Vern

Design Document

Team 21:

Carson Rupp

Ethan Clay

Kaung Zaw

Matthew Witherow

Rohan Shankar

Index

Purpose

Students across campus do not have a centralized platform to share and discover music. Music apps such as Spotify currently lack social media aspects that allow users to form groups and discuss the music they are listening to. Local music artists also lack a platform on Spotify and their work is heavily underrepresented. Most music apps focus solely on the global scale, and our project will differ from those and fulfill unmet needs by targeting college students, local artists and businesses that may want an expanded social platform for music making and enjoyment within the local campus community.

Vern will meet these requirements by making use of existing technology such as the Spotify API to provide basic functionality and data to the users. In addition, we will implement group functionality to allow users to create social groups to foster a thriving and active discussion on the local music scene. Local artists and businesses will also be able to use our app to promote live events and performances to the campus community.

Functional Requirements

1. **Users can make an account to join our platform and see generated playlists and recommendations based on the user base of the entire platform.**

   As a user,

   a. I would like to be able to sign up for a Vern account.
   b. I would like to be able to edit my profile to set or change my profile picture and change my bio.
   c. I would like to be able to add genres I am interested in to my profile.
   d. I would like to be able to see what people with similar music tastes to mine listen to.
   e. I would like to be able to see a generated playlist on the most popular songs each week.
   f. I would like to be able to see a generated playlist on local artists' new releases.

2. **Users can make a social group or join one and see interesting statistics about the group and engage in discussions about music with the group.**

   As a user,

   a. I would like to be able to create a social group.
   b. I would like to be able to join an existing social group.
   c. I would like a quick way to view what songs are popular in my social group.
   d. I would like to be able to make a post about albums or songs and post them in my social group.

3. **Local artists and businesses can advertise live events and performances to the user base of the platform.**

   As a local artist/business,

   a. I would like to be able to verify that I am a local artist or business if I am indeed one.
   b. I would like to be able announce and advertise future performances to users.
   c. I would like to be able to see statistics for my music.
   d. I would like to find other local artists to discover and collaborate with.
   e. I would like to be able to share my work with other users.

4. **Users can look up information and interact with live events and performances advertised by local artists and businesses.**

   As a user,

   a. I would like to see a list of local artists using the app.
   b. I would like to see the locations of local artists' performances on something like Google Maps.
   c. I would like to be able to add local artists' performances to my calendar.
   d. I would like to be able to receive notifications about upcoming performances.
   e. I would like to find local artists that write my favorite genres of music.
   f. I would like a quick easy way to share local artists on social media.

Non-Functional Requirements

1. **Client Requirements**

   As a developer,

   a. I want our application to be able to run on the Android platform.

   b. I want our application to be able to run on the iOS platform.

2. **Server Requirements**

   As a developer,

   a. I would like our server to have a comprehensive Restful API structure for quick and easy data requests

   b. I would like our server to be quick in its return to the client that requested from the database

   c. I would like our server to be able to handle requests sufficient to serve a user base of around 50,000.

   d. I want our server to be easily scalable in case we decide to take on more users.

3. **Database Requirements**

   As a developer,

   a. I want our database to be neatly structured with recognizable variable names for efficient access.
   b. I want our database to be secure so as to protect our user's privacy and data.
   c. I want our database to be ready to deploy on the cloud.

4. **Performance Requirements**

   As a developer,

   a. I want our application to be able to handle a user base size of a university student population.
   b. I want the server code to make efficient requests to the Spotify API when needed without making excessive 429 error code results.
   c. I want the application to gracefully handle exceptions and errors in the case that there are indeed errors.
   d. I want the mobile app to be responsive and snappy to all user inputs and events.
   e. I want our application to scale if necessary without making drastic changes to the client, server or database code.

5. **Security Requirements**

   As a developer,

   a. I want every user, local artist and business' account info to be properly handled using appropriate hashing algorithms to create a safe and secure platform.

6. **Appearance/Usability Requirements**

   As a developer,

   a. I want our application to have a clean, stylish and modern user interface.
   b. I want our application to have an intuitive UI that makes daily usage straightforward.

Design Outline

High-level Overview of the System

Our project is a social media app on iOS/Android for music sharing and discussion on a local, campus-wide scale. Our implementation will use a client-server-database model. The client through the Spotify API provides information to the server where important information relating to statistics, playlists and groups will be parsed and stored in the database. The server will then access this information on the database to display to the client/user as he needs. The database also interacts with an internal statistics module to generate recommended playlists and interesting stats.

