**Software Requirements and Design Document**

**For**

**Group 3**

**(WeShed)**

Version 1.0

**Authors**:

Eliot S.

Noal G.

Rogelio L.

Steven K.

# **1.** **Overview**

Our project is a web app meant to be used to practice jazz music. The core functionality deals with displaying backing tracks and lead sheets meant to be used to play along with to practice improvisation. The remaining features we will implement include time spent on each track, as well as maintaining a timestamp of the last time each tune was played, so the program is able to recommend songs that the user has not played in a long time. Social and gamification elements like a friends list, achievements and challenges will encourage users to spend more time on the site, and more time practicing.

# **2.** **Functional Requirements**

High priority requirements:  
 Interface for displaying songs/tracks  
 Login/Authentication system  
 Time tracking on each song  
 Search bar for users

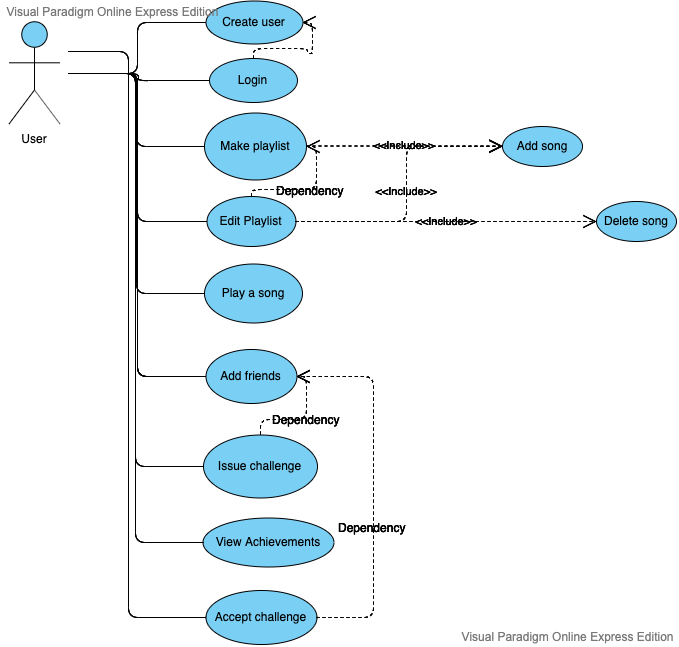
Medium Priority Requirements:  
 Recommendations based on timestamps  
 Friends list  
 Issue/Accept Challenges for time played  
 Notification System

Low Priority Requirements:  
 Achievement Tracking  
 Daily Streak Tracking

# **3.** **Non-functional Requirements**

As a result of our personal user login-based system, our project will require some form of data security in order to ensure the safety of all users. Furthermore, general quality assurance is needed for the design, performance, and reliability of the website, both in terms of code and visual design, in order to provide a high quality experience for the users.

# **4.** **Use Case Diagram**

**

# **5.** **Class Diagram and/or Sequence Diagrams**

*This section presents a high-level overview of the anticipated system architecture using a* ***class******diagram*** *and/or* ***sequence diagrams****.*

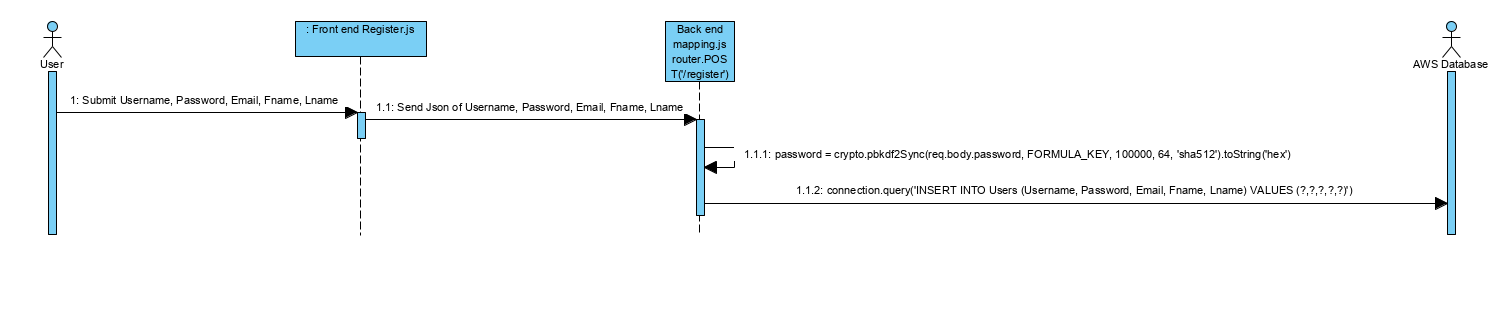
*If the main* ***paradigm*** *used in your project is* ***Object Oriented*** *(i.e., you have classes or something that acts similar to classes in your system), then draw the* ***Class Diagram******of the entire system and Sequence Diagrams for the three (3) most important use cases in your system.***

