**Design Document v1.0**

WaveSSoundS

1. **Description**

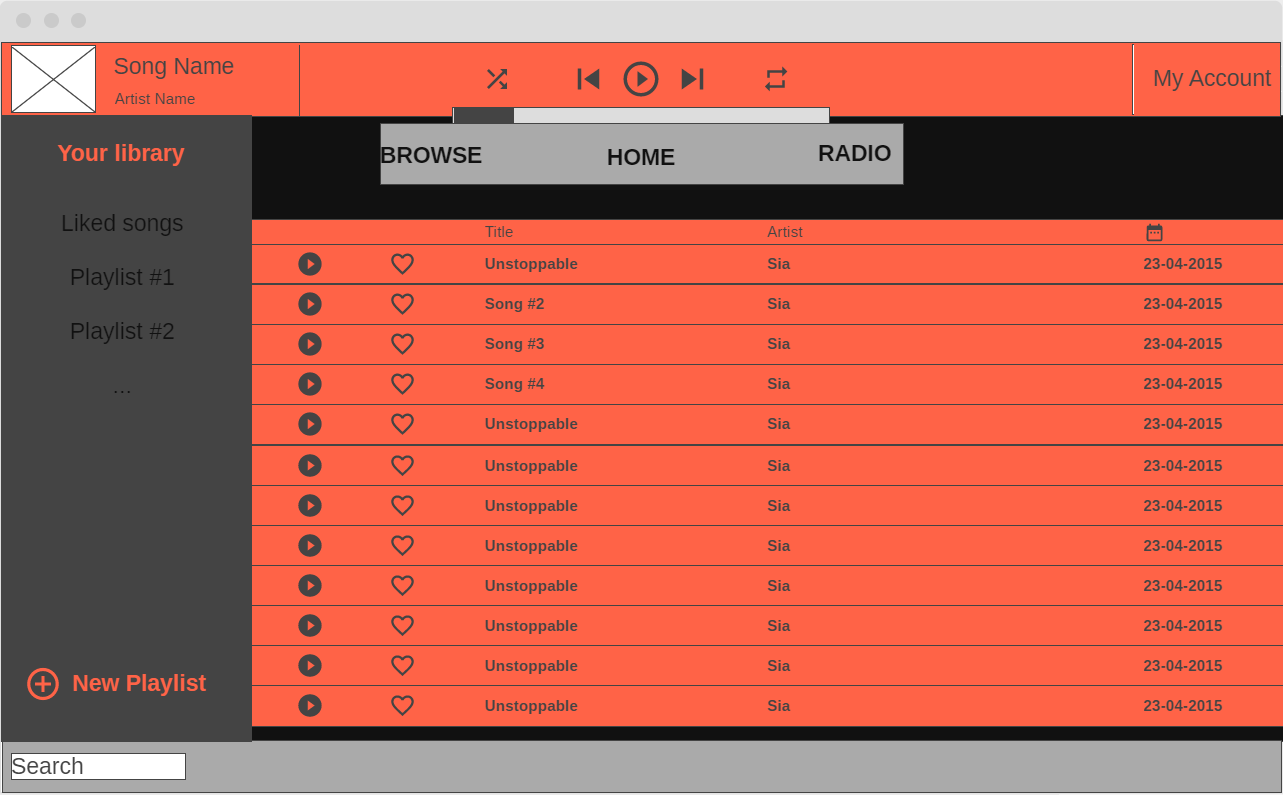
WaveSounds app will be a music streaming web application using Java backend with Spring framework and React frontend. The application will be kept simplistic and pleasant for the user. Without complex animations and screens, we are aiming efficiency for the clients not prettiness.

1. **Justification for choosing JS Framework**

For this project I finally concluded that React would be the best framework to use for this type of application. I will use the Spotify API in my Java backend. I chose React because I never used it until now and I think it will be a plus if I will learn a new framework. Even though Angular has the best documentation, React documentation is not that bad either, and besides, Angular I already used in the past.

1. **Wireframes**

Below you can see a prototype of how the app will look like. Most of the elements will be available at all time and the content container will change depending on the user’s choices. The search bar will be available on the bottom of the screen, which when clicked and used the content container will be also replaced with the search results.



The main 3 menus (Browse, Home, Radio) will be available on the top at all time and managing the user content will be made in the left container, creating playlists and seeing them.

