Web Engineering Project Report

Cornelis Zwart (S3331660) Richard Westerhof (S3479692))

October 24, 2019

1 Introduction

We created an API with a front-end to visualize the Million Song Dataset. For this we used a few technologies and frameworks which we will explain below.

We did not utilize all the data in the dataset, as this was not necessary for user experience.

2 Architectural Overview

The system makes use of the framework Spring Boot, which uses the following components:

- API: for handling GET/POST/PUT/DELETE requests
- Model: for the data itself
- Repository: for storing the data
- Configurations: for configuring (access control, and database connection)
- Service: for providing methods to retrieve/update data

3 Technology stack

3.1 Back-end Framework

For the back-end we used the Spring framework. Using Spring we can create normal Java classes for data storage. If we want to save an instance of this class, filled with data, we give it to a repository class. The repository then takes care of storage. Conversely, if we want to retrieve data the repository class can automatically cast to the correct class-type.

Making an API using Spring is also very simple. We specify the url and which HTTP method the function should handle. Per function we can specify whether it will return JSON or CSV. And what, optional, parameters can be included in the URL. If a function returns a Java object, or list of objects, Spring will automatically convert to the object(s) to a JSON format. For CSV we needed to make an function to change an object to a CSV style string, and then return that.

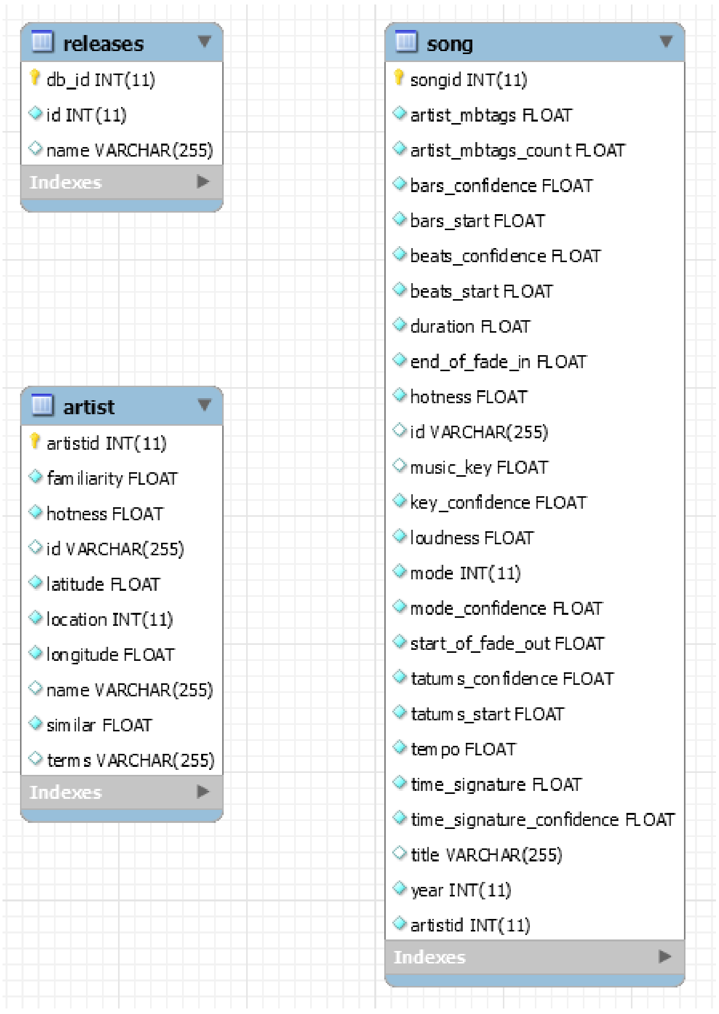
3.2 Front-end Framework

For the front-end we used Vue, as we already had some prior experience with it, and it is easy to learn. Vue works by having templates for each component that consist of some html but with some programming capabilities, such as using a loop to declare multiple similar parts that will then all be generated in pure html, or being able to use variables in the templates. Since vue is reactive, which means that it can react to changes made to the DOM by the code, those variables in the template automatically update their value when it gets modified. This makes it very easy to display data without having to worry about updating it on the page in the code manually. Vue also has the option to use a `router`, which allows it to simulate multiple different pages, each with their own uri, but in reality it is just a single page application, where it swaps out components depending on the route.

To make interacting with our API (i.e. sending http requests) a bit easier, we use axios. Axios provides easy methods to create and send an http request, and returns a promise that resolves to the http response of the API when it gets one, making it really easy to handle as soon as the response is received in the front-end.

3.3 Database

The database we chose was MySQL, because we were most familiar with it. See 1 (following image) for a schema representation;



releases
db_id INT(11)
id INT(11)
name VARCHAR(255)
Indexes

artist
artistid INT(11)
familiarity FLOAT
hotness FLOAT
id VARCHAR(255)
latitude FLOAT
location INT(11)
longitude FLOAT
name VARCHAR(255)
similar FLOAT
terms VARCHAR(255)
Indexes

song
songid INT(11)
artist_mbtags FLOAT
artist_mbtags_count FLOAT
bars_confidence FLOAT
bars_start FLOAT
beats_confidence FLOAT
beats_start FLOAT
duration FLOAT
end_of_fade_in FLOAT
hotness FLOAT
id VARCHAR(255)
music_key FLOAT
key_confidence FLOAT
loudness FLOAT
mode INT(11)
mode_confidence FLOAT
start_of_fade_out FLOAT
tatums_confidence FLOAT
tatums_start FLOAT
tempo FLOAT
time_signature FLOAT
time_signature_confidence FLOAT
title VARCHAR(255)
year INT(11)
artistid INT(11)
Indexes

Figure 1: Database schema

The tables are pretty self explanatory, the `song` table stores songs, the `artist` table stores artists and releases stores releases, however those are not used in our visualisation. We decided to include all information from the dataset, even if we don't use it, so that in the future it can be displayed without re-making the database. For Artist and Song we decided to have an integer key instead of the string used in the dataset. This because our implementation did not allow for primary keys to be strings. Though not visualized in the schema above, in the server it is made sure that foreign keys are matched with existing objects.