Songify Design Document

Issue Record

Version	Date	Notes
0.1	18/3/21	Added ERD, frontend framework research, explanation on design choices and how SOLID is guaranteed .
0.2	13/4/21	Numbered the chapters, C4 Model, APA style references and DOT Framework methods used.
0.3	14/5/21	Added CI/CD setup diagram, updated C4 model, updated backend design choice (H2 → MySQL), updated ERD.
0.4	3/6/21	Updated C4 and ERD.

Table of Contents

1. Introduction

The project Songify aims to create a web application that anyone can use to create, edit, view, delete and share playlists with friends. In this document the architecture of the applications involved will be discussed, and the choices made during the making of them will be justified.

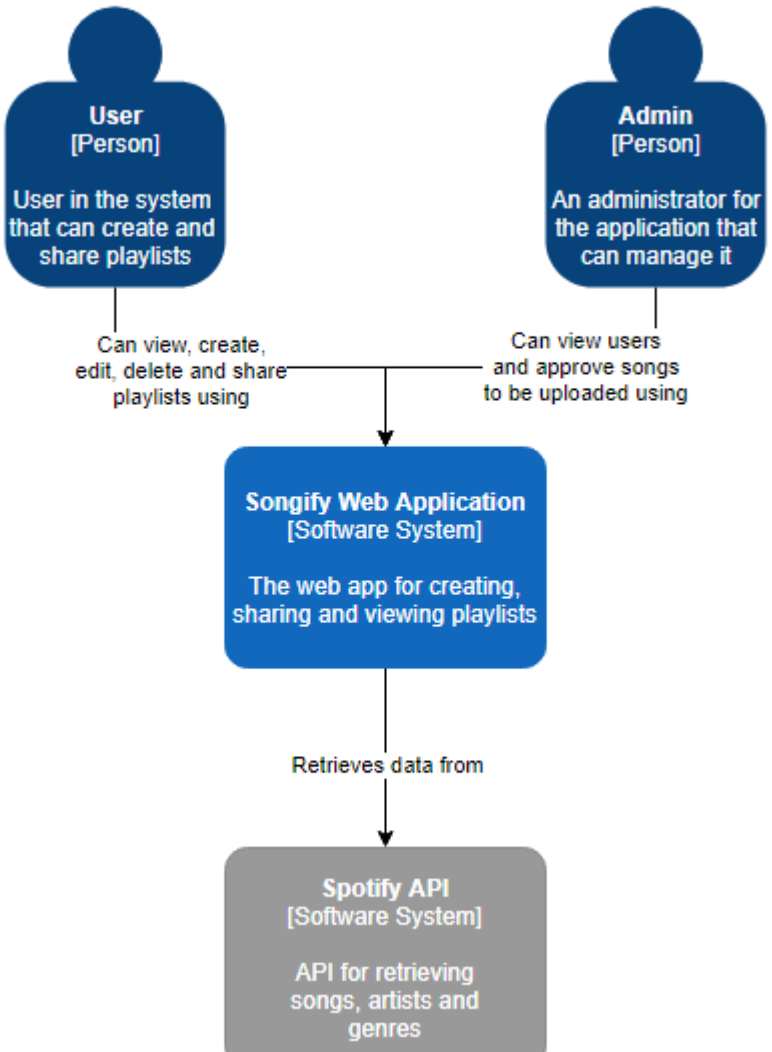
2. Architecture

2.1. C4 Model

The C4 model was made to show the architecture behind the Songify application. The reasoning for why specific systems and technologies were used are discussed later in this document.

