SoundByte; Suggesting Music for DJs & Hobbyists

University of Regina | Faculty of Engineering & Applied Science

Brandon Clarke, Jiwoun Kim, Mason Lane
200373287, 200329205, 200376573

Introduction

In the year 2021 you're likely no stranger to being recommended music by some sort of software application, be it through social media or a digital media streaming platform. SoundByte aims to recommend songs for the sake of music creation, it does this by leveraging musical features that SoundByte extracts from the user's library. Many popular services that recommend songs to its users typically use very subjective features. SoundByte challenges the norm by using objective features to fully place the power in the hands of the creator. SoundByte's use of musical features relinquishes creative control to the artist while providing auxiliary guidance

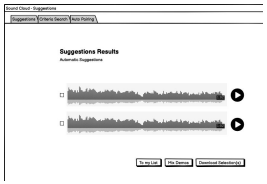
Aim

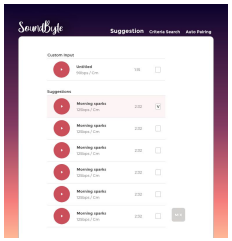
In Software Systems Engineering one of the fundamental practices is to determine the context and scope of proposed software before any work begins on the software design. With that being said our app is given its context and scope through its users. These users are DJs, musicians, content creators, hobbyists and even the average music enthusiast. SoundByte aims to organize a user's music library in a fresh light and provoke new ideas on how the user's music can be used through suggested songs that would make for a good music sample, mix, mashup, or fade out.

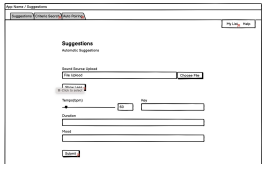
Research & Design

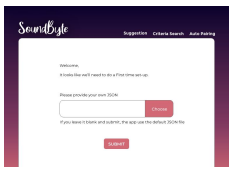
Beginning with our problem, we conducted empathy mapping, affinity diagrams and user stories. This was to ascertain what the user would need most to enable their workflow & achieve our project's goal. Doing this, we were then able to develop a general storyboard - which we would use for planning our roadmap on Jira. We also did stakeholder analysis and risk management documentation as we sought out requirements. Completing this stage, we finally took time to write a formal document for our business case and project charter.

Next, we began planning for the architecture of our application. This included a collection of Lofi & Hifi diagrams for the user interface(UI)

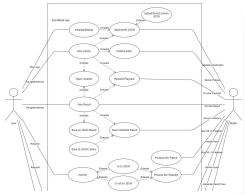








as well as class, activity, use case & MVC detailed much of the communication between backend and frontend.



For the development stacks, we planned to work with AWS. We wanted to support playback for our suggested music - something we could not do on AWS. Therefore, we opted for a desktop application as it would handle more songs in the user's possession - and most importantly, stay in their possession.

Research included how we would suggest songs. Searching the web revealed a dataset called million songs dataset. The technology that was used to create the bulk of this dataset was bought by and is presently used by spotify, so it is a pretty good place to start. The one problem was that the data consisted only of the extracted features leaving it to us to source the associated song files. The previously proposed design of training a model on tensorflow was not feasible at this point. This hurdle lead to more research and the discovery of Essentia which included a variety of options for feature extraction.

Method

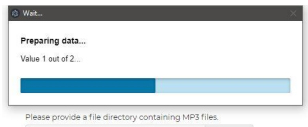
Our decision to use Electron was based out of familiarity with the framework, and also a desire to leverage node modules on NodeJS. Ultimately, our application used a mixture of NodeJS and Python to handle backend business logic. Our frontend consisted of JS, TypeScript, CSS and HTML.

We used JSON files to store song objects - which consisted of extracted feature data. These would be used in our suggestion calculations.

To extract feature data, we worked very hard to leverage EssentiaJS. EssentiaJS is capable of estimating & extracting a plethora of audio features. For our initial purposes, we used it to extract Key, Scale, and BPM. There were some hurdles however; EssentiaJS uses very large WASM files, which may not be larger than 4kb. This is a rule used in Chromium to keep the Document Object Model (DOM) from being held up. In order to use EssentiaJS, we needed to implement web-workers which would leverage multithreading. In these cases, WASM files may be larger. To our luck, Electron offers node module support for its web-workers; something you cannot easily do with just NodeJS. With these components in place, it was a trivial matter of decoding WAV files, and sending the data to our web worker. We used TuneBat to compare Essentia's predictions with its actual values - finding it to be exceptionally accurate.

We also used a variety of NodeJS modules to aid EssentiaJS in this task. Firstly, we use a module for reading WAV tags. From this, we may derive information including the song's title and author. Next, we use a module for reading song length from WAV files - we choose this over reading length tags, as we can guarantee a value. We then use a WAV decoder to send data to our EssentiaJS web worker.

Finally, we use a module for our loading bar in electron. This is to inform the user of the reading state.



