MoodList: An Application for Matching Text to Spotify Playlists

Allen Abraham *New York University*
aa10770@nyu.edu
Erin Miller *New York University*
em4919@nyu.edu
Ohm Patel *New York University*
odp2008@nyu.edu
Sriram Teja Kora *New York University*
sk9944@nyu.edu
Yubei Tang *New York University*
yt696@nyu.edu

***Abstract*—MoodList returns Spotify playlists personalized to text uploaded by the user, based on keyword extraction and mood.**

***Index Terms*—keyword extraction, emotion detection, Spotify playlist**

I. Introduction and Motivation

Spotify has over 4 billion playlists. It would be extremely time-consuming for the user to sort through billions of playlists. Additionally, sometimes we don't know what kind of music we're in the mood for. *MoodList* is a project that takes user input in the form of a snippet of text, and generates a playlist based on keywords and mood.

II. Codebase and Demo

View app here and codebase here. A video demo is also available here.

III. User Experience

On the front page, you will be directed to log into your Spotify account. Once you log in, you will be prompted to enter a message in the textbox. When you hit generate, you will be taken to another page that displays five embedded Spotify playlists based on search terms generated from the message you entered. You can see previews of each playlist, and save the playlist to your MoodList account. At this point, you can either go back to the homepage to generate more playlists, or visit your user page to see saved playlists. On the user page, clicking on the saved playlists also allows you to see the original text that generated the playlist suggestion. Users are also sent a copy of the playlists to their emails.
Note: You will notice that some playlist embeddings show request timeouts when iframes have not loaded. This is due to a timeout by Spotify and can be random. It can be mitigated to some extent by code optimizations on our side, but it is largely dependent on Spotify API's functionality.

IV. Tech Stack

AWS Technologies Used: S3 Buckets, IAM, Lambda, DynamoDB, API Gateway, Simple Queue Service (SQS), EKS (Elastic Kubernetes Service), Amazon Comprehend, SES, Amazon Load Balancing
Other Frameworks and Technologies: Spotify API, Python, Docker, HTML/CSS, Javascript (React, NodeJS), OpenAI API
Technologies Considered: Sagemaker

V. Architecture

A. Front End

User provides text, such as poem or messge, describing their current mood or feelings. Max length 500 words.