*If the main* ***paradigm*** *in your system is* ***not Object Oriented*** *(i.e., you* ***do not*** *have classes**or anything similar to classes in your system) then only draw* ***Sequence Diagrams****,* ***but for all the use cases of your system.*** *In this case, we will use a modified version of Sequence Diagrams, where instead of objects, the lifelines will represent the functions in the system involved in the action sequence.*

***Class Diagrams*** *show the* ***fundamental objects/classes*** *that must be modeled with the system to satisfy its requirements and* ***the relationships*** *between them. Each class rectangle on the diagram* ***must also include the attributes and the methods of the class*** *(they can be refined between increments). All the* ***relationships between classes and their multiplicity*** *must be shown on the class diagram.*

*A* ***Sequence Diagram*** *simply depicts* ***interaction******between objects*** *(or* ***functions -*** *in our case - for non-OOP systems) in a sequential order, i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.*

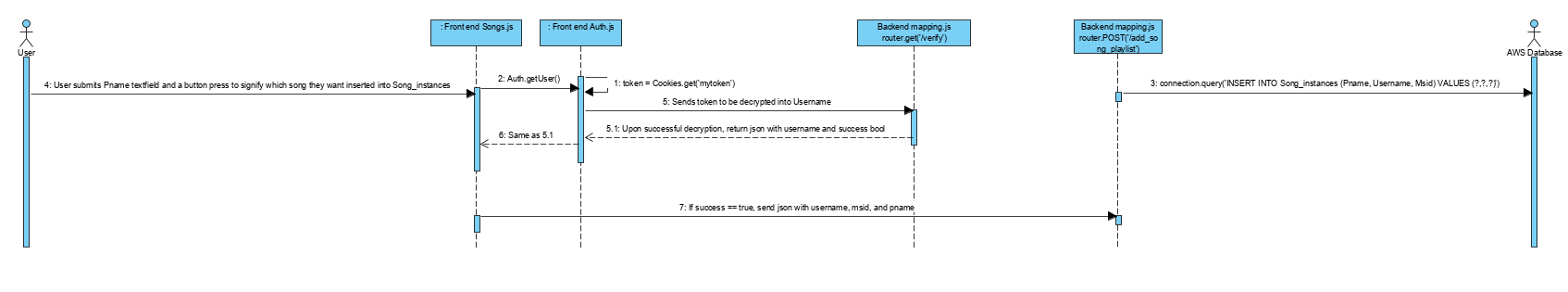
*Create User Sequence Diagram:*

**

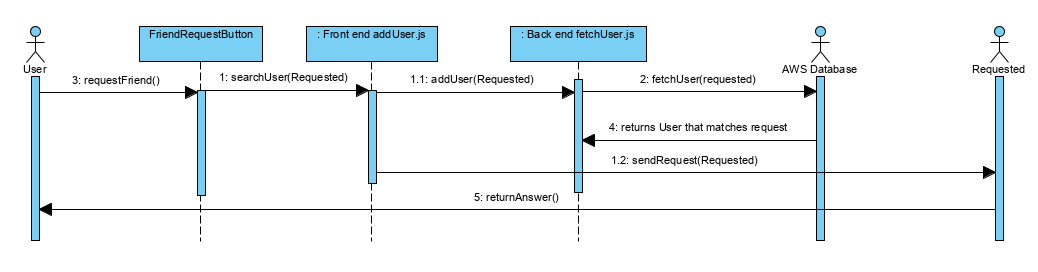
*Login Sequence Diagram:*

**

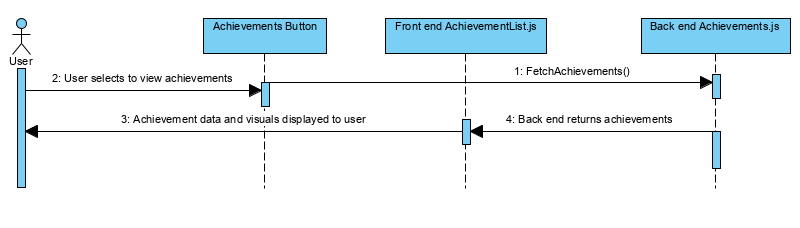
*Add Song Sequence Diagram:*

**

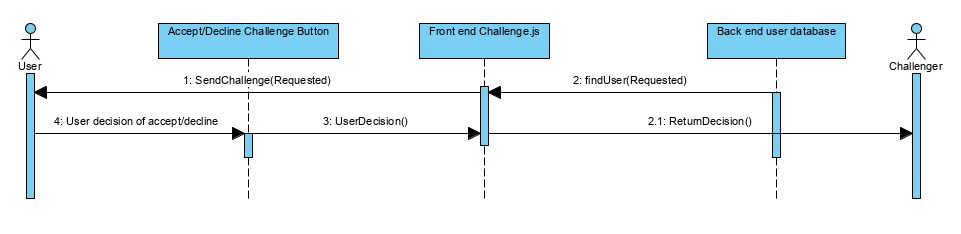
*Friend Request Sequence Diagram:*

