LyricCheck

Technical Report

PREPARED BY
Hassan Khan
Stacy Jong
Jack Si
Kenji Nakachi
Christopher Carrasco

Motivation

This website consolidates data on top artists (monthly listeners), charting songs, artist bios, song lyrics, and a profanity score. Using Perspective's NLP model to grade song lyrics, our project will also provide insight on the profanity preferences over time, based on location, and based on artists. It will additionally have a Spotify playlist analyzer that will grade the profanity level of Spotify playlists, allowing for a bulk solution to children-safe songs. There are sites that tackle one or even part of one responsibility that we hope our project will accomplish, but there are severe patches of information missing from each. By also making use of a profanity grade, our project can provide valuable insight into the worrying NSFW trend of modern-day pop culture.

User Stories

Phase I User Stories for Developer

- Page Explaining Good Statistics for a Team or Player

  "Hi, this is Chris from the customer team. As someone who doesn't know much about the statistics of a football team or player, I would like there to be some page explaining what is a good score for some attribute of the team or player and why. This way I can look at the statistics and understand just how good or average they might be."
- Chart of the Best Overall Players or Statistics

  "I am also someone who doesn't know too much about football stats. I would like to see some kind of chart of the best overall players or best # yards ran just to put into perspective some of the stats. Something to highlight what's a good number in certain stats."
- See How a School's Offered Salary Affects Win-Loss Ratio

  "As an addendum to addressing the question of whether salary affects coach's win-loss stats, it would be interesting to see how a school's offered salary affects the school's win-loss stats. If this could be implemented as a graph over time, that would be thought-provoking on the user's end. This would primarily be targeting schools' hiring committees as stakeholders."
- Text Popups to Explain Football Terms

  "I am a user who knows absolutely nothing about football whatsoever. While I was navigating the website, I found myself confused with some terms, such as Player positions (offensive tackle, defensive end...), Bowl Game, and others. Maybe some small text popups would be a nice way to show some information."
- A Program Endowment and Ticket Sales Attribute for the School Model

  "As your project is focused on the possible correlation between pay/tuition and win percentage, it would be beneficial if there is a program endowment and ticket sales attribute for the School model. There is already tuition, but program endowment is a more direct connection to the amount of financial freedom a school's football team has. As for ticket sales, it can be indicative of the support of the football team on campus."

Phase II User Stories for Developer

- Show Seasons Where a Team Did Best with the Smallest Tuition

  "As a user who wants to know more about a team's past, I would like to be able to see a season in a team's history where they did amazing with a small tuition. If there isn't something like that for

a team, then maybe just their stats for whenever they had the lowest tuition. With this, I can also compare how tuition affected a single team."

- Add Visual Aids to Website in Video Form
  "It would be beneficial to add another visual aid to your site in video form. This could be to the player pages (e.g. videos of great plays they had) or to the basic information page (e.g. a video explaining roles/positions), whatever your team have the means for. I think this will greatly enrich the user experience since for many, football is best enjoyed and understood through viewing."
- Graphs for Overall Growth of Coaches/Players
  "With the addition of more data for each player and coach this year, I would like an easier way of judging how well a coach/player is doing and if they're either getting worse or better. For the coaches, a W/L ratio on a graph throughout their years would help with visualizing their growth in their profession. Similarly, something to see a player's overall growth would help out too."
- Display and Compare Schools' Win/Loss Record
  "I want to know how different schools' records compare to each other. On the schools' chart, maybe the website could also display the school's wins and losses. I would also like to know how a school's attribute contributes to its win/loss record; e.g. how much does a school's size affect win/loss record, what is the average win/loss record for a specific conference, etc."
- Average Grade for Player Skill
  "Looking at a lot of other athletic databases (MaxPreps and BoxRec specifically), a 5 star ranking system is a wonderful way to simplify information for an inexperienced customer. It would be really neat if there was a "average" or holistic grade that can aid in rankings and visualizing a player's skillset. Maybe this is something that could be displayed on the charts page."

