agrres to share data with software
go through backend logic (see class diagram)
Display haikus from haiku folder

PROJECT MODEL - Agile perspective model

2. Development Perspective

- Tasks:
  - Pull user’s most listened to songs
  - Create haikus from user’s top songs
  - Display haikus on web page
- Processes:
  - Use API calls to get user’s top songs
  - Use API calls to retrieve lyrics for songs
  - Parse song lyrics to find lyrics that fit haiku scheme
  - Save haikus
  - Create web page
  - Display haikus on web page

PROJECT MODEL - Agile perspective model

3. Testing Perspective

Test cases:

1. Verify user’s top songs
2. Test lyric retrieval
3. Ensure parsing lyrics yields correctly sized lines for haikus
4. Haikus can be displayed on webpage

PROJECT MODEL - Agile perspective model

4. Operational Perspective

Deployment Tasks:

- Set up the server environment: Ensure that the server is configured to handle API calls for retrieving Spotify data and song lyrics.
- Integrate Spotify and lyrics APIs: Ensure that the necessary API keys are in place and that the application is able to securely make requests to Spotify and lyrics.ovh APIs.
- Design web page.
- Integrate logic into web page.

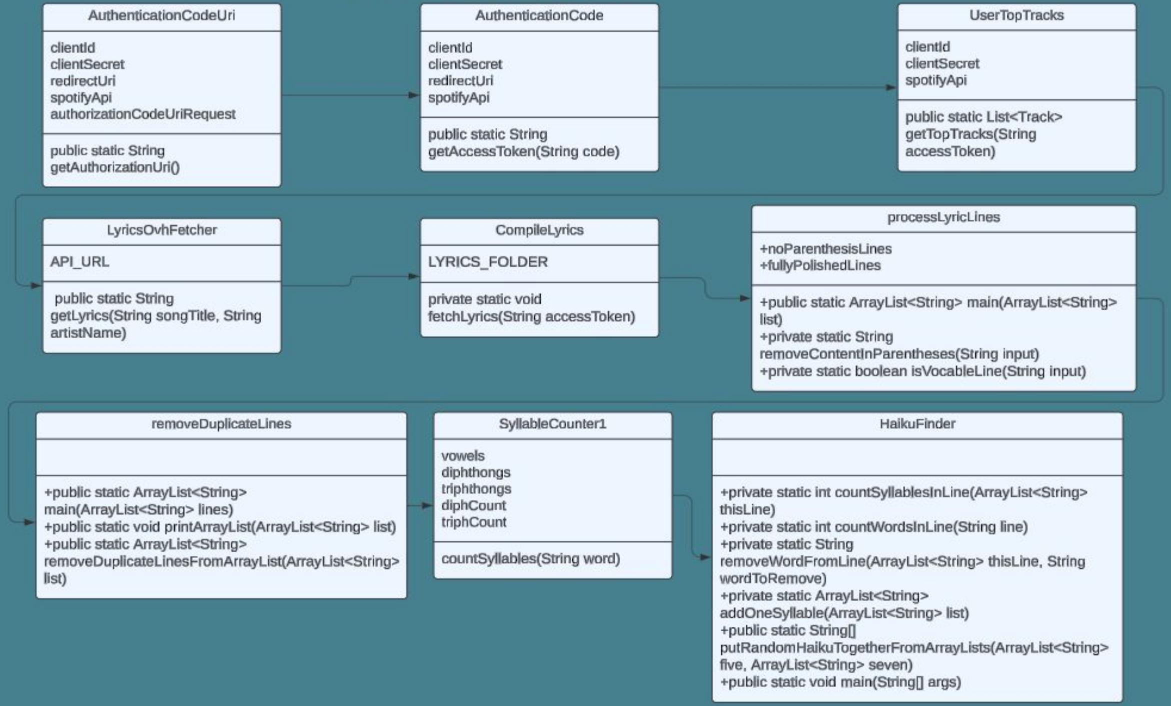
HIGH LEVEL ARCHITECTURE

Components:

- Front-end:
  - Website user interacts with.
  - We use React, Node.js, Javascript, CSS
- Back-end:
  - Haiku Generation Service: This service is responsible for taking the user’s top songs and generating haikus from the lyrics.
    - Utilizes the Spotify API to fetch user’s top tracks and the Lyrics.ovh API to fetch the lyrics.
    - Uses a text-processing algorithm to parse the lyrics and generate haikus based on the 5-7-5 syllable rule

HIGH LEVEL ARCHITECTURE

UML class diagram



AuthenticationCodeUri
clientId
clientSecret
redirectUri
spotifyApi
authorizationCodeUriRequest
public static String getAuthorizationUri()

AuthenticationCode
clientId
clientSecret
redirectUri
spotifyApi
public static String getAccessToken(String code)

UserTopTracks
clientId
clientSecret
spotifyApi
public static List<Track> getTopTracks(String accessToken)

LyricsOvhFetcher
API_URL
public static String getLyrics(String songTitle, String artistName)

CompileLyrics
LYRICS_FOLDER
private static void fetchLyrics(String accessToken)

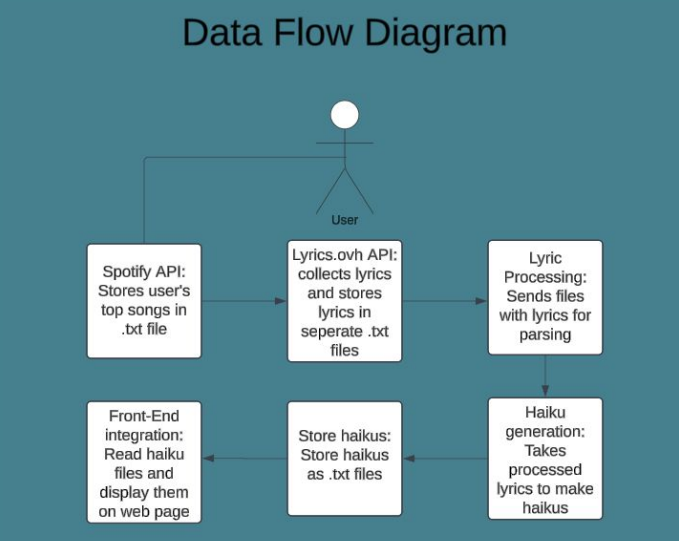
processLyricLines
+noParenthesisLines
+fullyPolishedLines
+public static ArrayList<String> main(ArrayList<String> list)
+private static String removeContentInParentheses(String input)
+private static boolean isVocableLine(String input)

removeDuplicateLines
+public static ArrayList<String> main(ArrayList<String> lines)
+public static void printArrayList(ArrayList<String> list)
+public static ArrayList<String> removeDuplicateLinesFromArrayList(ArrayList<String> list)