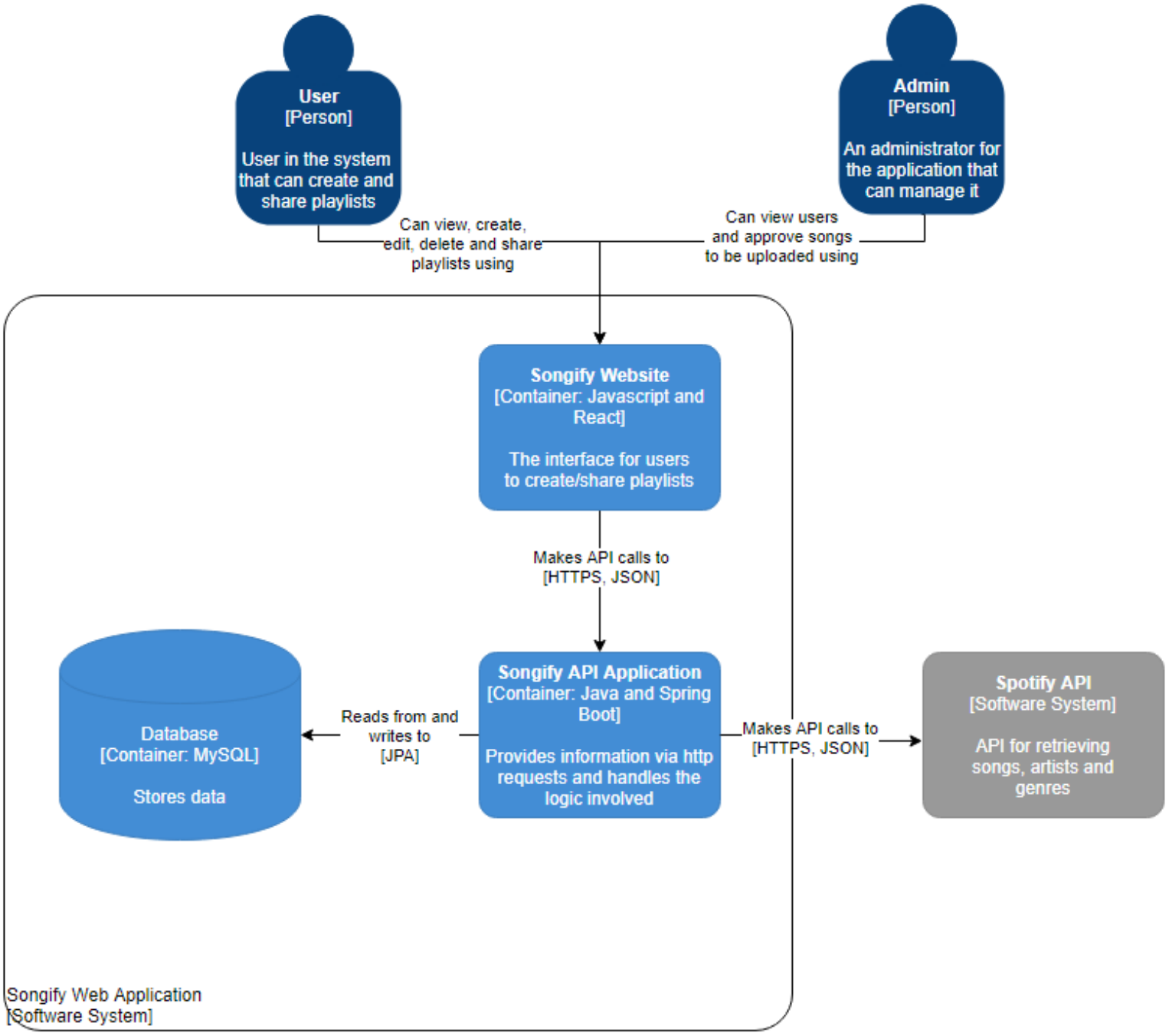
C1: System Context Diagram

The system context diagram shows the software being made, the actors it interacts with and the external system it uses. In the diagram the two actors are regular users and admins. The external system shown is the Spotify API.



