**Software Requirements and Design Document**

**For**

**Group 3**

**(WeShed)**

Version 2.0

**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:  
 Recommendations based on timestamps: Recommend users songs not played recently  
 Search bar for users: Allow users to search for other users profiles through  
 Profile page display: Robust page for display of both the users stats and other users queried in the search bar  
 Daily Streak Tracking: A system that tracks consecutive days that any song has been played

Medium Priority Requirements:  
 Unify CSS for all pages: Currently styling is not consistent, must be for final iteration  
 Make page easy to access and aesthetically pleasing: Easy access means users will want to use the page

Low Priority Requirements:  
 None

Achieved Requirements:  
 Interface for displaying songs/tracks  
 Login/Authentication system: Login system with encrypted authentication for security  
 Time tracking on each song: Log the amount of time spent on a song  
 A method to create, edit, and delete playlists

Views a potentially infinite number of friends when queried and adding friends with the input of a username string.

Views a potentially infinite number of challenges when queried and challenging users with the inputs of To, Plays, and Message.

Viewing a potentially infinite number of streaks when queried.

Streak-tracking through checking consecutive days of play sessions. When given N dates, it outputs corresponding groups for adjacent N dates.

Achievements:

Levelbadge implemented with input of total number of play sessions while the output is a render of the total number play sessions/5 + 1.

Donutchart implemented with input of unique dates from the total play sessions. The output is the number of unique dates of the total play sessions and rendering a green-arc record, gold record, or platinum record.

# **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, the website and services must be reliable in their use throughout all cases, and the website must provide scalability in order to store large amounts of data. Code should be written in a way that allows for easy maintainability and updating for any future changes.

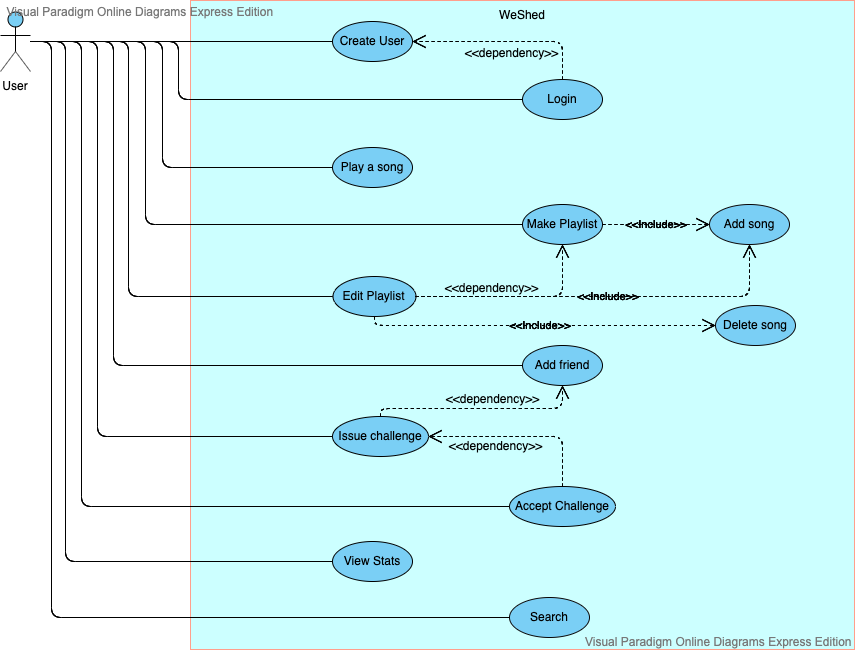
Achieved Non-functional requirements:

Security that hides similar passwords with unique hashes.

Passwords that have an encryption standard of 100000 pbkdf2Sync in sha512.

Unique encrypted hashes to hide usernames in a cookie that have an expiration date of two weeks.

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



Textual Descriptions:

Unique Name: Create User

Participating Actors: User

Entry Conditions: The user submits Username, Password, Email, First Name, and Last Name into a text field to be sent on a submit trigger with POST. The variables to be sent are updated every change of keystroke.

Exit Conditions: User is created.

Flow of events: The user will fill the form and the form data is sent to the backend and the user is inserted into the AWS database.

Unique Name: Login

Participating Actors: User, Database

Entry conditions: User submits via form

Exit conditions: User matches in database and a hash cookie is stored in browser or the user doesn’t match, and nothing happens.