SyllableCounter1
vowels
diphthongs
triphthongs
diphCount
triphCount
countSyllables(String word)

HaikuFinder
+private static int countSyllablesInLine(ArrayList<String> thisLine)
+private static int countWordsInLine(String line)
+private static String removeWordFromLine(ArrayList<String> thisLine, String wordToRemove)
+private static ArrayList<String> addOneSyllable(ArrayList<String> list)
+public static String[] putRandomHaikuTogetherFromArrayLists(ArrayList<String> five, ArrayList<String> seven)
+public static void main(String[] args)

HIGH LEVEL ARCHITECTURE cont.


Data Flow Diagram
User
Spotify API: Stores user's top songs in .txt file
Lyrics.ovh API: collects lyrics and stores lyrics in seperate .txt files
Lyric Processing: Sends files with lyrics for parsing
Haiku generation: Takes processed lyrics to make haikus
Store haikus: Store haikus as .txt files
Front-End integration: Read haiku files and display them on web page

Our software primarily handles data in the form of .txt files containing lists of user’s top songs, files containing the lyrics for the songs, and files containing haikus made from the lyrics.

PROJECT EXECUTION

All of the planned essential requirements of the Kigo App are implemented in the product delivered. These include:

- Spotify authentication
- Fetching the user’s top tracks
- Retrieving lyrics from these top tracks
- Processing these lyrics
- Generating several haikus.

Currently, the Kigo App is fully functional in terms of these requirements. We planned to implement other capabilities that would make the software more robust and portable, but were unable to fully accomplish these due to time constraints.

TESTING - Syllabification

Began with the most basic method of counting syllables in a word by counting vowels. Often 60-75% accurate. Overcounts silent vowels, diphthongs, and triphthongs.

Rule-based syllable counters (Kigo) tend to achieve **90–95%** accuracy, depending on how edge cases (e.g., silent "e", diphthongs, triphthongs, etc.) are handled.

Machine learning-based approaches (like those leveraging neural networks or decision trees) can achieve **96–98% accuracy** but often require significantly more computational resources.

For a perfect syllable counter, reaching 100% accuracy in syllabification is almost impossible due to the irregularities and exceptions in English pronunciation.

TESTING - Syllabification

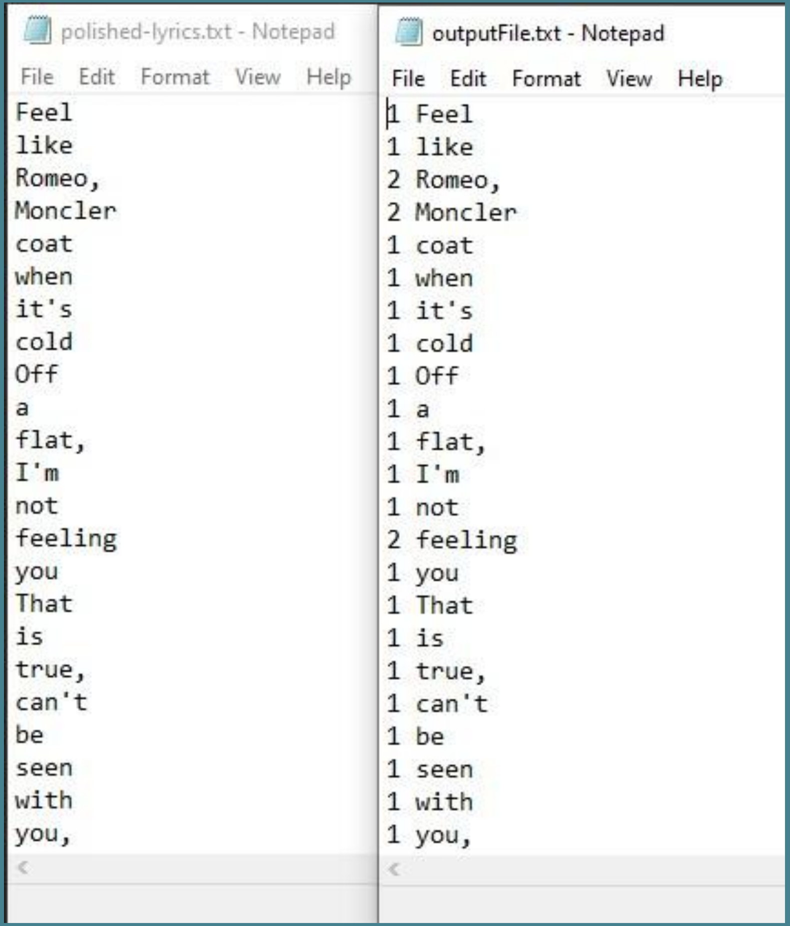
To increase testing speed and accuracy, we implemented a separate class testingSyllableCounter which processes a file containing words to be tested and outputs to a new file: one integer value (count) followed by the word tested.

Utilizes SyllableCounter.net which uses a dictionary to count syllables. This website outputs words with corresponding count.

Then testingSyllableCounter opens both the correct-syllables.txt & output.txt and compares the files line by line, keeping totals for syllable count, word, missed words.

