BILLBOARD TOP 100

Polco

Author: Piyush Subedi

10/05/2021

OVERVIEW

This is a simple Web based application to view the Top 100 Billboard songs of all time. Users will also be able to filter the songs based on the artist, album and/or genre. In addition, users can mark songs as favorite.

Out of Scope

Some of the features that are out of scope for this project **at this time** have been listed below.

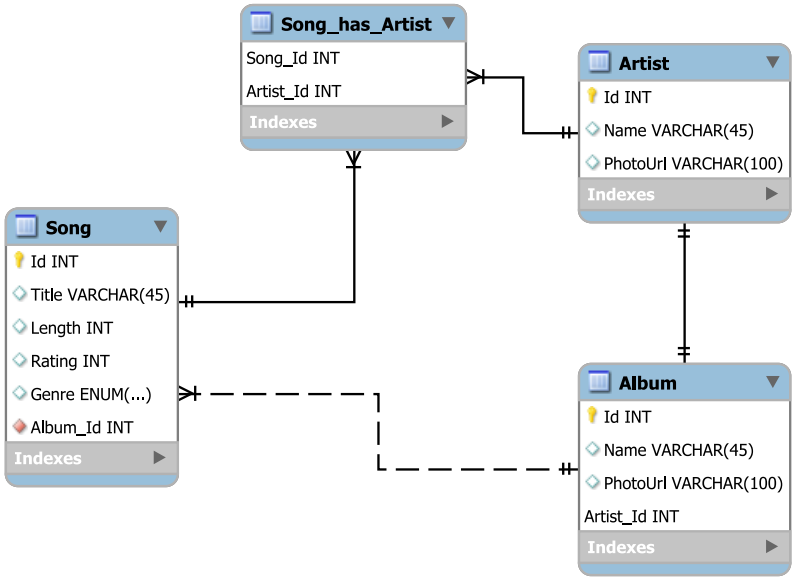
- Users will not be able to play the songs
- Users will not be able to save the filtered list

DATABASE MODEL

Given that the size of the database is relatively small (just a few hundred songs), I opted for a simple relational database (SQL) as I am most familiar with it. Even if the application were to be scaled higher to include more songs, the performance of a relational database would still be good enough for such an application.

> *The other way to store songs would have been to use a document model (NoSQL), since a song would be a document and all the related information such as artists and albums could be gathered just by querying the songs without having to JOIN tables. However, I believe the benefits are insignificant considering the scale of the application.*

Schema



BACKEND ARCHITECTURE

A Node.js server is setup that handles client-side requests to interact with the database.

METHODOLOGY FOR ACCESSING DATA

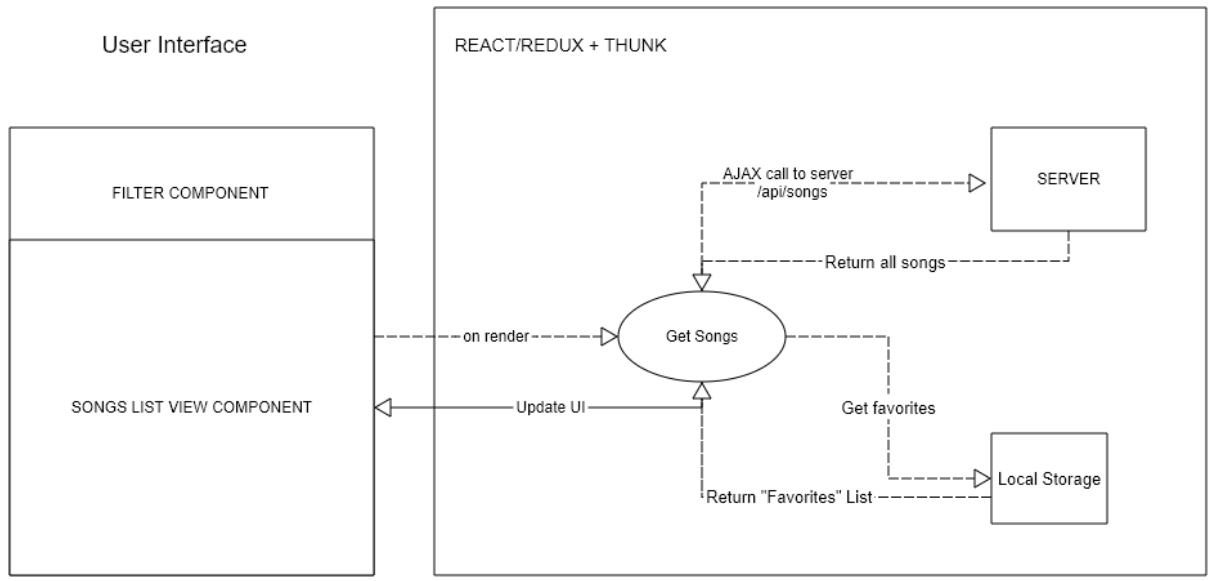
As pointed out in the figure above, the data will be accessed using the (REST) API exposed by the server.

/api/songs → Returns all the Top 100 Billboard **ordered by their ratings,** with similarly rated songs ordered with respect to the **release year** of their corresponding albums.

> *Since we want to allow filtering in the UI, we will also be supplying a separate list of filters for easier access in the frontend.*

FRONT-END ARCHITECTURE

React/Redux with Thunk Middleware has been chosen for constructing the client-side application. The UI will comprise of components like SongsView and FiltersView. Material UI shall be used for styling components.



Favoriting songs

Favoriting is a simple process of marking a song as favorite. Such information is purely user specific. To avoid the complexity associated with introducing Users in the system, **"favorites" are locally stored in client systems**. Every time a user marks or unmarks a song as favorite, the local storage is updated accordingly. On initial render, along with fetching all the songs from the server, the system also retrieves all the favorite items from the local storage.

TESTING STRATEGY

Server-Side Tests / Testing Endpoints (Example)

Tools

Mocha – JS testing framework

Chai – Assertion library

SuperTest – Library for testing HTTP endpoints

```
const express = require('express');
const chai = require('chai');
const request = require('supertest');

const app = express();

describe('GET songs', () => {
        it('should get all songs', () => {
            request(app)
                .get('/api/songs')
                .expect(200)
                .then((res) => expect(....)));

        });
});
```

Client-Side Tests (Example)

Tools

Jest - JS framework, included in React, and acts as a test runner, assertion library, and mocking library

We could test if the songs get rendered on load and that the filters do filter the songs. Further, we can also test the "favoriting" logic and see if favorites get persisted and retrieved accordingly.

```
describe("Form behaviour",  () => {
  it('validate user inputs, and provides error messages', async () => {
    const { getByTestId,  getByText  } = render(<Home/>)
    await act (async () => {
          fireEvent.change(screen.getByLabelText(/artist/i), {
          target: {value: ‘Artist 1”},
      });
….
```