Phase I User Stories from Customer

- List Countries on Artist Pages
  "I was using your site when I found an artist I hadn't heard of before. I wanted to find out what country they were from, but it wasn't listed on the artist's page. I would like a link to the country page(s) that each artist is from."
  - Status: Substituted with countries an artist is popular in.
  - Implemented: Phase 2.
- Lyric Format

"Hello, I am a user who really enjoys reading lyric lines, but it is slightly difficult to do so on your website. I think it would be great if they were spaced line by line. This would really help me be able to follow along with the lyrics easily."

   - Status: We were able to split a song's lyrics and space them line by line. Unfortunately, MusixMatch does not provide the complete lyrics for a song.
   - Implemented: Phase 2.

- Artist Images on Top Songs

  "Hi, I'm a user who likes visuals and tries to recognize artists by their faces. I noticed that under Charting, the Top Artists had images of the artists whereas Top Songs didn't. I think if the Top Songs could also include artist images, then it would look more consistent across the board and appealing."
   - Status: The customer communicated that they would prefer album covers, so we added album cover images to our Songs model.
   - Implemented: Phase 2.
- Profanity Metric Explanation

  "Hello, I am a user who wants to look at lyrics before I let my child listen to them. I think it would be great if the profanity average had an explanation. This would make it much easier than having to read all the lyrics."
   - Status: We have added a new page named Profanity Metrics that can be accessed through the Nav Bar.
   - Implemented: Phase 2.
- Similar Songs Attribute

  "Hello, I am a user who is actively looking for new music. I like that you have a "similar artists" attribute on the Top Artists models. However, I think that I would also benefit from a "similar songs" attribute for the on the Top Songs models!"
   - Status: We are planning on making this a stretch goal. If we do not have enough time to implement it, we may drop it entirely.

Phase II User Stories from Customer

- Artists' Bio Readability

  "I am a customer who prefers text to be organized and spaced so it's easy to read. For top artists, is there any way to organize or add some space within the Bio component (maybe split into

multiple paragraphs) so readability is easier? Currently, each bio is just one super long paragraph, so it seems like a lot to read at first glance."
  - Status: We were able to split an artist's bio into readable chunks.
  - Implemented: Phase 2.
- Artist Page - Top Songs, Some sort of Profanity stat?
  "I am a customer who is curious about how similar or different a single artist's songs are compared to one another. For a particular artist's page, it lists their top songs. Is it possible to also list some sort of profanity measure for each song? That way, on a specific artist's page, you could see which specific songs may have a lot of profanity and which ones do not."
  - Status: Not implemented. Definitely should be done before Phase 4.
- Country Page - Policies that Influence Presence of Profanity
  "I am a customer who's curious about countries and how they are run. On a specific country page, is it possible to include some facts, rules, or policies that may explain why their profanity score is either low or high? For example, would a country having a low profanity score be a result of having certain censorship laws?"
  - Status: We may drop this request if data is too difficult to find.
- Top Songs - Explanation of Views
  "Hi, I'm a customer who's curious about the popularity of songs. On a specific song page, it lists a number of views. Can an explanation be added for where the view count comes from? How is this number measured, is it from like Youtube, Soundcloud, Spotify, etc.?"
  - Status: Not implemented. Definitely should be done before Phase 4.
- Country's Top Genres
  "Hi, I am a customer curious about what kind of genres are popular in a particular country. For a country page, could the most popular genres be listed? Could they be listed in some sort of ranking order too, if possible? Can view counts for genres be determined, or is there some other way to determine how popular a genre is compared to another?"
  - Status: Not implemented. The most popular genres should be possible, but view counts per genre may be too difficult, so they might not be able to be ranked.

Comments: Although some of the features in our customer stories are within the scope of Phase II, we were not able to attempt them as the user stories came in close to the Phase II submission date, leaving insufficient time to implement.

RESTful API

We used Postman to design our API. Documentation can be found [here].