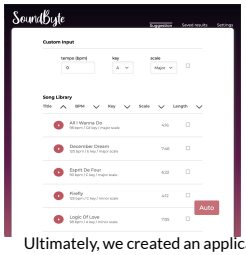
Extracted data is stored as Song objects, which contain a song metadata object (for storing Key, Scale, BPM).

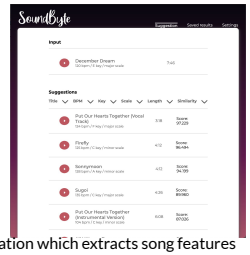
Much of our middleware & EssentiaJS backend is written in TypeScript.

We elected to use TypeScript as it allowed us to follow a more object-oriented design in line with our original architecture. This allowed us to maintain Song objects, Suggestion objects, etc. You will find that our suggestion & extraction code attempts to use inheritance & varying design patterns to reduce the amount of repetitive code. In our design we have made efforts to follow the open-closed principle in regards to our types and suggestion logic.

Once features had been extracted from the song library and stored within a JSON file in the application the next step was to put this data to work. These features are leveraged in one of two ways. Firstly, the user may pass Song objects from their library to receive suggestions. Secondly, the user may create a Feature object from user-defined features and pass it to suggestion algorithm. Once the error function receives this Feature object it iterates over the users song library and develops a measure of similarity. The library of songs is passed back from the error function with each song containing its respective measure of similarity. Note that we use a python-shell with NodeJS to communicate between Python and NodeJS. This list of songs is then built into a ResultsData object and passed to the front end to be displayed, sorted and previewed. A Suggestion object contains input & Result objects. This was designed to associate input with output.

Results





Ultimately, we created an application which extracts song features within a decent margin of error - thanks to the use of EssentiaJS. Our suggestion algorithm is capable of suggesting a similarity percentage - but it is need of a more thorough weighting system. Furthermore, our application includes features we deemed most integral to the usability of the application, including random suggestions, suggestions by song, suggestions by feature, view filtering & suggestion scoring.

Here, users may select a folder of WAV songs (it may be a folder of mixed files - we simply read WAV only). From there, we await a series of web workers to propagate our library json data with Song objects. These web workers use EssentiaJS to extract our most important features (Key, Scale, BPM) . Once finished, we direct the user to the main library menu; here they may listen to music, sort by various characteristics or request suggestions.

Suggestions come in three forms, and return a Suggestion object. A Suggestion object contains both our input & its results.

Our application also includes various exception handling & restrictions for user input - as well as a setting menu for re-initializing a library.

Hurdles

In the front-end side, the Electron framework had some learning curves being different from native Javascript; such as navigation between menus. Furthermore, being a desktop application meant that we needed to test directory navigation for Linux & Windows.

Initially trying to implement EssentiaJS on Electron/NodeJS was a difficult task with its WASM sizes. Luckily, we were able to leverage web workers for this task and ultimately benefited from EssentiaJS.

During development, we also endeavored to include content security policies - but our use of WASM files forced us to allow unsafe-eval during our initialization component. This is not an error, but also not good practice. We have yet to find a solution.

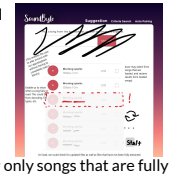
We did cut some feature creep from our original design. This is also in favor of changes we deem have greater priority.

Moving Forward

One problem we have not been able to surmount is the time required for feature extraction. In our tests of ~10 songs, we found that the initial setup held the DOM for too long.

Currently, our applications wait for a web worker to finish. It is better practice for our web workers to run synchronously with the application. what we propose is the following:

- Read tag data from songs, save to objects and json.
- Direct to library.
- If a Song object is not fully propagated, read the song's WAV into EssentiaJS. Refresh the DOM with full Song objects
- If a song fails to read, update the DOM with a failure explanation. Allow the user to delete or re-read the song
- Allow the user to remove or update or add individual songs
- meanwhile, allow user to interact with the DOM (play, pause, suggest)
- Requests for suggestions will consider only songs that are fully read into our system. Partial, awaiting or failed songs will not.



Conclusion

We are quite happy with the revelations made along the way, but are also eager to explore improvements in the future. As an MVP, we achieved the majority of our epics- but can see now the many improvements our application may benefit from in future iterations.

In retrospective, our design of data types, suggestion models & introduction of web-workers has enabled our application to evolve in future iterations. It would not be hard to introduce the aforementioned improvements.

Acknowledgements

For advisory & general mentoring:

- Kin-Choong Yow. Ph.D. SSE
- Yasser Morgan. Ph.D. SSE
- Timothy Maciag. Ph.D. SSE

Node Modules & Dependencies:

electron	- MIT License
electron-progressbar	- MIT License
essentia.js	- AGPL-3.0 License
get-audio-duration	- MIT License
music-metadata	- MIT License
python-shell	- MIT License
typescript	- Apache-2.0 License
wav-decoder	- MIT License