Flow of Events: The user submits via form and then that is attempted to be matched with the database user. If the match is successful, send a hash cookie back to the user to store.

Unique Name: Add song to playlist

Participating Actors: User, Database

Entry conditions: User submits via form and the current song is not null

Exit conditions: None

Flow of Events: The user inputs a playlist name; the name and song info are then inserted into the song instances. It will always insert, even if there are duplicates.

Unique name: Send Friend Request

Participating Actors: Sending User, Requested User, Database

Entry conditions: Entry is started upon the user sending a friend request through the friend request button

Exit conditions: Exit is completed upon an answer to the friend request being returned to the Sending User

Flow of events: The friend request sequence involves sending a request to add a user, which then fetches that user from the database, returns that fetched user to the friend request function, sends a request to that user, and then returns accept/deny to the user that originally sent the friend request

Unique name: View Achievements

Participating Actors: User, Achievements Database

Entry conditions: Entry is started upon the user sending a request to view achievements through the Achievements page.

Exit conditions: Exit is completed upon the page displaying all the achievements and the user’s unique data of their achievement progress.

Flow of events: The view achievements sequence involves a user sending a request to view achievements, which then sends the users data to the database and returns it so that the achievements are properly displayed and individualized based on which of these achievements the user has completed/made progress in.

Unique name: Accept Challenge

Participating Actors: Challenging User, Challenged User

Entry conditions: Entry is started upon the user accepting/denying the challenge that has been issued to them, through the accept/deny challenge buttons.

Exit conditions: Exit is completed upon an answer to the challenge being returned to the Challenging user.

Flow of events: After the challenging user has successfully sent the challenge request through the Send Challenge sequence, the user database finds the challenged user and displays the challenge request from the challenging user. The challenged user then makes his decision to either accept or deny the challenge, and this is sent to the user database, which finds the original challenging user and returns the challenged users answer to their request.

Unique Name: Search

Participating Actors: User, Database

Entry conditions: User looking for song

Exit conditions: None

Flow of Events: User clicks on search button; inputs the name of the song; as the name of song is being typed, the list narrows down to results.

Special Requirements: None

Unique Name: Edit playlist

Participating Actors: User, Database

Entry conditions: User wants to add song or delete song or delete playlist all together

Exit conditions: None

Flow of Events: The user clicks on playlist and has the option to choose and add a song. If the song is already in the playlist, they can choose to delete from the chosen playlist. There’s also an option to delete a playlist by clicking a button.

Special Requirements: None

Unique Name: Delete song

Participating Actors: User, Database

Entry conditions: User chooses song and deletes it from the current playlist

Exit conditions: None

Flow of Events: The user goes to the song list and picks a song – If the song is in playlist then they have the option to remove the song from the playlist that has been typed.

Special Requirements: None

Unique Name: Delete Playlist

Participating Actors: User

Entry Conditions: The user selects a song from the playlist and the playlist array index is loaded into the current data to later be offloaded to the backend via a POST submission that is triggered by a delete playlist button.

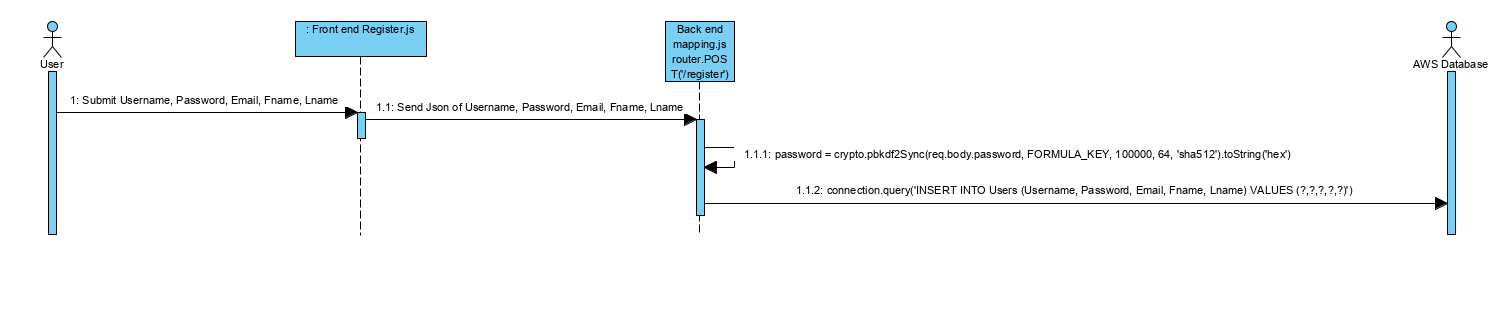
Exit Conditions: The playlist is deleted.