TESTING - Syllabification

Reads lyric words 1-by-1 and produces the file on the right.


polished-lyrics.txt - Notepad
File Edit Format View Help

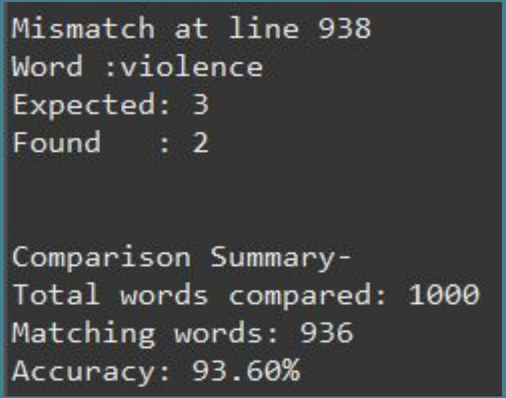
```
Feel
like
Romeo,
Moncler
coat
when
it's
cold
Off
a
flat,
I'm
not
feeling
you
That
is
true,
can't
be
seen
with
you,
```

outputFile.txt - Notepad
File Edit Format View Help

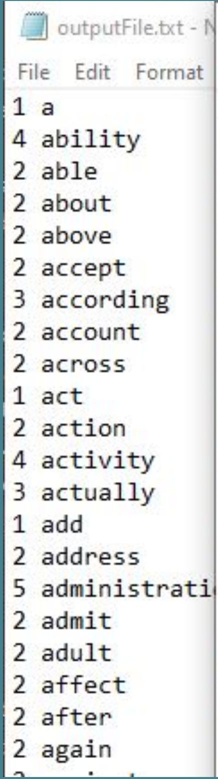
```
1 Feel
1 like
2 Romeo,
2 Moncler
1 coat
1 when
1 it's
1 cold
1 Off
1 a
1 flat,
1 I'm
1 not
2 feeling
1 you
1 That
1 is
1 true,
1 can't
1 be
1 seen
1 with
1 you,
```

TESTING - Syllabification

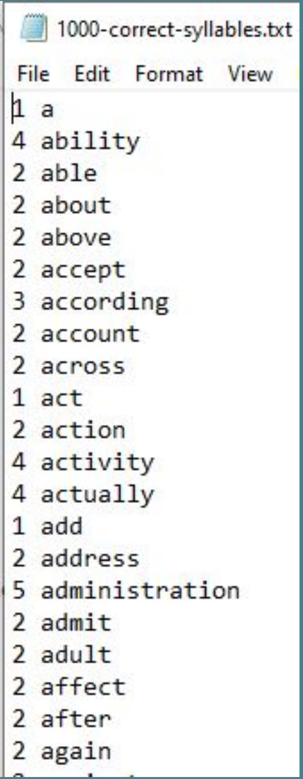
Tested against 1000 most common English words



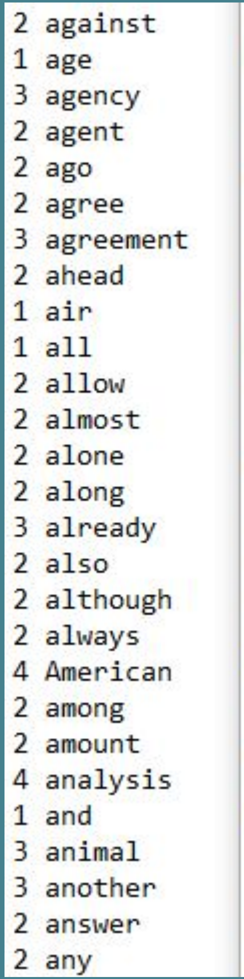
```
Mismatch at line 938
Word :violence
Expected: 3
Found   : 2


Comparison Summary-
Total words compared: 1000
Matching words: 936
Accuracy: 93.60%
```



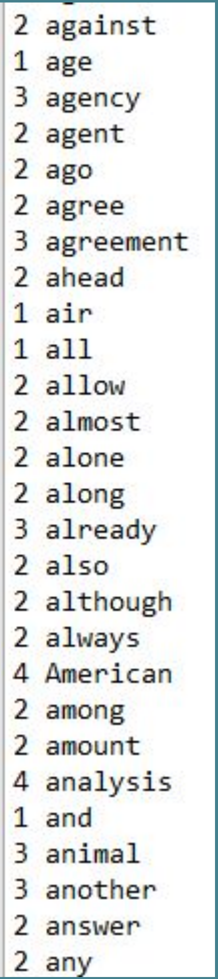
```
1 a
4 ability
2 able
2 about
2 above
2 accept
3 according
2 account
2 across
1 act
2 action
4 activity
3 actually
1 add
2 address
5 administrati
2 admit
2 adult
2 affect
2 after
2 again
```



```
1 a
4 ability
2 able
2 about
2 above
2 accept
3 according
2 account
2 across
1 act
2 action
4 activity
4 actually
1 add
2 address
5 administration
2 admit
2 adult
2 affect
2 after
2 again
```



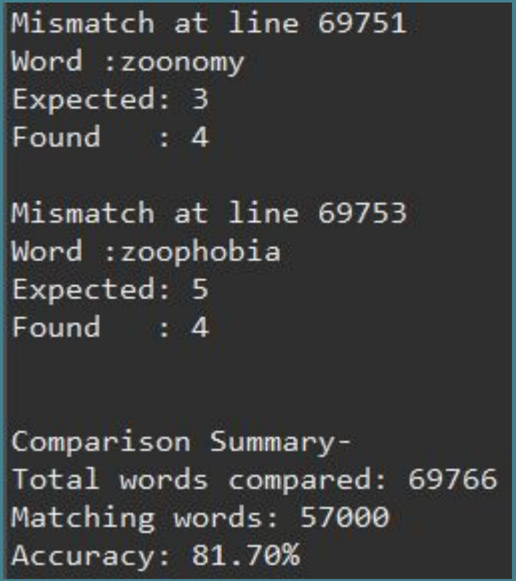
```
2 against
1 age
3 agency
2 agent
2 ago
2 agree
3 agreement
2 ahead
1 air
1 all
2 allow
2 almost
2 alone
2 along
3 already
2 also
2 although
2 always
4 American
2 among
2 amount
4 analysis
1 and
3 animal
3 another
2 answer
2 any
```