Detailed Overview of Client/Server/Database Architecture

1. Client
    a. Implemented in React Native for both iOS and Android.
    b. The mobile client provides a user interface for the user.
    c. The client sends requests to the server.
    d. The client receives request responses from the server and displays them accordingly in its user interface.

2. Server

a. Implemented in NodeJS.

b. The server receives and handles requests from the client.

c. The server validates requests and sends queries to the database to add, modify and get data based on the user's request.

d. The server packages the appropriate response and sends it back to the target client.

3. Database

a. A relational database implemented in MongoDB.

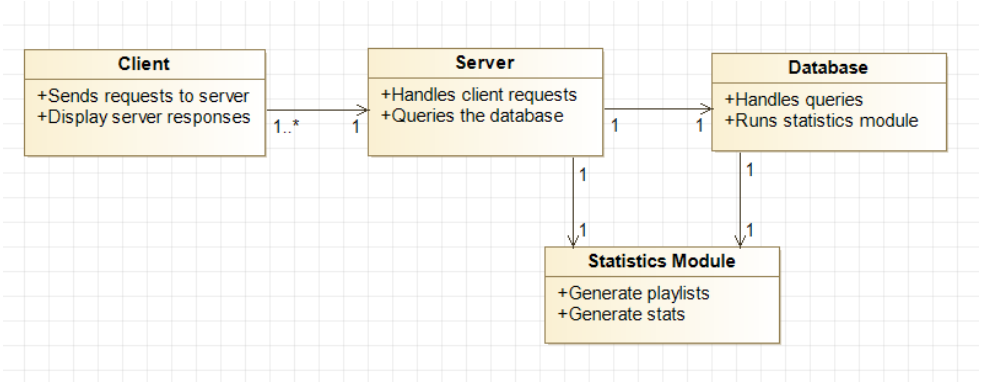
b. Stores all data used in the application, such as user info, performances and live events, etc.

c. Runs the statistics module weekly to generate recommended playlists and interesting stats which are then parsed and stored in the database.

4. Statistics Module

a. Invoked by the database to generate playlists and stats.

High Level UML



Design Issues

Functional Issues

1. **Do users need to login to our service?**

   Option 1: Third-party login through Google, Facebook, etc.

   Option 2: Allow users to create login unique to our application

   Option 3: No

   Decision: We chose to allow users to create a login unique to our application because our app targets users in the campus community. As such, we decided to allow users to register with only a @purdue.edu login.

2. **Should we ask users to link their Vern account with their Spotify account?**

   Option 1: Yes

   Option 2: No

   Decision: We decided that having users link their Vern account with their Spotify account would be necessary since it would allow our app to do so much more: recommended playlists, weekly interesting statistics on listening habits as well as local artists being able to advertise their work on Spotify through our app.

3. **How will we deal with different types of Events that could limit audiences?**

   Option 1: First come first serve

   Option 2: Set up an event that allows them to input and share event information.

   Decision: We decided that allowing businesses to input event information would be the best way to go about this issue. Since the businesses/clubs may have different venue sizes, who may actually show up, and the time, place and artist would be easier to give the businesses power over this decision rather than us having to input this information ourselves.

4. **How can groups be managed?**

   Option 1: Everyone can see everyone and all the groups are open

   Option 2: Create different groups for some privacy

   Decision: We decided to go with the adverse groups users, clubs, or businesses will have to have both privacy and a group of people who share a common interest in music. Users will be able to have open and private groups and manage it through an invite system for easy joining and sharing.

5. **Should we keep the beta testing to only Purdue or other users as well?**

   Option 1: Only Purdue

   Option 2: Anyone

   Decision: We decided to go with letting anyone be beta testers but be slightly favored towards software developers. If we decided to keep the beta testing limited to only people at Purdue, we wouldn't be getting a very good sample size of users. Especially in respect to the age of our users as the majority of students at Purdue are between ages 17-23. Regardless, we will still have many Purdue students on our beta and should be sufficient for testing local charting among campuses.

Non-Functional Issues

1. **Which programming language / framework are we going to use to implement our app?**

   Option 1: Swift + Xcode

   Option 2: Java/Kotlin + Android Studio

   Option 3: Javascript + React Native

   Decision: We chose Javascript and React Native because we wanted our app to be cross-platform to have the largest user base. We also think React Native is a much easier, straightforward and easier-to-learn framework than developing natively.