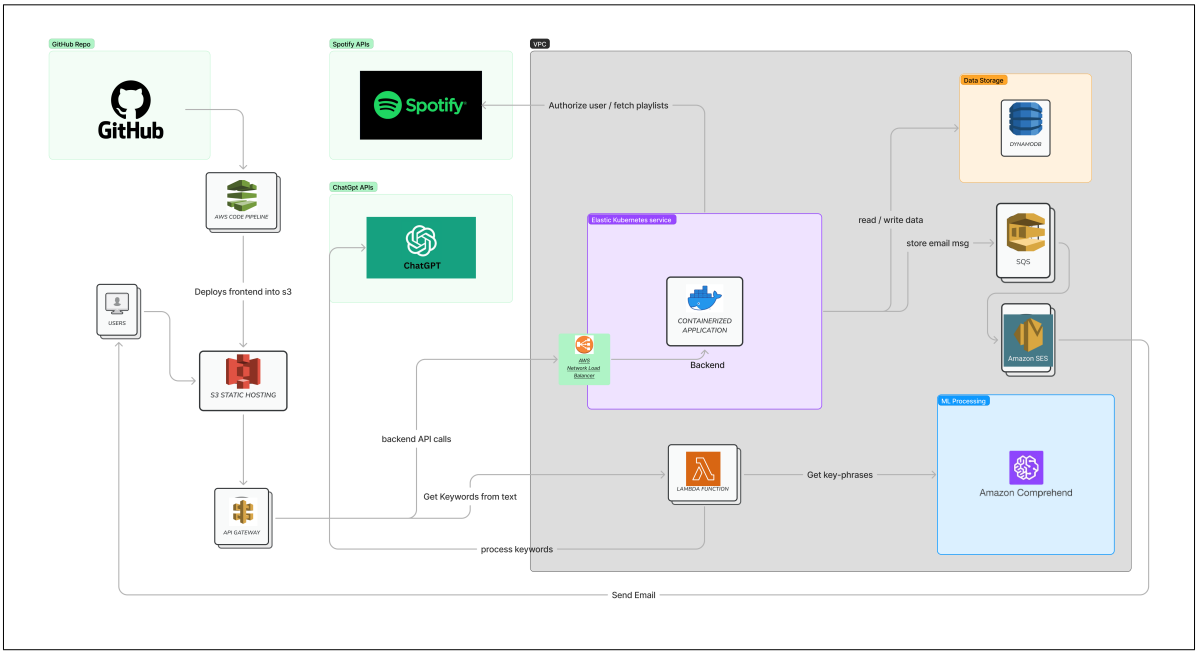


Fig. 1. Architecture diagram

The front end is built using React. We chose React because our team members were most familiar with the technology, and because Spotify's web is built on React, which allows for easy integration.

B. Backend

Express/Node.JS application containerized in EKS. It's responsible for all authenticationa and communication with Spotify, as well as data handling with DynamoDB.

C. AWS Load Balancer

We are using an AWS network load balancer (different from the default Classic load balancer) which works on the 4th layer of the OSI Layer.

D. API Gateway:

Facilitates communication between the front end and our back end cluster. It also captures the text through an endpoint and API Gateway exposes a REST API that takes in the message.
We use Postman to help with API development and request testing.
API Gateway is also used as a Proxy to the kubernetes cluster. API Gateway helps with throttling, parameter caching and rate limiting our APIs. This ensure that the Kube cluster is not doing redundant work and can be safeguarded against DDOS attacks.

E. Kubernetes

Backend services are deployed and managed within Kubernetes.
We have a Kuberneters cluster of 2 t4g.medium Nodes. Each pod is configured with liveness and readiness probes to monitor for health and readiness. The AWS NLB also uses this health checks to route traffic between the Autonomous Zones ensuring High Availability.
We are using Kube secrets to store our sensitive information like API credentials and database keys. This is later read by the Application from the Environment variable.

F. Docker

We are running (Docker) containerized application in the Kubernetes cluster.

G. User Account and Playlist Management in DynamoDB

DynamoDB stores two tables: users and playlists. The Users table stores UserID (Primary Key), username, authentication code, and access token. Playlist stores userID, playlist link, date of creation, text associated with playlist.

H. Amazon Comprehend

User input text is sent directly to Amazon Comprehend for processing. Amazon Comprehend outputs a string of space-separated keyphrases.

I. OpenAI API for Mood Detection

Key phrases are passed to OpenAI, which is asked to generalize an emotion.

J. Spotify API

Spotify API is used for user log-in. We also use Spotify API's Playlist and Search features to sort through playlists and embed them on our pages.

K. Simple Queue Service (SQS)

Emails are placed in a message queue using SQS.

L. Simple Email Service (SES)

An email is automatically sent to the user when the playlist results are generated.

M. Lambda

We have a lambda function that receives key phrases from Amazon Comprehend and sends them to OpenAI API.

N. S3 Buckets

Front end files are stored in S3 buckets.

O. User Tokenization and Authentication

Since users log in with their Spotify credentials, we use Spotify Authorization features from the API to handle log-in.

P. Code Pipeline

Finally, we have a code pipeline that automatically deploys our front end to the S3 bucket and posts code changes to our Github.