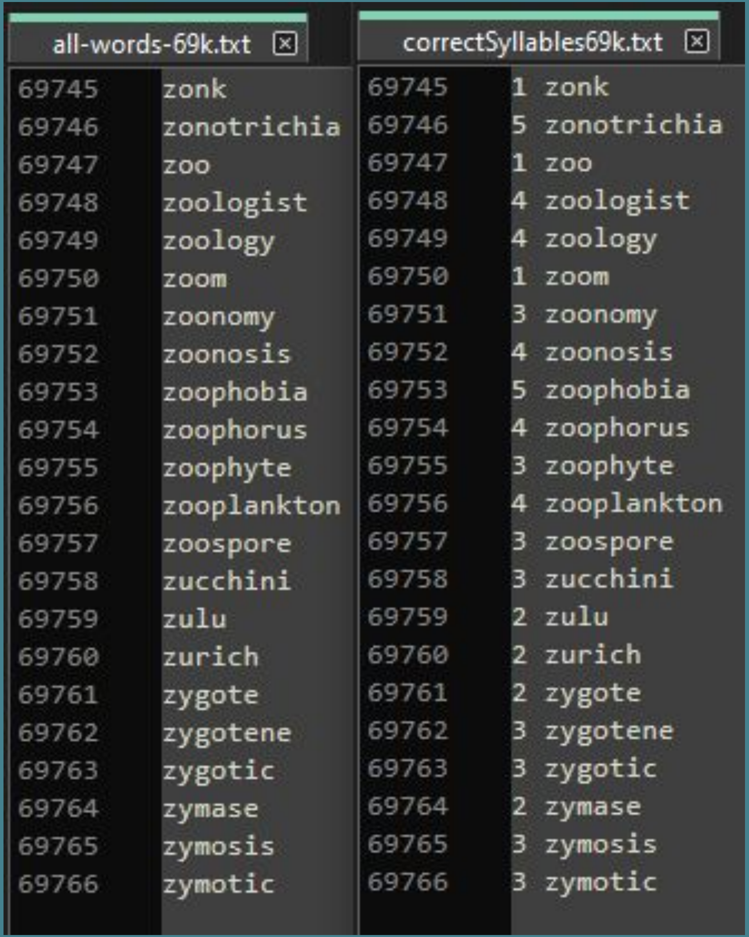
```
2 against
1 age
3 agency
2 agent
2 ago
2 agree
3 agreement
2 ahead
1 air
1 all
2 allow
2 almost
2 alone
2 along
3 already
2 also
2 although
2 always
4 American
2 among
2 amount
4 analysis
1 and
3 animal
3 another
2 answer
2 any
```

TESTING - Syllabification

Tested against dictionary file of over 69,700 words. Complex & uncommon words.



```
Mismatch at line 69751
Word :zoonomy
Expected: 3
Found   : 4

Mismatch at line 69753
Word :zoophobia
Expected: 5
Found   : 4


Comparison Summary-
Total words compared: 69766
Matching words: 57000
Accuracy: 81.70%
```



all-words-69k.txt

```
69745 zonk
69746 zonotrichia
69747 zoo
69748 zoologist
69749 zoology
69750 zoom
69751 zoonomy
69752 zoonosis
69753 zoophobia
69754 zoophorus
69755 zoophyte
69756 zooplankton
69757 zoospore
69758 zucchini
69759 zulu
69760 zurich
69761 zygote
69762 zygotene
69763 zygotic
69764 zymase
69765 zymosis
69766 zymotic
```

correctSyllables69k.txt

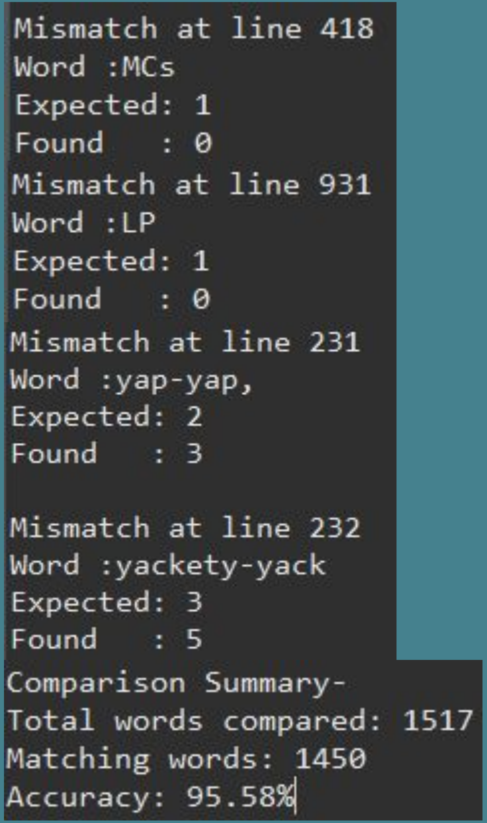
```
69745 1 zonk
69746 5 zonotrichia
69747 1 zoo
69748 4 zoologist
69749 4 zoology
69750 1 zoom
69751 3 zoonomy
69752 4 zoonosis
69753 5 zoophobia
69754 4 zoophorus
69755 3 zoophyte
69756 4 zooplankton
69757 3 zoospore
69758 3 zucchini
69759 2 zulu
69760 2 zurich
69761 2 zygote
69762 3 zygotene
69763 3 zygotic
69764 2 zymase
69765 3 zymosis
69766 3 zymotic
```

TESTING - Syllabification

Tested against Eminem's Rap God as additional challenge.

S.C. Does not handle syllables in acronyms or hyphenated words, this song has a lot of both.