2. **Which framework will we use to implement our back-end server?**

   Option 1: Node.js

   Option 2: None - client handles all processing

   Decision: We chose Node.js because there will be a lot of requests to the Spotify API that our application will need to handle. Using a dedicated back-end also means that we will have an easier time when we scale.

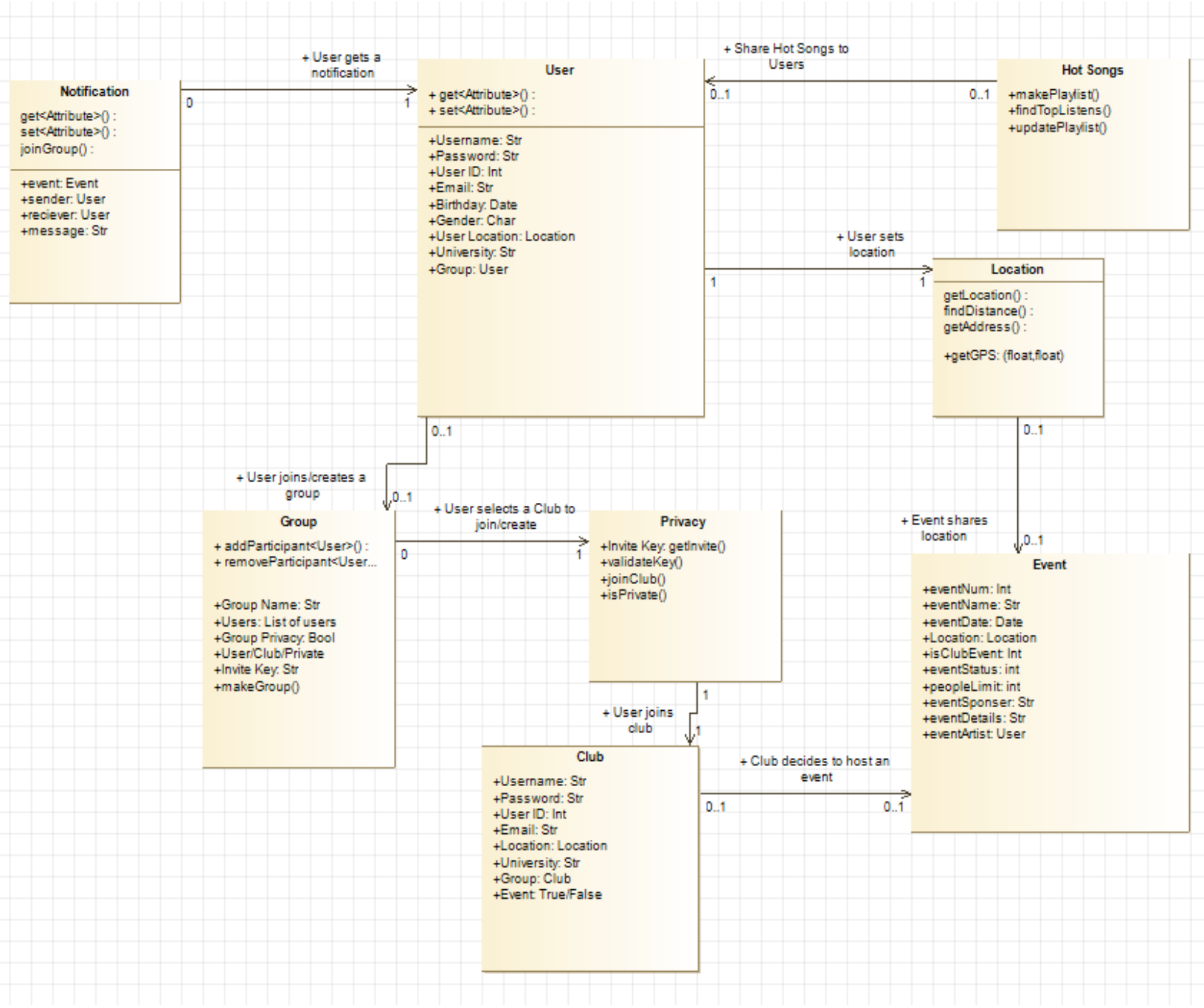
3. **Which database service are we going to use for our project?**

   Option 1: Firebase

   Option 2: MongoDB

   Decision: Both are perfectly fine, so our decision here is more of a personal preference. We chose to go with MongoDB because there seemed to be a higher amount of support and documentation for a full-stack MERN app using MongoDB than Firebase. For example, password storing with Node.js Bcrypt and its integration with MongoDB is pretty straightforward.