Flow of events: The user clicks on a song and the current variables are loaded with data. Upon clicking delete playlist the current variables are sent in a POST submission to trigger a MySQL deletion of all Song\_instances with Pname equal to the current Playlist name. A reset of the playlist arrays is needed, so the updated data gets fetched from the MySQL Database and the arrays are loaded with fresh data.

Special Requirements: None

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

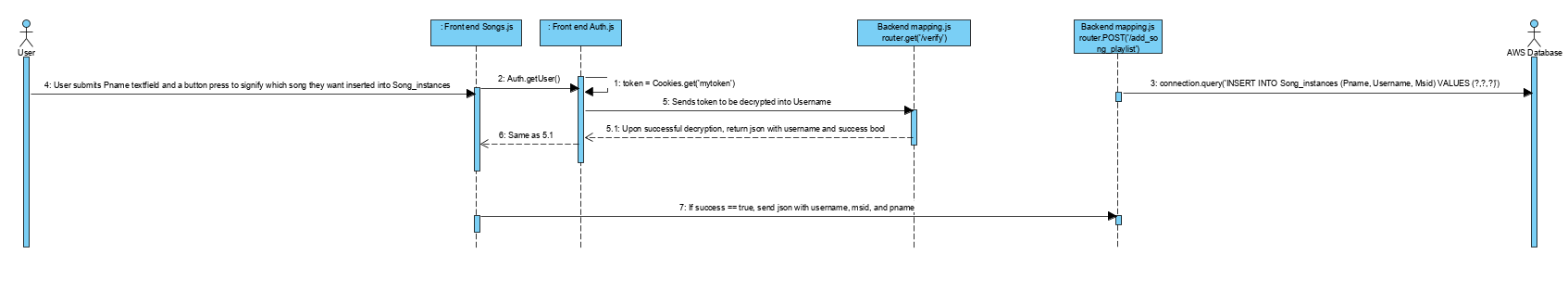
*Create User Sequence Diagram:*

**

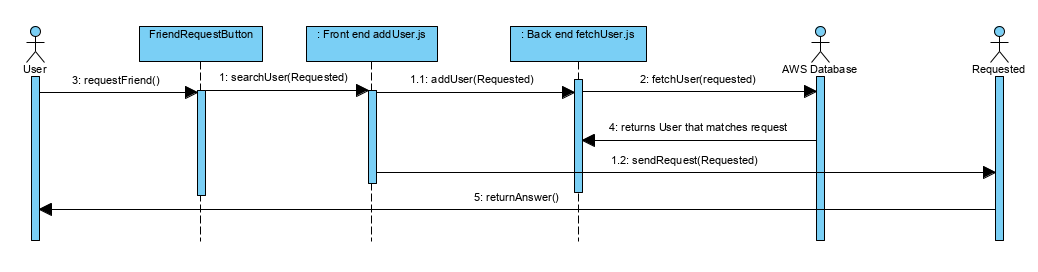
*Login Sequence Diagram:*

**

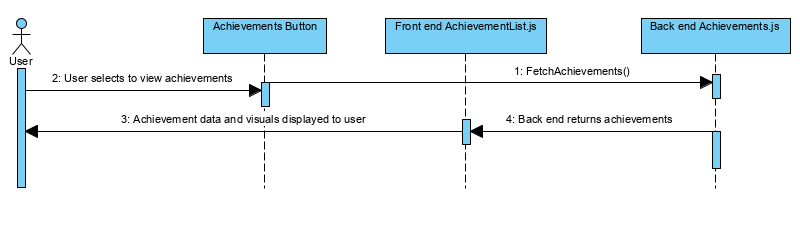
*Add Song Sequence Diagram:*

**

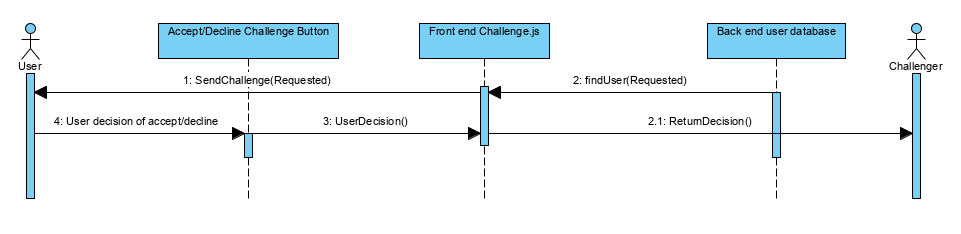
*Friend Request Sequence Diagram:*

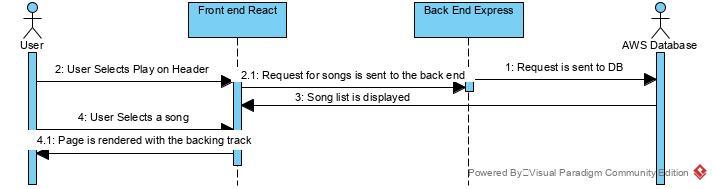
**

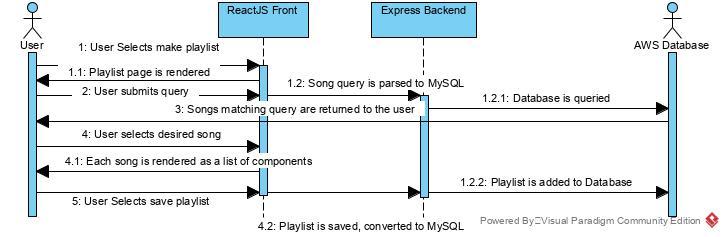
*View Achievements Sequence Diagram:*