Achieved 95.58% accuracy.



```
Mismatch at line 418
Word :MCs
Expected: 1
Found   : 0
Mismatch at line 931
Word :LP
Expected: 1
Found   : 0
Mismatch at line 231
Word :yap-yap,
Expected: 2
Found   : 3

Mismatch at line 232
Word :yackety-yack
Expected: 3
Found   : 5
Comparison Summary-
Total words compared: 1517
Matching words: 1450
Accuracy: 95.58%
```

TESTING - Syllabification

Tested against Eminem’s Rap God as additional challenge. Overall accuracy score average 95%, proving the algorithm to be highly competitive for Rule-Based system.

If work continues, adding code to further handle complex and edge cases.

TESTING - HaikuFinder

- Originally began using a “sliding window” method.
- Treats text as a continuous stream of syllables.
- Slides a "window" of three lines across the text.
- Checks if the syllable counts of the three lines match the 5-7-5 structure.

Efficient for large datasets since it avoids reprocessing the same lines.

Captures haikus even if they’re split irregularly across a text file.

Challenges associated:

- May miss haikus if line breaks are inconsistent.
- Can produce false positives with non-haiku patterns matching 5-7-5.

TESTING - HaikuFinder

Algorithm: Whole Line Farming

- Processes the text line by line, categorizing each line by its syllable count.
- Lines are stored in buckets based on syllable counts (e.g., 5, 7, or other).
- Constructs haikus by combining lines from the buckets to match the 5-7-5 pattern.
- Effective for well-structured text with clear line breaks.
- Easy to implement and extend with additional rules.
- This proved to be much better at producing Haikus with whole sentences.

TESTING - HaikuFinder

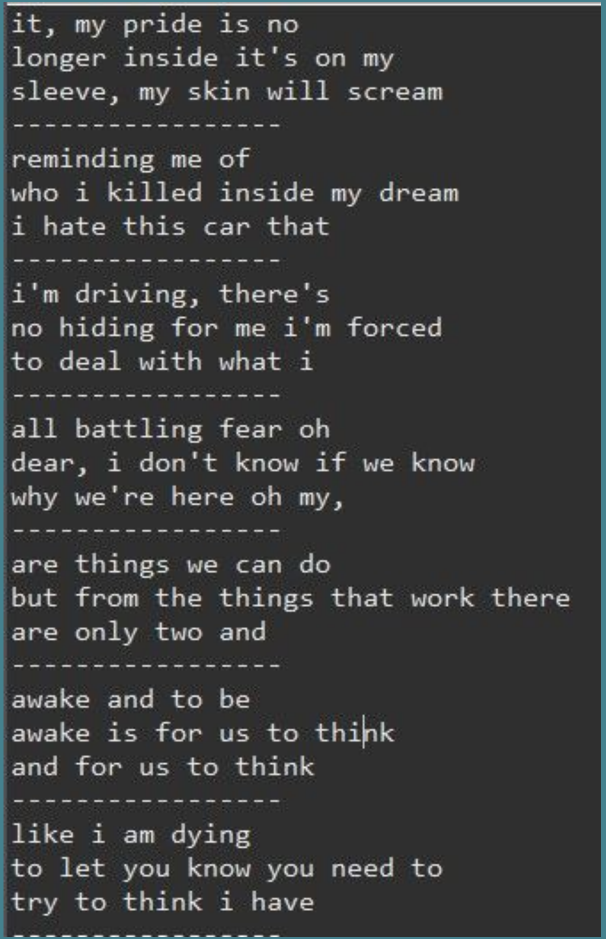
Algorithm: Whole Line Farming

- Processed lyric lines are fed into HaikuFinder class and sorted based on their syllable count into corresponding Arraylists<String>.
-

TESTING - HaikuFinder

Output before switching to the whole line farming algorithm.

Incomplete sentences, cut-off sentences.

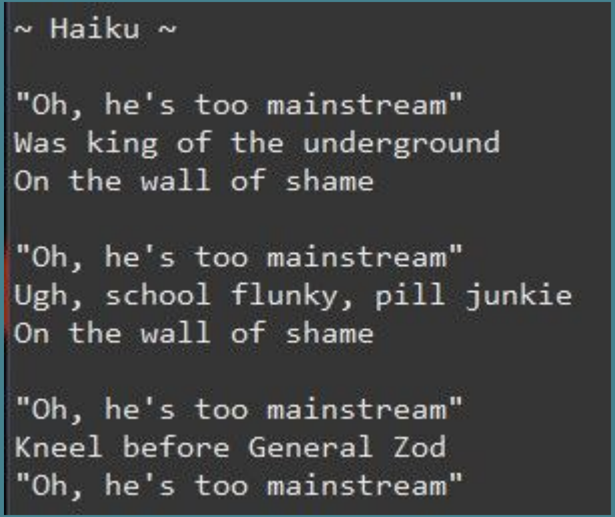


```
it, my pride is no
longer inside it's on my
sleeve, my skin will scream
-----------------
reminding me of
who i killed inside my dream
i hate this car that
-----------------
i'm driving, there's
no hiding for me i'm forced
to deal with what i
-----------------
all battling fear oh
dear, i don't know if we know
why we're here oh my,
-----------------
are things we can do
but from the things that work there
are only two and
-----------------
awake and to be
awake is for us to think
and for us to think
-----------------
like i am dying
to let you know you need to
try to think i have
```