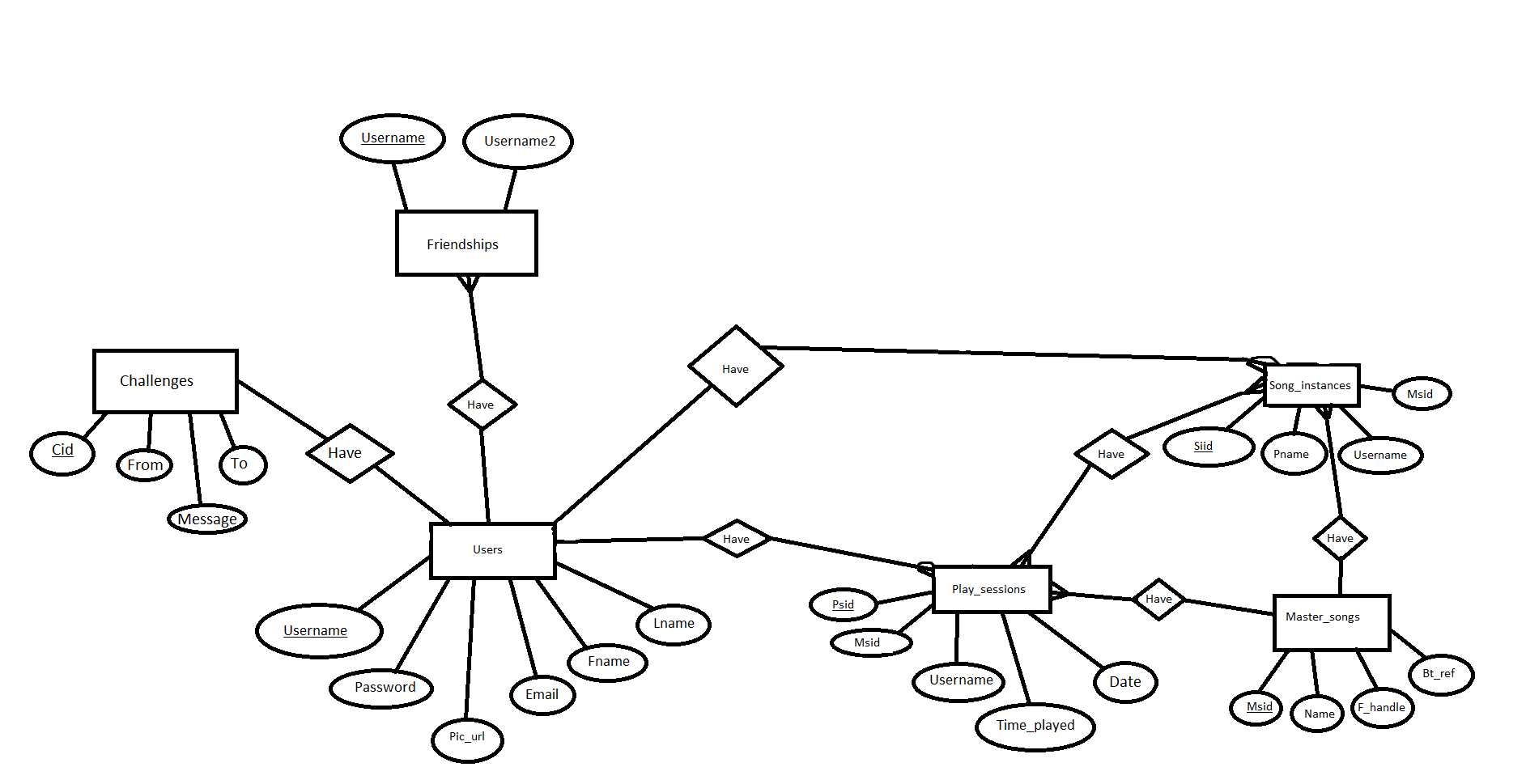
**

*View Achievements Sequence Diagram:*

**

*Accept Challenge Sequence Diagram:*

**

*Tentative database schema: *

# **6.** **Operating Environment**

Our project will be a web application, with the front-end being run in the browser using React.js, and the backend running in a node.js runtime using Express.js and MySQL. For the purposes of this project, it will be assumed that both the front and back ends will be run locally, as opposed to being deployed on a web server.

# **7.** **Assumptions and Dependencies**

It is assumed that the front end, back end, and database will all run simultaneously. There are several dependencies in the project which could halt development entirely. This project depends on using npm start to run the front end and node app.js to run the back end. The back end makes queries or responds to front end requests and relies on an AWS database which is always running. If any one of these falter, the project will not operate as intended. If Cors(Cross origin resource sharing), mysql, jstoken, cookie-js, express, or react become deprecated or fail then the entire project will lose functionality. The mytoken hash in a cookie on localhost serves as a key to the entire website, so if cookies within browsers change during our progression, this could ruin our persistent login system and protected routes.