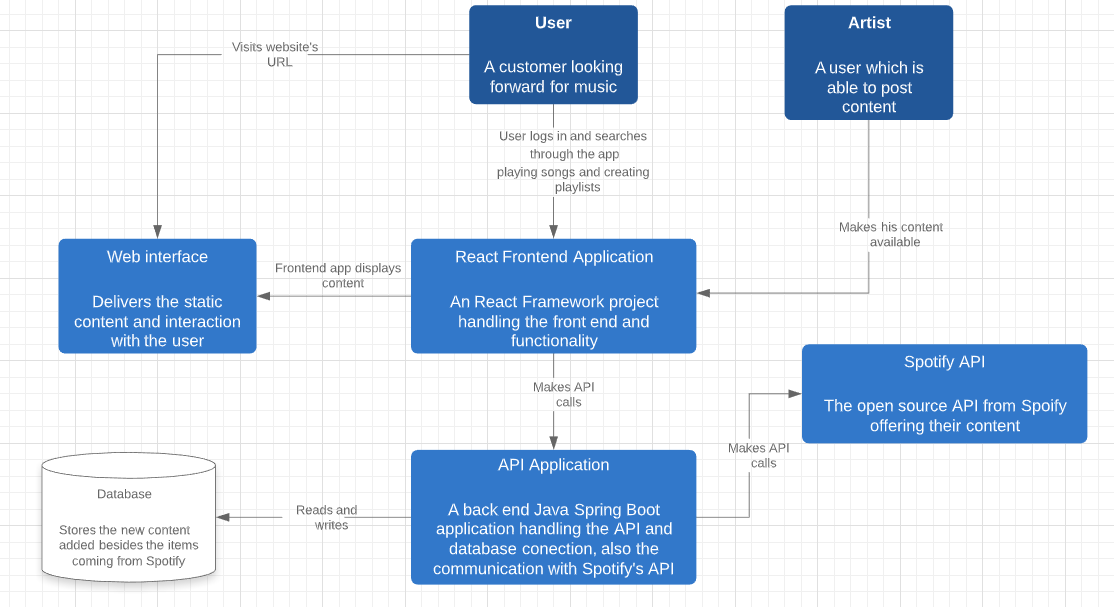
1. **System Overview**

The main purpose of this application is to make music available to the user. So, the main goal to achieve is to create a music streaming platform with search functionality and sorted content using the shared items from Spotify. Besides that, the user should be able to create playlists linked to their account and also to like a song which will add the song to your Liked Songs library. The music player most probably will not be functional, but a preview of each song should be available. From the artist point of view, the app should also be able to post new songs from with an artist account. These new songs will not have any audio in them.

In a possible failure scenario, the data from Spotify API will not work as expected and the search functionality will not be available properly. In this case the database will be modified and hopefully it will work close to the way that is supposed to.

1. **Context diagram**

**Here you can see the basic architecture and actors of the application and the communication around it.**



1. **Milestones**

The planning is made starting with Sprint 3.

1. **Sprint 3 milestone**
2. **Document view:**
   1. Review on the 1st version of the Design Document
   2. 2nd version of the Design Document
3. **Technical view:**
   1. Creating the connection between backend and Spotify API
   2. Setting up 1st version of the database
   3. Getting used with React
   4. Creating a practice app with React
   5. Creating a first version of the Database
   6. Connect the database to the backend
   7. Create a good ERD Diagram
   8. Implement another functionality with Spotify API
   9. Install Sonarqube with Docker
4. **Testing view:**
   1. Constant unit testing and analyze
   2. Automated testing with CI
5. **Sprint 4 milestone**
6. **Document view:**
   1. Review on the 2nd version of the Design Document
   2. 3rd version of the Design Document
   3. Report on how the OWASP criteria relate to your application
7. **Technical view:**
   1. Authentication integration
   2. Authorization integration
   3. Frontend update
   4. 1st release version
8. **Testing view:**
   1. Constant unit testing and analyze
   2. Automated testing with CI
9. **Sprint 5 milestone**
10. **Document view:**
    1. Final version of the Design Document
11. **Technical view:**
    1. 2nd release version
    2. Finding problems and finishing the little things
    3. Trying to deploy the app using the CI/CD pipeline
12. **Testing view:**
    1. Constant unit testing and analyze
    2. Automated testing with CI