TESTING - HaikuFinder

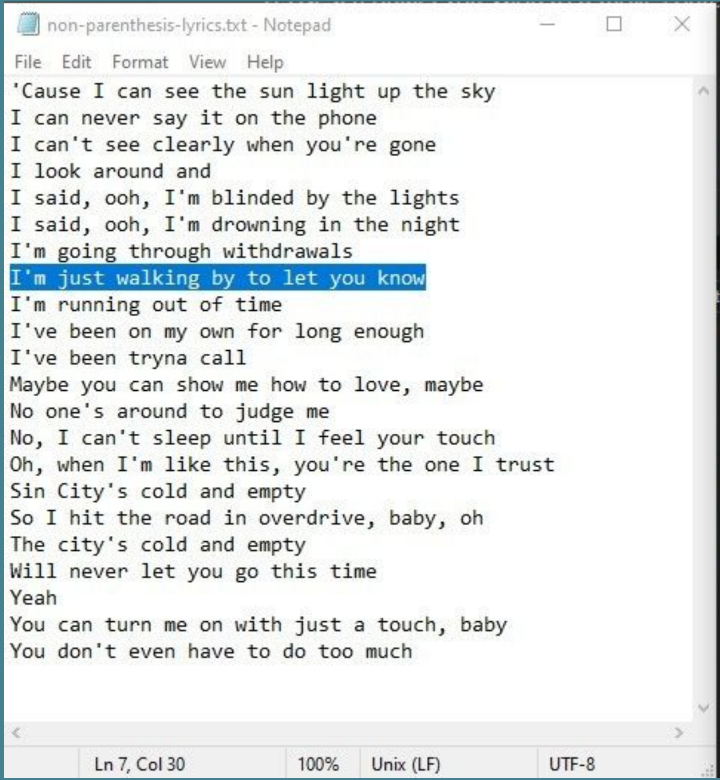
Output is much better, complete thoughts/sentences.

Then implemented a random line selection feature. All usable lines are stored for later use. If there are multiple usable lines found, generates a random number representing which line to use.



```
~ Haiku ~

"Oh, he's too mainstream"
Was king of the underground
On the wall of shame

"Oh, he's too mainstream"
Ugh, school flunky, pill junkie
On the wall of shame

"Oh, he's too mainstream"
Kneel before General Zod
"Oh, he's too mainstream"
```

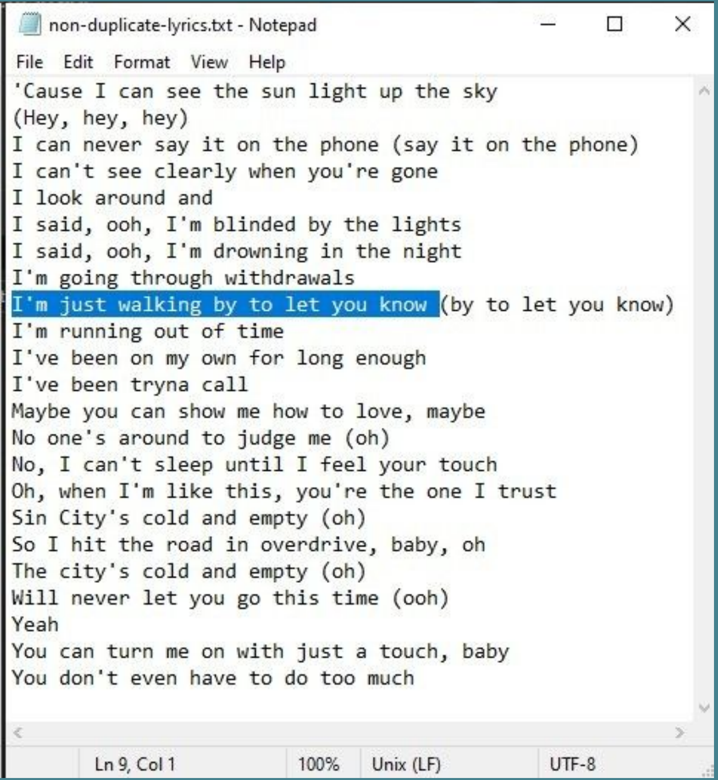
Processing Lyrics Files



non-parenthesis-lyrics.txt - Notepad

```
'Cause I can see the sun light up the sky
I can never say it on the phone
I can't see clearly when you're gone
I look around and
I said, ooh, I'm blinded by the lights
I said, ooh, I'm drowning in the night
I'm going through withdrawals
I'm just walking by to let you know
I'm running out of time
I've been on my own for long enough
I've been tryna call
Maybe you can show me how to love, maybe
No one's around to judge me
No, I can't sleep until I feel your touch
Oh, when I'm like this, you're the one I trust
Sin City's cold and empty
So I hit the road in overdrive, baby, oh
The city's cold and empty
Will never let you go this time
Yeah
You can turn me on with just a touch, baby
You don't even have to do too much
```

Ln 7, Col 30 | 100% | Unix (LF) | UTF-8



non-duplicate-lyrics.txt - Notepad

```
'Cause I can see the sun light up the sky
(Hey, hey, hey)
I can never say it on the phone (say it on the phone)
I can't see clearly when you're gone
I look around and
I said, ooh, I'm blinded by the lights
I said, ooh, I'm drowning in the night
I'm going through withdrawals
I'm just walking by to let you know (by to let you know)
I'm running out of time
I've been on my own for long enough
I've been tryna call
Maybe you can show me how to love, maybe
No one's around to judge me (oh)
No, I can't sleep until I feel your touch
Oh, when I'm like this, you're the one I trust
Sin City's cold and empty (oh)
So I hit the road in overdrive, baby, oh
The city's cold and empty (oh)
Will never let you go this time (ooh)
Yeah
You can turn me on with just a touch, baby
You don't even have to do too much
```

Ln 9, Col 1 | 100% | Unix (LF) | UTF-8

ETHICS OBSERVATION

Original concerns:

Explicit content

- Main concern: Explicit lyrics may be shown to underage users
- Solution: Utilize the explicit flag provided by Spotify
  - Users under 18 only receive haikus generated from songs without explicit lyrics
  - Users over 18 are allowed to toggle the “Allow Explicit Content” preference to allow explicit lyrics to be used in haiku generation
  - Data usage adheres to Spotify and Lyrics.ovh API guidelines and user consent for data retrieval is secured

ETHICS OBSERVATION

Updated concerns:

User Age Verification

- Initially, we considered implementing age verification to ensure compliance with content restrictions.
- Upon review of Spotify's user agreement, we found that age verification is covered in the terms of service:
    - Users must affirm they are 18 years or older, or 13 years or older with parental consent.