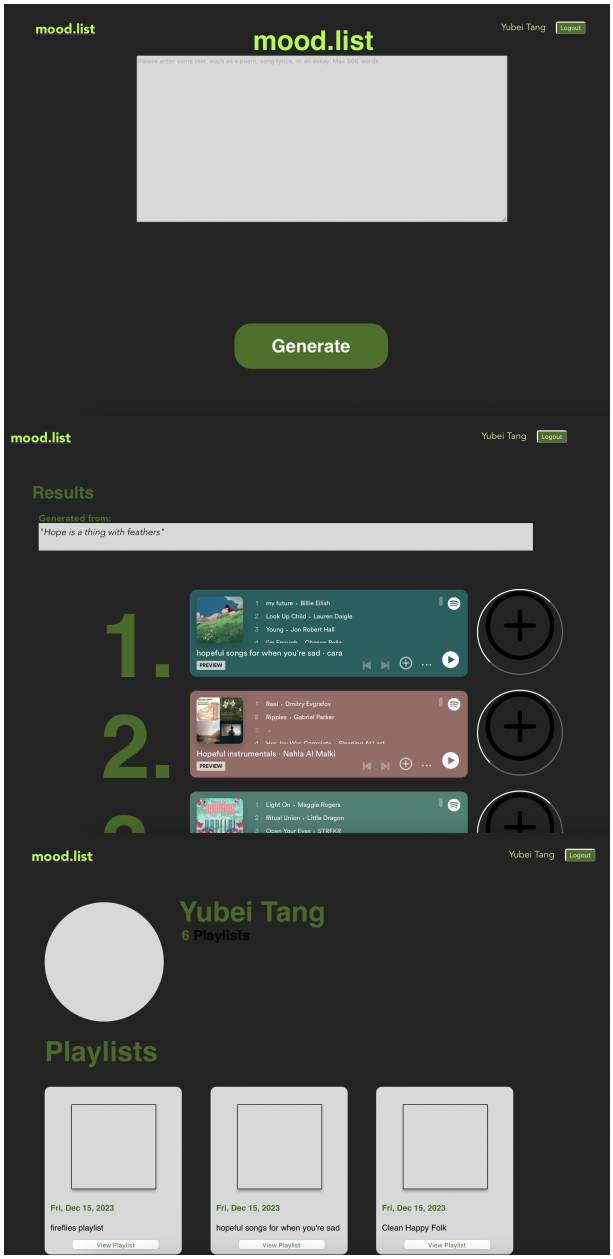
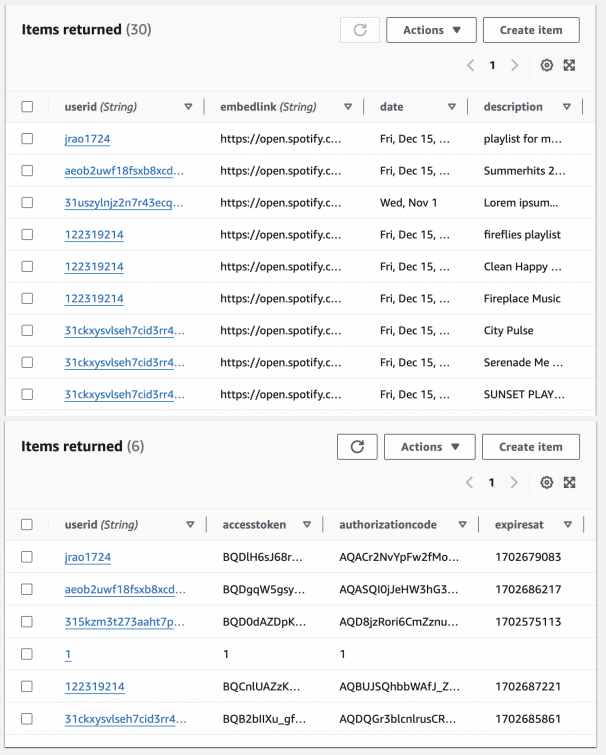


Fig. 2. MoodList application. From top to bottom: (1) Front page after logging in (2) Results page after entering text. (3) Account details with saved playlists.