Songs API Endpoints

- GET filtered songs.
  - api.lyriccheck.me/songs?filter[object]&sort&page[number]

  Returns a list of all the songs that match the filter.
- GET a single song by id.
  - api.lyriccheck.me/songs/:id

  Returns the information of the song for the given id.

Artists API Endpoints

- GET filtered artists.
  - api.lyriccheck.me/api/artists?filter[object]&sort&page[number]

  Returns a list of all the artists that match the filter.
- GET a single artist by id.
  - api.lyriccheck.me/artists/:id

  Returns the information of the artist for the given id.

Countries API Endpoints

- GET filtered countries.
  - api.lyriccheck.me/countries?filter[object]&sort&page[number]

  Returns a list of all the countries that match the filter.
- GET a single country by id.
  - api.lyriccheck.me/countries/:id

  Returns the information of the country for the given id (code).

Models

Song

- id varchar(50) PRIMARY KEY NOT NULL
- spotify_id varchar(50)
- name varchar(100)
- artist_name varchar(100)
- artist_id varchar(50)
- artist_spotify_id varchar(50)
- genre varchar(50)
- album_name varchar(100)
- album_id varchar(50)
- countries_popular_in varchar(2)[]
- views int
- profanity_toxicity float
- profanity_profanity float
- profanity_identity_attack float
- profanity_sexually_explicit float
- lyrics text
- description_summary text
- description_content text
- image_url varchar(100)
- spotify_uri varchar(50)

Each song page begins by displaying the song title and the artist that wrote it. Afterward, a view count derived from the last.fm is shown, allowing users to see how popular a certain song is compared to others. We can sort the top songs by this statistic, as well as song name, genre, and profanity score. The profanity score, is derived from last.fm, provides a score for the song's lyrics, including profanity, toxicity, and sexually explicit and identity-attacking lyrics. The lyrics of the song are also displayed on the page for the user to read for themselves.

Artist

- id varchar(50) PRIMARY KEY NOT NULL

- name varchar(50)
- genre varchar(50)
- top_tracks_ids varchar(50)[]
- top_tracks_names varchar(100)[]
- similar_artists varchar(50)[]
- countries_popular_in varchar(2)[]
- views int
- followers int
- popularity int
- profanity_toxicity float
- profanity_profanity float
- profanity_identity_attack float
- profanity_sexually_explicit float
- image_url varchar(250)
- spotify_uri varchar(50)
- bio_summary text
- bio_content text

Each artist's page displays their portrait on the upper-right side of the screen of them or their preferred picture visible on Spotify. Below that, running down the right side of the page is their social media. Below that is their profanity score. And finally below that is their stats on Spotify. To the left of all that information begins their biography. Then, below that is their albums and songs are listed in decreasing order of popularity. The album/song's covers, along with the text below saying its name, will be linked to that album/song's page.

Country

- country_code varchar(5) PRIMARY KEY NOT NULL
- name_common varchar(50)
- name_official varchar(200)
- region varchar(10)
- subregion varchar(50)
- continents varchar(50)[]
- languages varchar(50)[]

- profanity_toxicity float
- profanity_identity_attack float
- profanity_profanity float
- profanity_sexually_explicit float
- top_tracks_ids varchar(50)[]
- top_tracks_names varchar(200)[]
- top_artists_ids varchar(50)[]
- top_artists_names varchar(200)[]
- flags_svg varchar(100)

Each country page has a display of each country's name and a picture of their national flag. In addition to their common name, an official name and country code are provided. In the future, these country codes will be used to be able to look up countries more efficiently without having to write out their whole name. After their geographical location (given by continent and relative location), the profanity score of each country will be displayed. These scores consist of songs that originate from their country and their average amount of toxicity, profanity, sexually explicit and identity-attacking lyrics, and average overall. These scores are derived from the Perspective API, which produces these scores for each country it is given. Below that is a reactive Google Maps window positioned on the country. The languages spoken in the country are listed below that. Finally, a list of top songs and top artists of that country is shown.