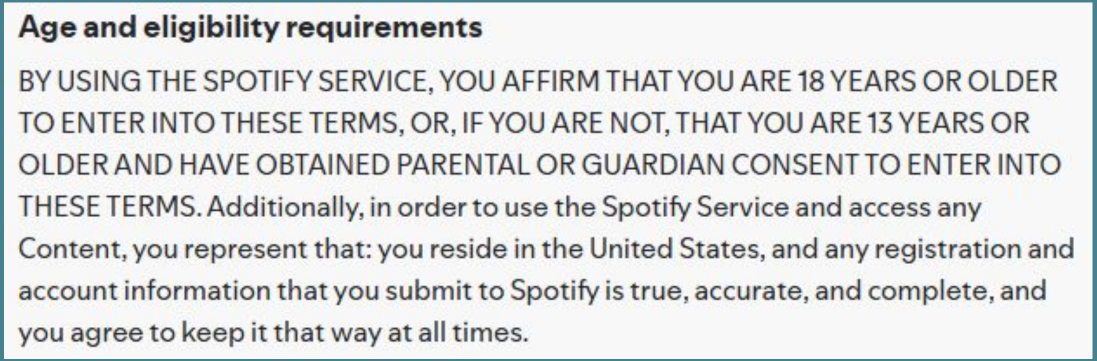
ETHICS OBSERVATION

Filtering Explicit Content

- Explicit content filtering was also deemed unnecessary as during account initialization, user is asked whether or not to allow the user to listen to songs with explicit content.
- Spotify's terms require users to adhere to its content policies, placing the responsibility on users and their guardians.

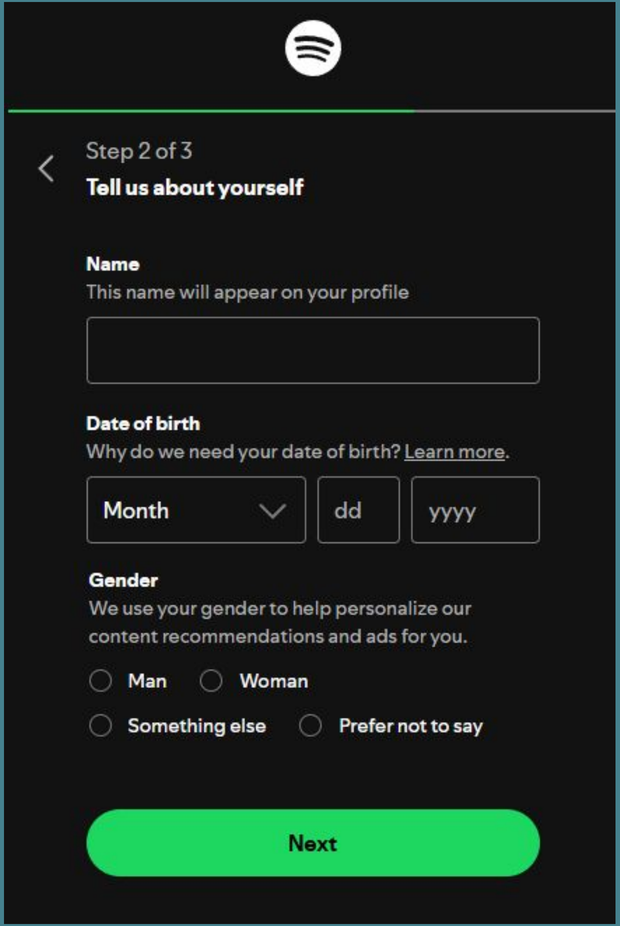
ETHICS OBSERVATION

Screenshots taken from Spotify



Age and eligibility requirements

BY USING THE SPOTIFY SERVICE, YOU AFFIRM THAT YOU ARE 18 YEARS OR OLDER TO ENTER INTO THESE TERMS, OR, IF YOU ARE NOT, THAT YOU ARE 13 YEARS OR OLDER AND HAVE OBTAINED PARENTAL OR GUARDIAN CONSENT TO ENTER INTO THESE TERMS. Additionally, in order to use the Spotify Service and access any Content, you represent that: you reside in the United States, and any registration and account information that you submit to Spotify is true, accurate, and complete, and you agree to keep it that way at all times.



Step 2 of 3

Tell us about yourself

Name

This name will appear on your profile

Date of birth

Why do we need your date of birth? Learn more.

Month | dd | yyyy

Gender

We use your gender to help personalize our content recommendations and ads for you.

Man Woman Something else Prefer not to say

ETHICS OBSERVATION

Copyright issues

- Main concern: Ensure using lyrics from songs fall into fair use
- Requirements to ensure fair use:
  - Receive no profit from lyric use
  - Clearly state where lyrics are taken from
  - Transformative use, if the meaning of the lyrics changes from the original content it is more likely to be considered fair use (creating haikus' from the lyrics should be considered parody and therefore be considered fair use by copyright law.)

ETHICS OBSERVATION

Most of our obstacles were related to logistics-

For example:

- Finding solutions to completing certain requirements
- Connecting the frontend and backend
- Hosting the backend on a public server so that the app would be accessible to all (could not achieve this in time).

PROJECT DOCUMENTATION

Planned documentation:

- Technical Documentation - Detailed description of the system architecture, including the flow of data between components (APIs), Java classes, and the GUI, how the APIs are used, and explanation of key algorithms.
- Requirements Documentation - Functional and non-functional requirements.
- Test Documentation - Detailed descriptions of the test plan, the testing strategy, cases that were tested for, steps, expected results, actual results, and summary.
- Version control - How git was employed for source control.
- Licensing - Open Source Licensing & API licensing
- Project summary/Final report

PROJECT DOCUMENTATION

Final documentation:

- Installation and Execution README file
- Software Development Report
- Project Test Report
- User Manual
- Developer Documentation
- License Documentation (MIT)
- Requirements Spreadsheet
- Project Schedule
- Test Plans

PROJECT COMPLETION

Our completed project is a web app that is only currently only accessible through local set-up.

