Web Engineering
Architecture Document

Group 13
Maria-Sophia Stefan (s3413896) & Anda-Amelia Palamariuc (s3443817)

October 24, 2019

Introduction

The purpose of this document is to provide a high-level architectural view of our Web application. It aims to show how our software is structured, offering information about our Technology Stack and database implementation, as well as how each component interacts with each other.

Technology Stack

Our software package consists of two main components: the RESTful API as the back-end of the Web app and the UI front-end (which is built against the aforementioned API).

Back end

Our aim is to implement a Web application which uses the same programming language for both the back-end and the front-end layers. Taking this into consideration, we have decided to use **JavaScript**, which has several advantages, such as: simplicity, popularity, versatility, scalability and efficiency.
For the purpose of this project, we have agreed that **ExpressJS** is the obvious framework choice. It is a popular and lightweight prebuilt NodeJS framework, which offers us the possibility to create a server-side web application in a fast and simple way. Apart from being easy to use, ExpressJS is minimalistic, fast, straightforward, flexible and scalable.
As far as our database choice is concerned, we opted for **SQLite3** due to it being lightweight and reliable. Also, it offers better performance (fast reading and writing, only the needed data is loaded and only the necessary parts are being overwritten). Moreover, the fact that it is portable and serverless comes as another significant advantage. Furthermore, SQLite3 is suitable considering that our project will require low to medium database usage.

Front end

To develop the UI of our Web application, we use the standard front-end stack: **HTML, JavaScript** and **CSS3**. In order to compliment our full stack JavaScript application, we decided to use **React** for the front-end. The reason why we chose React amongst other popular frameworks is because it is widely integrated with well-established companies and it is fully backed by Facebook. Thus, it is well-documented and easily maintainable. Moreover, React is simple, scalable and allows us to benefit from resusable components. In addition to this, its language (**JSX**) combines JavaScript and HTML which will allow us to implement the Single-Page Application (SPA) design pattern.

Architecture

Following our technology stack choices, we are planning on building a reactive application based on the Single-Page Application (SPA) design pattern. This will mean that most of the resources (HTML+CSS) are loaded once throughout the lifespan of the application. As opposed to Multi-Page Application (MPA) design pattern, the first page load will be slower, but the overall application will be faster. Moreover, the bandwidth is significantly reduced as only data is exchanged.
Considering the scope of this application, we have decided to use a simple client/server architecture based on SPA, as we have mentioned above. *Figure 1* below gives a clear overview of our application's architecture.

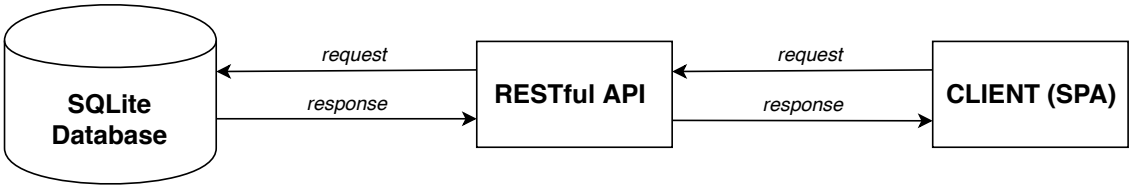

Figure 1: Graphical representation of the architecture

Database design

The database schema, as per *Figure 2*, was designed in such manner that it includes all the information given in the provided CSV file. The attributes for each table can be easily deducted from the given schema.
Furthermore, given the data set we identified two relationships:

1. ONE-TO-MANY *(An artist can have many releases)* which is illustrated by the `Artist_id` foreign key in the **RELEASE** table.
2. ONE-TO-MANY *(A release can have many songs)* which is illustrated by the `Release_id` foreign key in the **SONG** table.

Hence, for the purpose of retrieving all songs by an artist all three tables can be joined together, using a JOIN clause on their aforementioned foreign keys. This design choice is better than having duplicate data (e.g. an additional foreign key `artist_id` in the **SONG** table pointing at the **ARTIST** table).

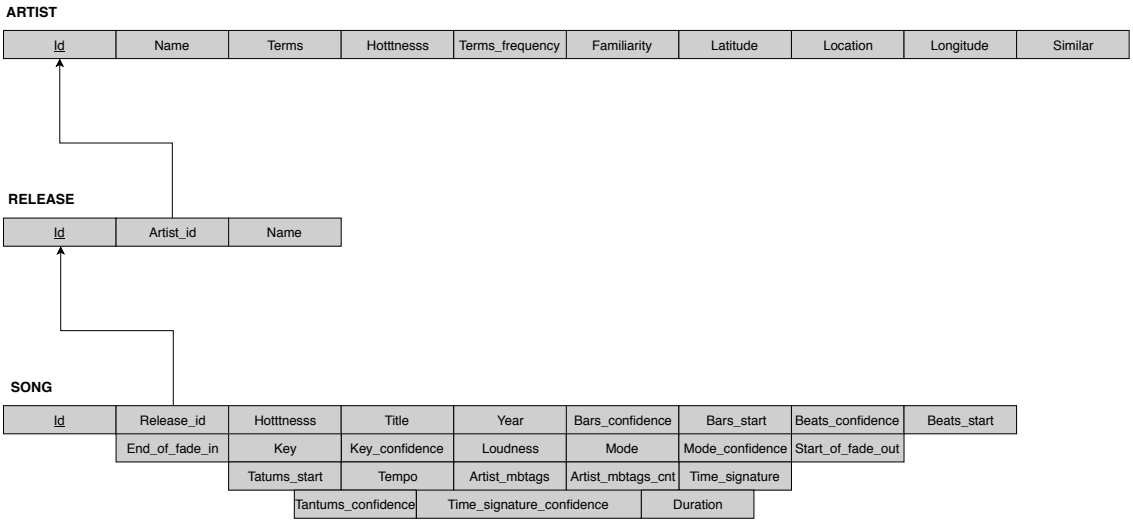

Figure 2: Database relational mapping diagram