VI. Design Choices

A. Capturing the Mood of a Piece of Text

For this project, we decided to pass on an emotion as a search term for playlists. Originally, we proposed passing on a mix of keywords and one emotion to Spotify. We ran keyword extraction and emotion classification, categorizing text into one of six basic emotions. However, we used an alternative solution for the following reasons:



Items returned (30)

Actions | Create item

userid *(String)*	embedlink *(String)*	date	description
jrao1724	https://open.spotify.c...	Fri, Dec 15, ...	playlist for m...
aeob2uwf18fsxb8xcd...	https://open.spotify.c...	Fri, Dec 15, ...	Summerhits 2...
31uszylnjz2n7r43ecq...	https://open.spotify.c...	Wed, Nov 1	Lorem ipsum...
122319214	https://open.spotify.c...	Fri, Dec 15, ...	fireflies playlist
122319214	https://open.spotify.c...	Fri, Dec 15, ...	Clean Happy ...
122319214	https://open.spotify.c...	Fri, Dec 15, ...	Fireplace Music
31ckxysvlseh7cid3rr4...	https://open.spotify.c...	Fri, Dec 15, ...	City Pulse
31ckxysvlseh7cid3rr4...	https://open.spotify.c...	Fri, Dec 15, ...	Serenade Me ...
31ckxysvlseh7cid3rr4...	https://open.spotify.c...	Fri, Dec 15, ...	SUNSET PLAY...

Items returned (6)

Actions | Create item

userid *(String)*	accesstoken	authorizationcode	expiresat
jrao1724	BQDlH6sJ68r...	AQACr2NvYpFw2fMo...	1702679083
aeob2uwf18fsxb8xcd...	BQDgqW5gsy...	AQASQI0JJeHW3hG3...	1702686217
315kzm3t273aaht7p...	BQD0dAZDpK...	AQD8jzRori6CmZznu...	1702575113
1	1	1	
122319214	BQCnlUAZzK...	AQBUJSQhbbWAfJ_Z...	1702687221
31ckxysvlseh7cid3rr4...	BQB2bIIXu_gf...	AQDQGr3blcnlrusCR...	1702685861

Fig. 3. Top, Bottom: (1) DynamoDB Playlists Table (2) DynamoDB Users Table

1) This did not yield the best results. Even though the model was accurate on new data, we found the emotion classifier did not pick up nuances of text.
2) We found keyword search did not pick up very good playlists. Also, for short snippets of text (less than 500 words), keyword extraction is not an effective tool simply because there aren't enough words for NLP techniques to process for relative importance.

Instead, we decided to use Amazon Comprehend to extract key phrases, which are sent to OpenAI to detect more fine-grained emotions such as "loneliness" or "inspiration." We had the extra step of reducing text down to a few key phrases to OpenAI in order to compress our data and reduce latency. Although the reduction in time was not significant, we built in this feature keeping in mind that it can handle larger texts, such as a book.

B. Back End

We opted for two nodes in Kubernetes since our app is still small. We decided to use Kubernetes because having multiple nodes means resource division is automatically handled by our nodes and we can easily scale up our system if needed.
We used AWS Load Balancer to distribute request traffic.

VII. Challenges

A. Costs

Costs and billing were something we all kept in mind throughout the project. One reason we decided against using SageMaker was because we needed a notebook instance with a GPU, and that meant running up costs by the hour. We ended up finding a cheaper solution by linking with OpenAI.

B. Latency

Originally, we wrote more lambda functions to handle keyword extraction and passing data between endpoints, but we found that this slowed down our system quite a bit. Particularly hosting a lot of NLTK modules on lambda led to about 10 or more seconds of processing time.

C. Complexities

Even though AWS services as designed to work together, as the number of services expanded, there were more difficulties involving integration. One challenge was integrating the Load Balancer with the API Gateway. To integrate our API Gateway with a the Load Balancer, we used API Gateway HTTP integration. However, this requires that our backend be external facing, which could lead to security issues. This seems to be an AWS-specific quirk, and as we will mention later, the solutions to this problem are costly and difficult to implement.

VIII. Changes from Proposed Architecture

We originally proposed two neural networks to extract keywords and detect emotions in the user-submitted text. However, this did not make it into our final project, because we realized linking with the ChatGPT API provided a faster and more accurate solution.
Initially, we explored Amazon Sagemaker, which handles deployment of machine learning models. We found that Sagemaker's pre-built models did not fit our goals very well, so we opted to build a custom model. However, we had some trouble integrating this into Sagemaker, because a lot of

pre-built containers were either incompatible with our TensorFlow model or deprecated. One option was to build our own container for our model in Docker. However, by this time, we were close to the deadline, and felt that the When we explored keyword extraction, we tried out three libraries: `NLTK`, `YAKE`, and `BERT`. All three libraries produced very similar results. We decided that since we were submitting short pieces of text, we did not need a cutting-edge algorithm. We were able to incorporate keyword extraction into our project by writing a Lambda function, but ultimately decided to use Amazon Comprehend and ChatGPT as they were better able to match the tone of the text.