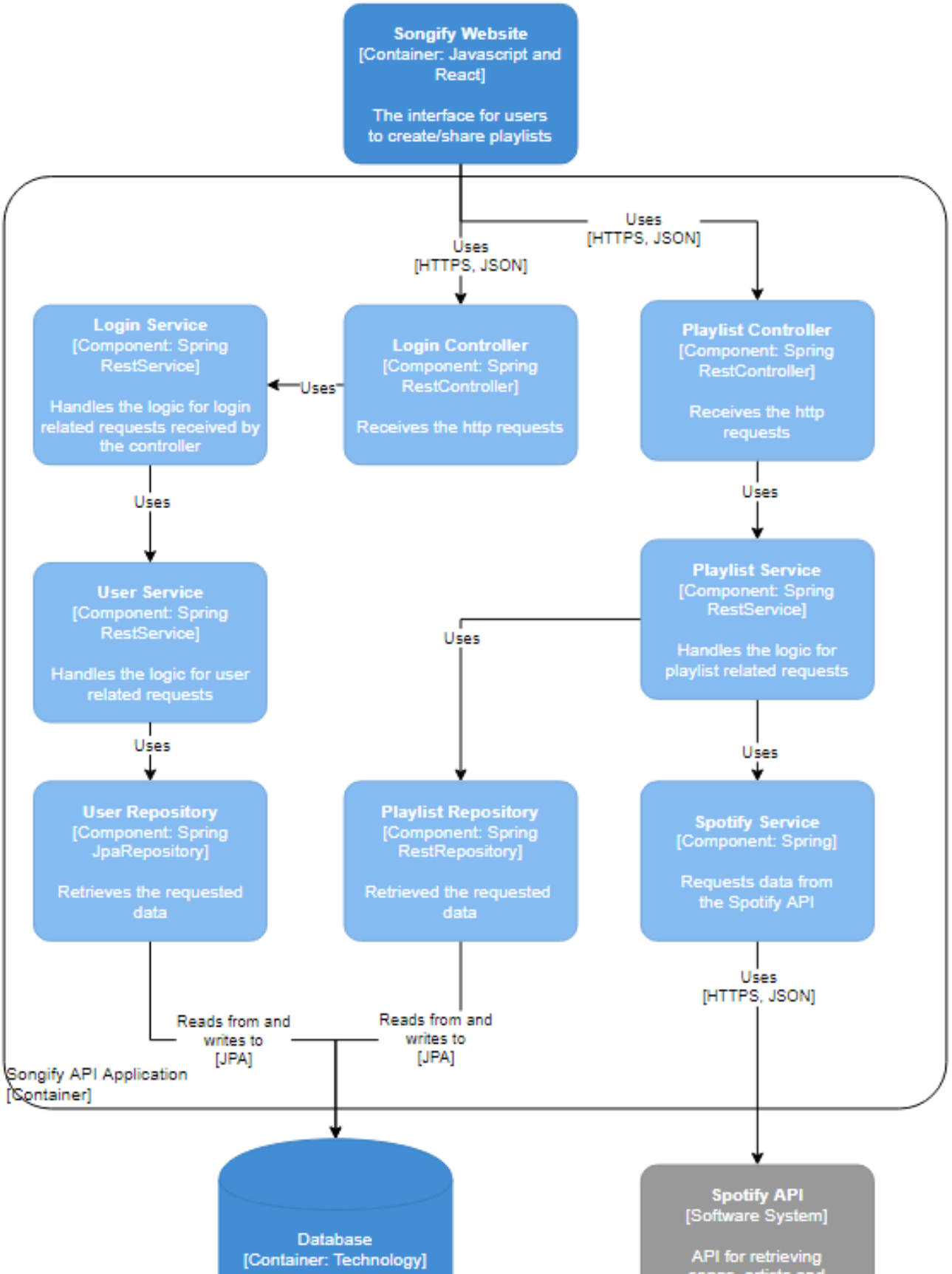
C2: Container diagram

The container diagram provides a deeper look into the Songify Web Application software system shown in C1. The application consists of a single page web application made with React, a Spring Boot restful api and a MySQL database to store data.