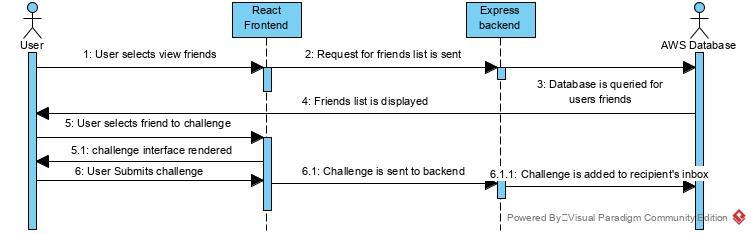
**

*Accept Challenge Sequence Diagram:*

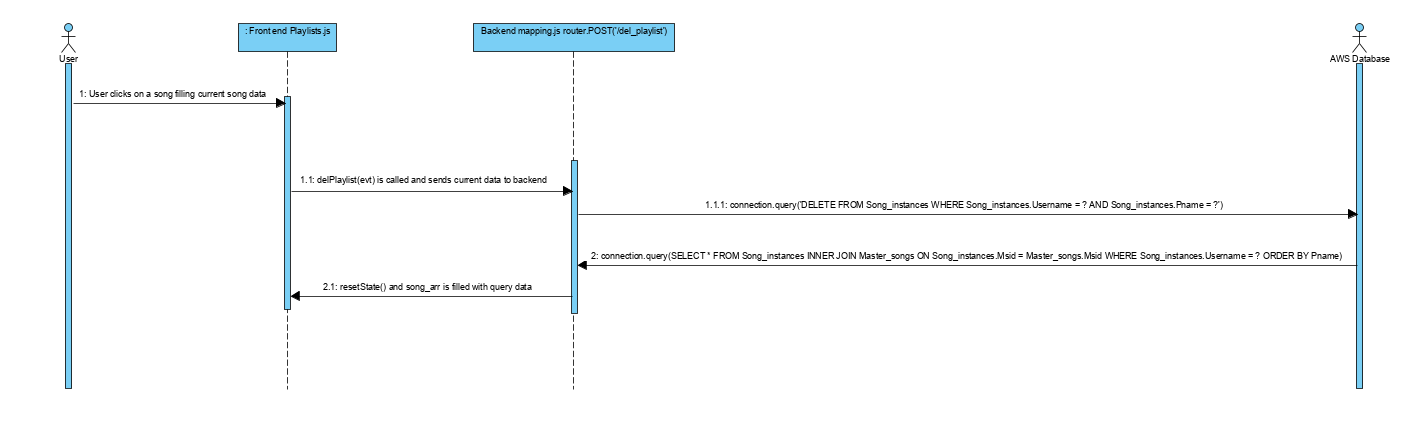
**

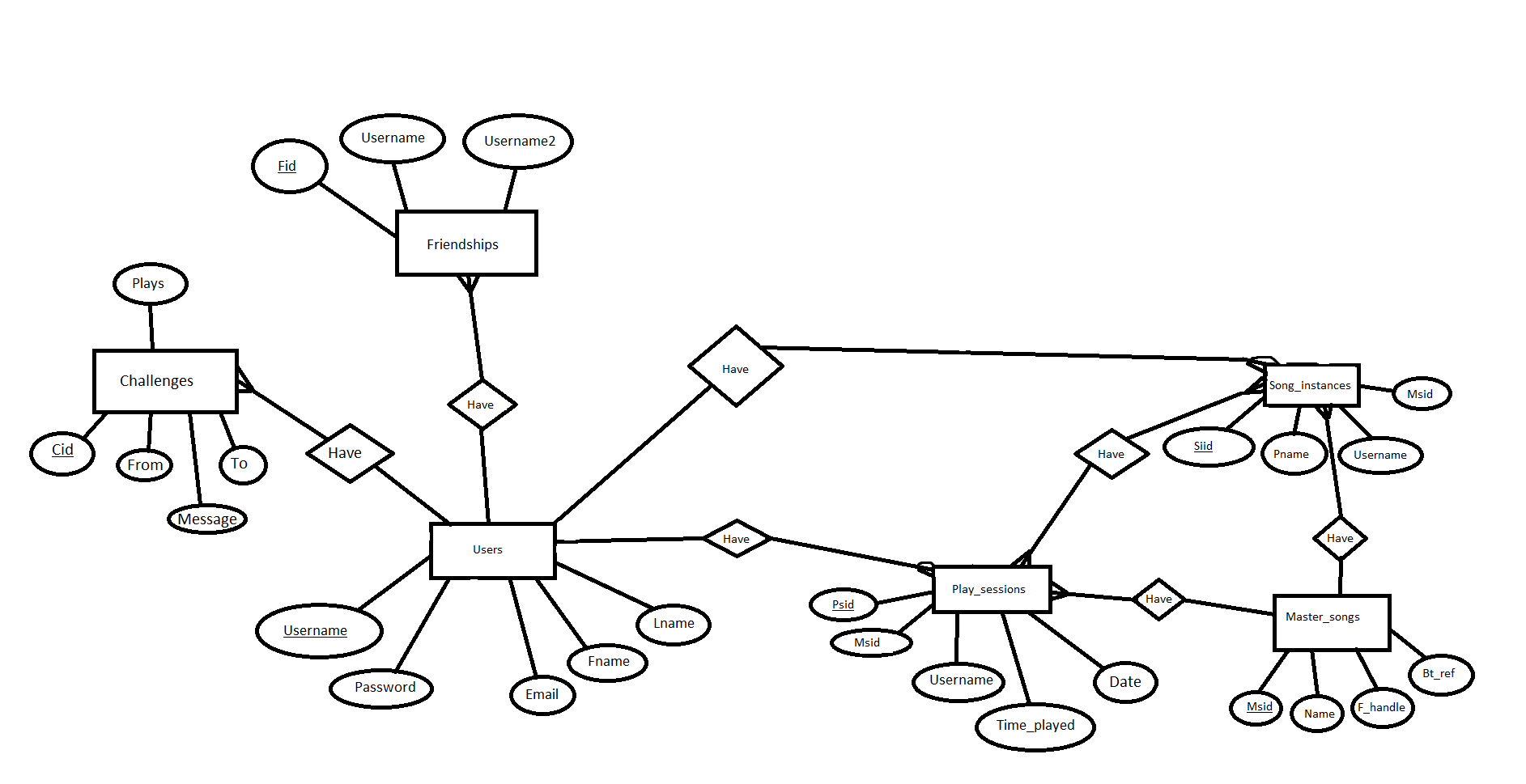
*Play Song Sequence Diagram:*

*Make Playlist Sequence Diagram:  
*

*Issue Challenge Sequence Diagram:  
*

*Delete Playlist Sequence Diagram:*

**

*Increment 3 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 falters, 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.