Filterable Attributes:

Songs:

- Genre
  - Dropdown menu with a list of all genres
- Listens:
  - Dropdown menu with a list of ranges (e.g. 0 - 1,000)
- Country popular in
  - Dropdown menu with a list of all countries
- Profanity score (multiple)
  - Dropdown menu with a list of ranges (e.g. 0 - .25)

Artists: (identical attributes will have a dropdown menu in the same manner as with songs)

- Genre
- Listens
- Followers

    - Dropdown menu with a list of ranges (e.g. 0 - 1,000).
- Country popular in
- Profanity score (multiple)

Countries:

- Region
    - Dropdown menu with a list of all regions
- Subregion
    - Dropdown menu with a list of all subregions
- Continent
    - Dropdown menu with a list of all continents
- Language
    - Dropdown menu with a list of all languages
- Profanity score (multiple)
    - Dropdown menu with a list of ranges (e.g. 0 - .25)

Sortable Attributes:c

Songs: name, artist, album, listens, genres, average profanity score.

Artists: name, top songs (by name), listens, followers, genres, average profanity score.

Countries: name, country code, region, subregion, top song (by name), top artist (by name), average profanity score.

Searchable Attributes:

Songs: name, artist, album, lyrics, description, country popular in.

Artists: name, top tracks, similar artists, biography, lyrics of songs.

Countries: name, name official, country code, top tracks, top songs, region, subregion, continents, languages.

Media:

Songs: lyrics, cover art, Spotify player.

Artists: biography, artist image, Spotify player.

Countries: flag, google maps.

Connections:

Songs: Connects to artist through their artist. Connects to country through the countries the song is charting in.

Artists: Connects to their songs. Connects to country through the countries the artist is charting in.

Countries: Connects to songs through the top songs in the country. Connects to artist through the top artists in the country.

Tools

Frontend

React

- ReactJS - used for web application development and interfacing with backend.
- React Bootstrap - CSS framework used for splash, model, about pages, and navigation bar.
- Material UI - CSS framework used for charting pages.

Backend

- Namecheap - domain name registrar used to obtain lyriccheck.me.
- AWS Amplify - cloud platform used to host the React app.
- AWS Elastic Beanstalk - used to deploy the API for our website.
- PostgreSQL - database used to store model data
- Flask-Restless-NG - framework used to turn database into API.
- Docker - container used to run backend

DevOps

- GitLab - DevOps software used for source code management and Continuous Integration.
    - Kenji Nakachi / LyricCheck · GitLab
    - Note: The majority of failing CI Pipelines for Phase 2 is a result of commenting out "test" from the stages section of the .gitlab-ci.yml file. This was done to avoid wasting our free minutes for the GitLab pipeline runner.

API Documentation

- Postman - used for testing and building APIs.
    - Documentation

Style

- ESLint - linter was used to statically analyze our code to quickly find problems.
- Prettier - an opinionated code formatter.
- Black - formatter used to format backend Python files

Data

Perspective - Perspective API

- We use Perspective API to provide a “profanity score.” A song’s lyrics are ran through the API and provides a score based on profanity, severe toxicity, sexual explicitness, and identity attacks. These metrics are returned along with an average.
- Using this, we can give an artist, or album, a profanity score by averaging the scores of their songs. Likewise, we will be able to use the user's playlist using the profanity scores of the songs in the playlist. We also give a profanity score for a country by averaging the profanity score of the current top 50 songs in the country.

Spotify - Web API | Spotify for Developers

- Spotify does not provide all the artist and song information we would like, but we are using it to provide media information. We pull image data and embedded iframe players for songs and artists.

Last.fm - API Docs | Last.fm

- Last.fm provides most of the information concerning an artist, album, or track such as an artist’s bio, a song’s lyrics, the top albums/tracks for an artist, information on the current top charts, etc. The information may be a bit out of date, though.

REST Countries - REST Countries

- We use REST countries to get a list of all the countries and non-music-related data about them. Out of the API, we’re able to get names, location information, language, map URLs, and flag URLs images.