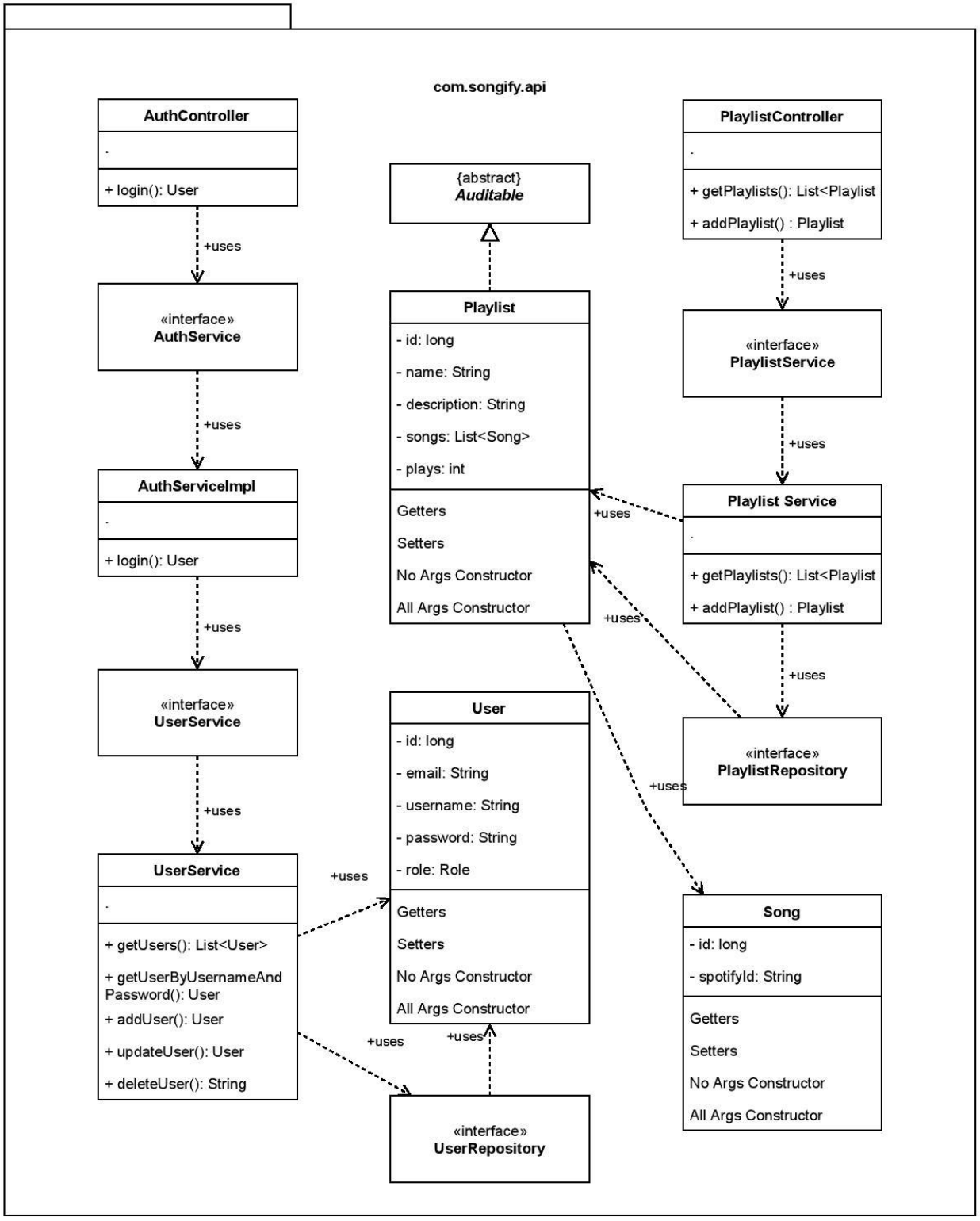
C3: Component diagram

The component diagram goes even deeper and shows the components that make up the Songify API Application container shown in C2. Because there would be too many to fit on the page only a few routes are shown, instead of all.