We built a supervised learning model in Keras/TensorFlow to classify text into 6 emotions: sadness, anger, love, joy, surprise, fear. The model had 99.8% accuracy after running for 10 epochs. After tokenizing the text, we applied one-hot encoding to remove ordinal context. Since the model was adapted from another text classification model, we reused a lot of the same hyperparameters: including 3 hidden layers, ReLu, SoftMax, and Adam optimizer. Although we felt this was a very good model when we tested it ourselves, we were unfortunately unable to incorporate this model in our final project (for reasons explained earlier).

We also decided not to generate songs for a playlist, because we felt that Spotify's existing library of user-curated playlists provided a better listening experience than a randomized playlist, albeit one that takes into account the user inputs. Our final product ended up being a tool for sorting Spotify playlists, rather than generating new playlists.

IX. Other Work

As far as we know, there are no commercial or major applications that try to match short pieces of text to match the tone and feel of a short message. MoodPlaylist will automatically generate a playlist based on one mood and taking into account the user's music tastes. (Note: MoodPlaylist was formerly Moodika, and seemed to have changed its name in the middle of our project!) Playlistable is a paid service that generates playlists based on short snippets of text that describes an action like "dreamy lazy afternoon" and "lost in a digital cyberpunk future." These texts are then mapped to genres like "Lo-Fi" or "Indie Folk" and the playlist is generated based on these genres. We think Playlistable has a similar idea to ours, which is performing basic sentiment analysis and keyword extraction.

We do believe that as AI-generated text and artworks become more popular, we will see more applications like this, such as generating an artwork from a playlist and vice versa.

X. Extensions of Work

A. Monitoring and Debugging

We would set up a central monitoring dashboard where we can view logs of pods running inside a Kubernetes cluster. This makes it much more convenient to check pod status (as opposed to running a command). We would set up a Elasticsearch cluster so that logs produced by the Kube pods would be pushed to the OpenSearch DB to be used for Monitoring and debugging.

B. Matching Songs to Text

We would look more into SageMaker's capabilities or how to deploy a custom machine learning model. Our current model summarizes text as keywords and sentiments, but we could develop a more sophisticated model to map text to music that would take into account genre and other audio features, such as danceability and instrumentalness.

C. User Preferences

We could take into account user's saved artists and song history and incorporate that into the playlist search. The Spotify API gives developers access to user's listening history, so it would not be a difficult feature to add.

D. User Features

We could add more features to the front end, such as a community feature where you could see most popular types of searches. This would also not be a difficult feature because our database already stores keywords, so we could pull trending keywords at a certain time.

E. Security

As mentioned earlier, we had an external facing backend in order to integrate API Gateway with Load Balancer. One solution we came across was setting up a private integration, which requires a Network Load balancer, and keeping our backend locked down in a virtual private cloud.

ACKNOWLEDGMENT

This project was completed as a requirement of Cloud Computing at NYU. Thanks to Sambit Sahu, Taha Junaid, Shashakt Jha, and Geetika Bandlamudi for their help and mentorship.

REFERENCES

[1] S. Srivastava, "Set up a monitoring dashboard to view your Kubernetes pod logs"
[2] A. Wittig, Amazon Web Services in Action, 3rd ed: Manning, 2023.
[3] Anonymous, "Emotion Classification on Twitter Data Using Transformers"
[4] Anonymous, "Keyword Extraction Methods from Documents Using NLTK"
[5] N. Van Otten, " How To Implement Keyword Extraction [3 Ways In Python With NLTK, SpaCy & BERT]"
[6] Spotify for Developers
[7] https://openai.com/blog/openai-apiOpenAI API
[8] Build an API Gateway REST API with HTTP integration
[9]