MusixMatch - Musixmatch API

- MusixMatch lets us search songs and get the lyrics for the results. We feed the list of songs we get from Last.me into MusixMatch to get their lyrics, then we can feed the lyrics into Perspective to get their profanity score. Unfortunately, MusixMatch does not provide the complete lyrics for a song.

Hosting

Amazon Web Services (AWS)

- We obtained our URL "lyriccheck.me" from the domain name registrar Namecheap. We chose to use AWS Amplify to deploy our React application. Following the setup guide provided (https://github.com/forbesye/cs373/blob/main/React_AWS_Setup.md), we set the host to our Gitlab repository. We then connected our domain name and received an SSL certificate to enable HTTPS. There was a bit of difficulty setting this part up as we had previously connected our domain name to a hosting zone/buckets and registered it with NameCheap for our static HTML website, but it did not take too long to sort out. We then added CNAME and ALIAS records to our registrar and waited for the changes to propagate. Our site is now accessible from lyriccheck.me, www.lyriccheck.me, https://lyriccheck.me, and https://www.lyriccheck.me.
- We use AWS Elastic Beanstalk to deploy our website API. We followed the setup guide provided (https://github.com/forbesye/cs373/blob/main/Flask_AWS_Deploy.md). We set up Elastic Beanstalk CLI, initialized our application, built and ran the production Docker image, and then created an Elastic Beanstalk environment. Next, we configured the TLS/SSL domain to allow our API to be accessible at https://api.lyriccheck.me. We also set up automatic deployment of our API on GitLab pipelines in the .gitlab-ci.yml file. We use AWS RDS to host our PostgreSQL database.

Phase II Features

Testing

We use Python's unittest framework to test calls to our backend.

We use Jest to test the JavaScript of our frontend. We utilize enzyme to test that our components render without crashing and contain the expected text. We also utilized Snapshots to test our components.

We use postman/newman:alpine to test our Postman documentation by exporting our Postman collection using the Collection v2.1 JSON schema and running newman on the resulting file renamed Postman.json.

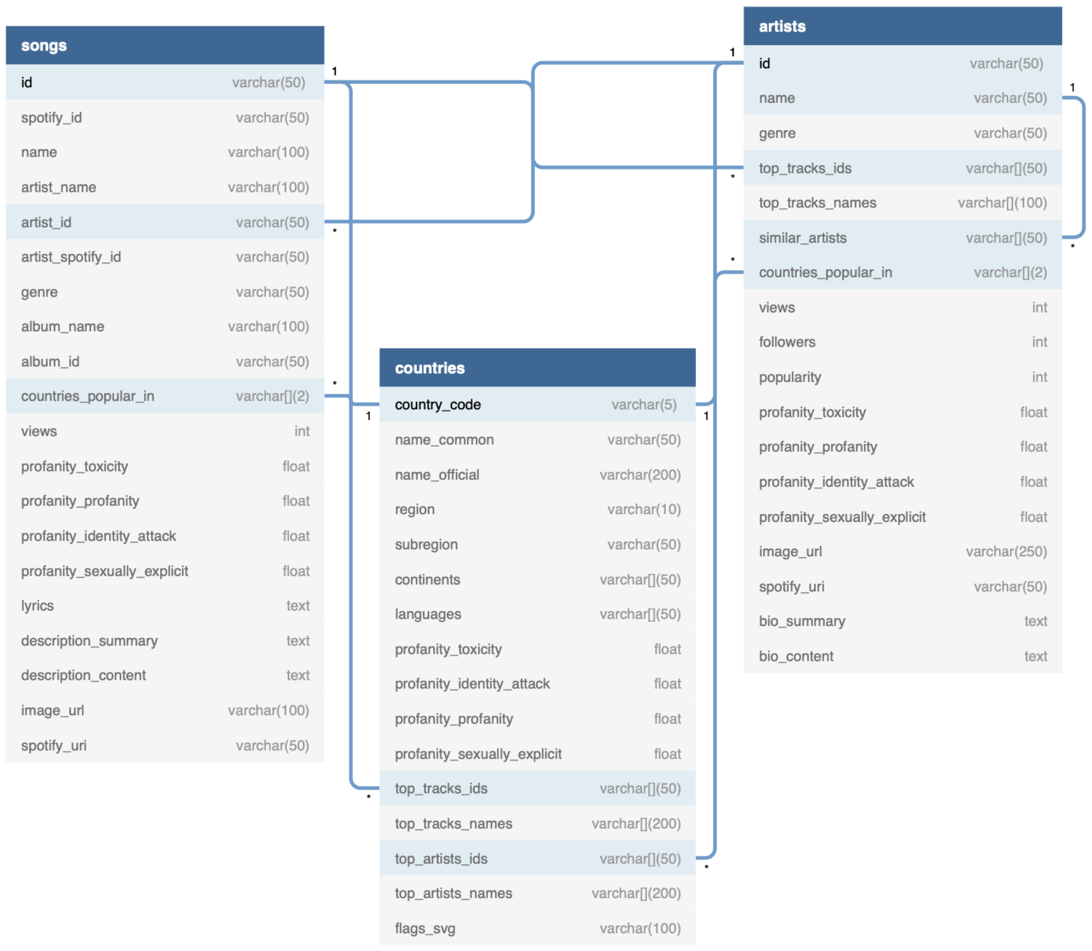
We use Selenium to test our GUI using the navigation links displayed on the pages for our models. For Chrome 100.0.4896.60 on Mac M1 we use the appropriate ChromeDriver version 100.0.4896.20.