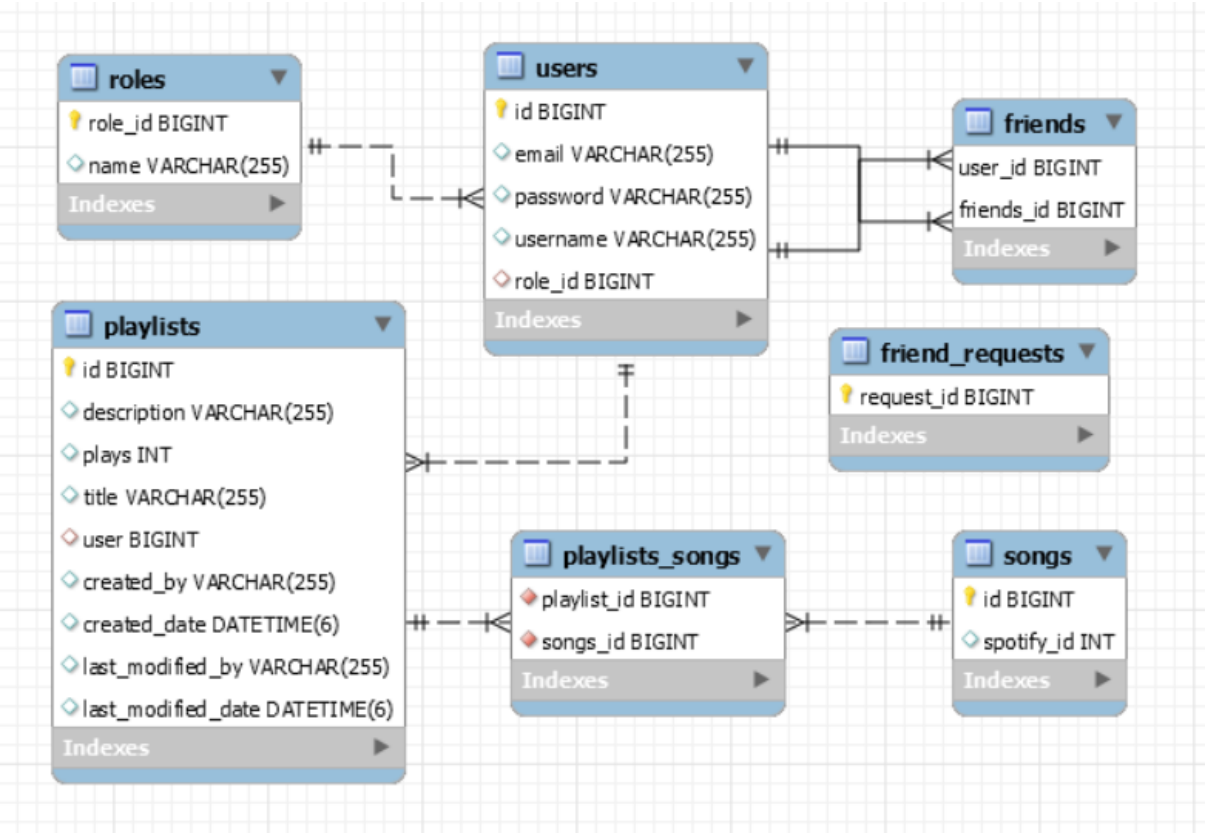


C4: Code diagram

The C4 diagram is essentially a UML Class diagram. The classes shown are not all of the ones used for the api, but the ones related with the C3 diagram. The inheritance from playlist to Auditable is shown and the use of interfaces that extend the JpaRepository class created by the spring framework. The use of interfaces for the service layer is also shown.



2.2 Entity Relationship Diagram



In the ERD above the entities currently in the system are shown. Because the songs are retrieved via the Spotify API, the table only contains the id used in the Spotify database. The roles table handles the difference between regular users and admins. Lastly, the friend_requests table will for now only store friendships but in later iterations could also be used to store other relationships such as followers.

2.3 SOLID Principles

To ensure solid, a 3 layered design has been implemented. The presentation, logic and data access layer. The logic and data access layers are within the Spring boot application while the presentation layer is the react application.

3. Design Choices

3.1 DOT Framework methods used

The research strategies used to find, test and eventually decide on a final option are the following:

Library **Workshop**

3.2 Backend

Why Spring Boot?
Spring Boot significantly decreases development time because it has a default setup for tests and it comes with many dependencies that can be plugged into the Spring application. (Scand, 2020) Another option was to create the API with ASP.NET Core. I have previous experience with this but because of this I decided to take on a new challenge in Spring boot.

Why MySQL?
From my experience, MySQL is easy to use in combination with Spring Boot and it is very popular therefore many examples, guides, and commonly encountered errors can be found easily online.

Why H2 Database?
H2 is the in-memory/embedded database that was originally used in the main application, but is now only reserved for unit testing. It was used for the application first because of the fact that data will not persist, which makes developing very easy, but with production/release coming soon, data will need to persist therefore the switch to MySQL has been made.

3.3 Frontend

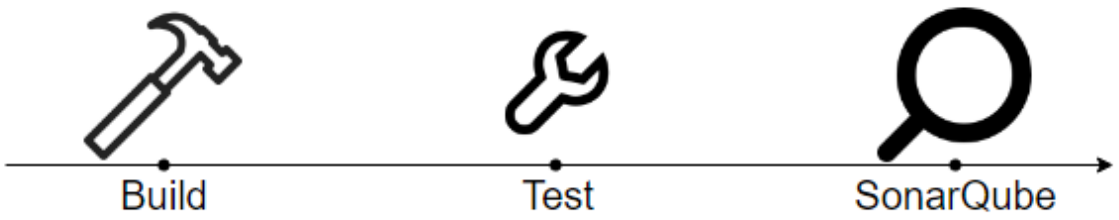
For this project I have chosen React.

	Angular	React	Vue
Ease of Learning	Medium	Medium	Easy
Performance	Good	Good	Good
State Management	✓	✓	✓
Form validation & handling	✓	✗	✗
Routing	✓	✗	✓

Language	TS	JS/JSX	JS/JSX
Personal experience	Minor	None	None

In the table above are the most important factors for me when choosing a framework, with performance and previous experience being the least important. At times, Angular can feel a bit big, because there are many features built into Angular which from personal experience I have never needed. Because of this and the fact that Vue has a small number of components/plugins, I chose React. In the table you may have seen that React lacks form validation and routing, but this isn't an issue because third-party libraries such as Redux and Formik can be used to implement these features. The option to add these third-party libraries is another good reason I chose React. (*Angular Vs React Vs Vue*, 2020)

4. CI/CD



The diagram above shows the stages within the CI pipeline. Once code is pushed, firstly a build is made, then the unit tests are executed and lastly sonarqube is used to test code quality. In a future iteration integration tests will also be added to this pipeline, but since they are not implemented yet, I did not add it as a stage in the diagram.

Bibliography

Angular vs React vs Vue. (2020, March 19). Academind. Retrieved March 18, 2021, from https://academind.com/tutorials/angular-vs-react-vs-vue-my-thoughts/

Scand. (2020, June 26). *Key advantages of using spring boot*. SC & Scand. Retrieved March 18, 2020, from https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/