1. User Interface
   - Consists of two large gray boxes, one which acts as a window to display information, and another that contains buttons relating to the user’s account and haikus.
2. Spotify authorization and haiku generation flow
   - Leads the user through the process of authenticating their Spotify account, generating haikus, and viewing them in the display widow.
   - The app is currently still in developer mode, where only pre-confirmed Spotify accounts (up to 25) can utilize Kigo.

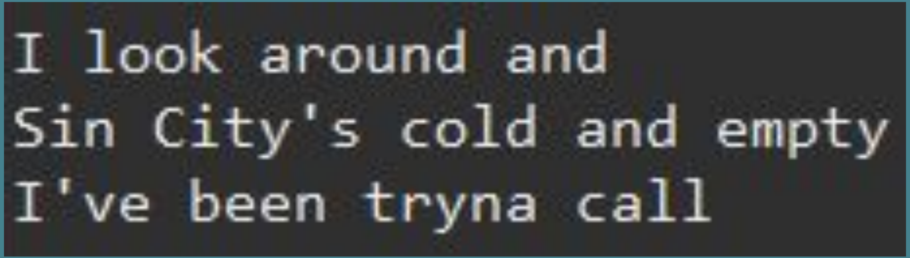
PROJECT COMPLETION

While the project meets all essential functional requirements, there are still some limitations:

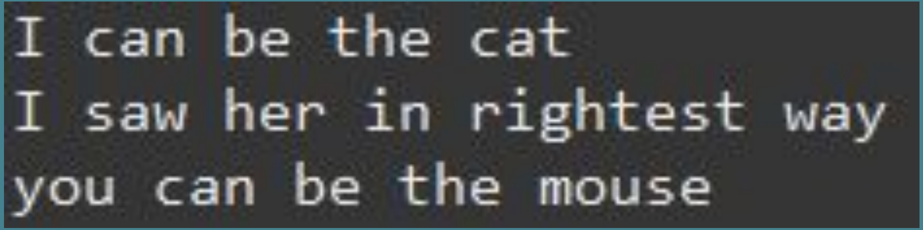
- Dependency on external APIs
- Limited compatibility with streaming platforms
- Insufficient user-facing error handling
- Missing (non-essential) feature implementation

These limitations do not significantly impact the core functionality of the app but may negatively affect user satisfaction and overall usability. These issues should be prioritized in future updates.

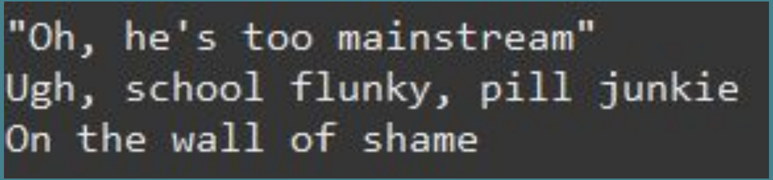
Wow, great job!!



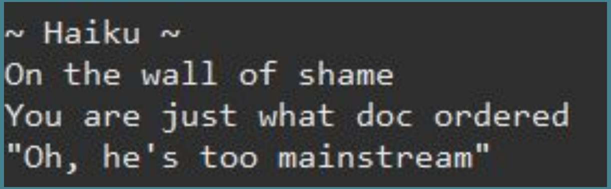
```
I look around and
Sin City's cold and empty
I've been tryna call
```



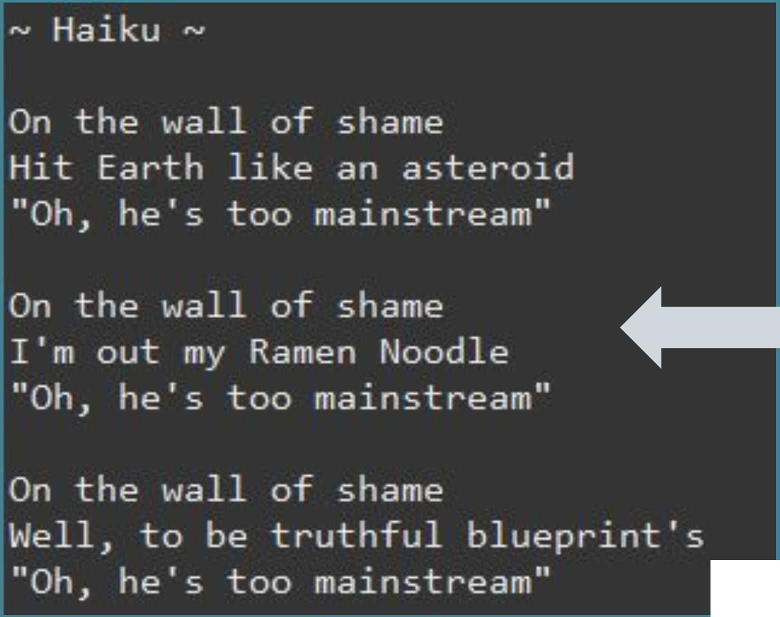
```
I can be the cat
I saw her in rightest way
you can be the mouse
```



```
"Oh, he's too mainstream"
Ugh, school flunky, pill junkie
On the wall of shame
```



```
~ Haiku ~
On the wall of shame
You are just what doc ordered
"Oh, he's too mainstream"
```



```
~ Haiku ~

On the wall of shame
Hit Earth like an asteroid
"Oh, he's too mainstream"

On the wall of shame
I'm out my Ramen Noodle
"Oh, he's too mainstream"

On the wall of shame
Well, to be truthful blueprint's
"Oh, he's too mainstream"
```

Wow, great job!!

Ramen Noodle Haiku = bonusPoints++;