Pagination

We added pagination of our API results through the query parameters. For example:

- api.lyriccheck.me/songs?page[number]=1 would return the first 10 songs
- api.lyriccheck.me/songs?page[number]=2 would return the second 10 songs

Database



songs	
id	varchar(50)
spotify_id	varchar(50)
name	varchar(100)
artist_name	varchar(100)
artist_id	varchar(50)
artist_spotify_id	varchar(50)
genre	varchar(50)
album_name	varchar(100)
album_id	varchar(50)
countries_popular_in	varchar[](2)
views	int
profanity_toxicity	float
profanity_profanity	float
profanity_identity_attack	float
profanity_sexually_explicit	float
lyrics	text
description_summary	text
description_content	text
image_url	varchar(100)
spotify_uri	varchar(50)

countries	
country_code	varchar(5)
name_common	varchar(50)
name_official	varchar(200)
region	varchar(10)
subregion	varchar(50)
continents	varchar[](50)
languages	varchar[](50)
profanity_toxicity	float
profanity_identity_attack	float
profanity_profanity	float
profanity_sexually_explicit	float
top_tracks_ids	varchar[](50)
top_tracks_names	varchar[](200)
top_artists_ids	varchar[](50)
top_artists_names	varchar[](200)
flags_svg	varchar(100)

artists	
id	varchar(50)
name	varchar(50)
genre	varchar(50)
top_tracks_ids	varchar[](50)
top_tracks_names	varchar[](100)
similar_artists	varchar[](50)
countries_popular_in	varchar[](2)
views	int
followers	int
popularity	int
profanity_toxicity	float
profanity_profanity	float
profanity_identity_attack	float
profanity_sexually_explicit	float
image_url	varchar(250)
spotify_uri	varchar(50)
bio_summary	text
bio_content	text

We designed our database with three tables, one for each model:

- Songs connect to artists through artist_id and connect to countries through countries_popular_in (links to country_code).
  - “id” is the primary key we obtain from Last.fm.
- Artists connect to songs through top_tracks_ids (links to id), connects to artists through similar_artists (links to name), and connects to countries through countries_popular_in.
  - “id” is the primary key we obtain from Last.fm.
- Countries connect to songs through top_tracks_ids and connect to artists through top_artists_ids (links to id).
  - “country_code” is the primary key we obtain from REST countries.