Design Details



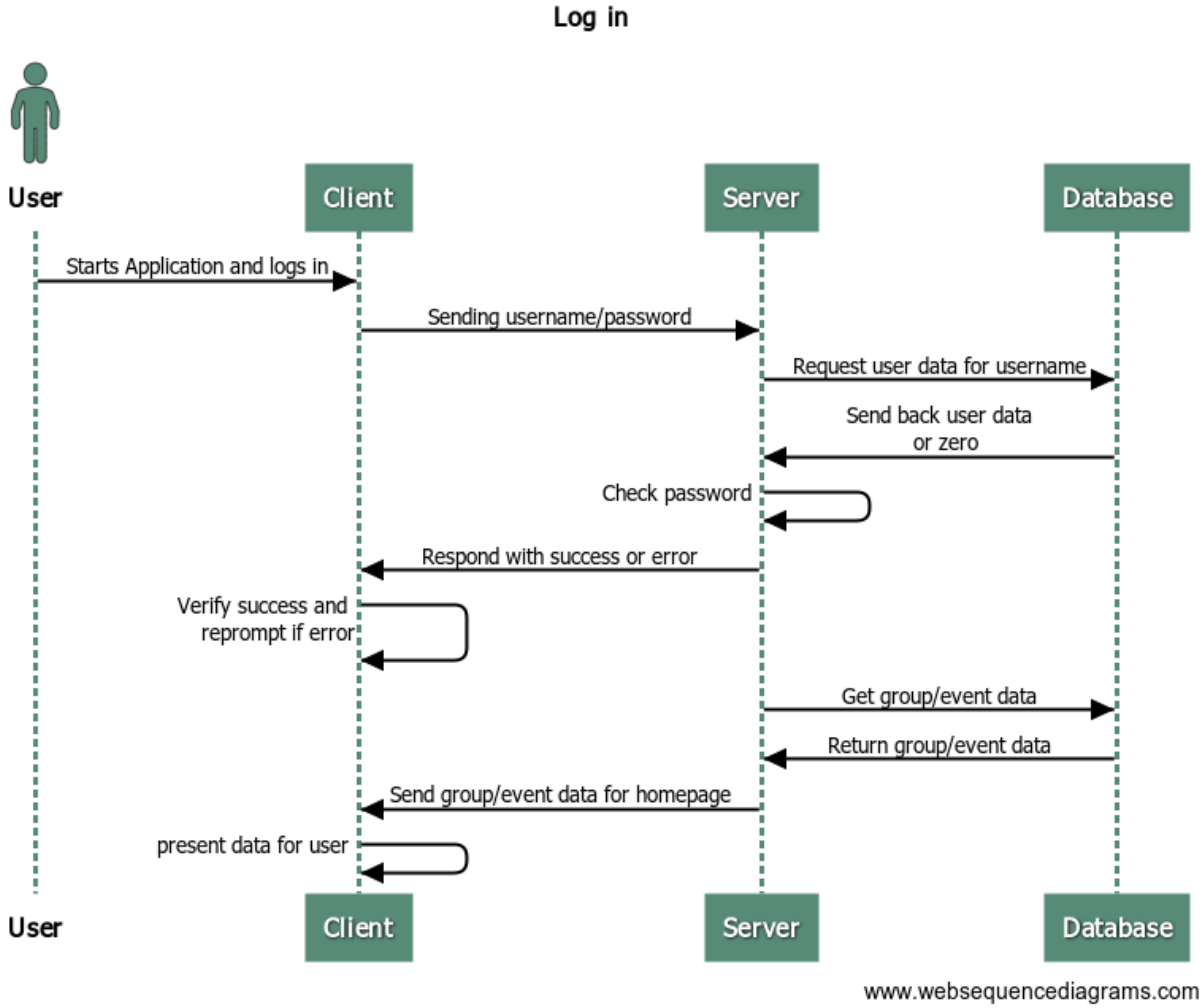
Class Descriptions and How They Connect

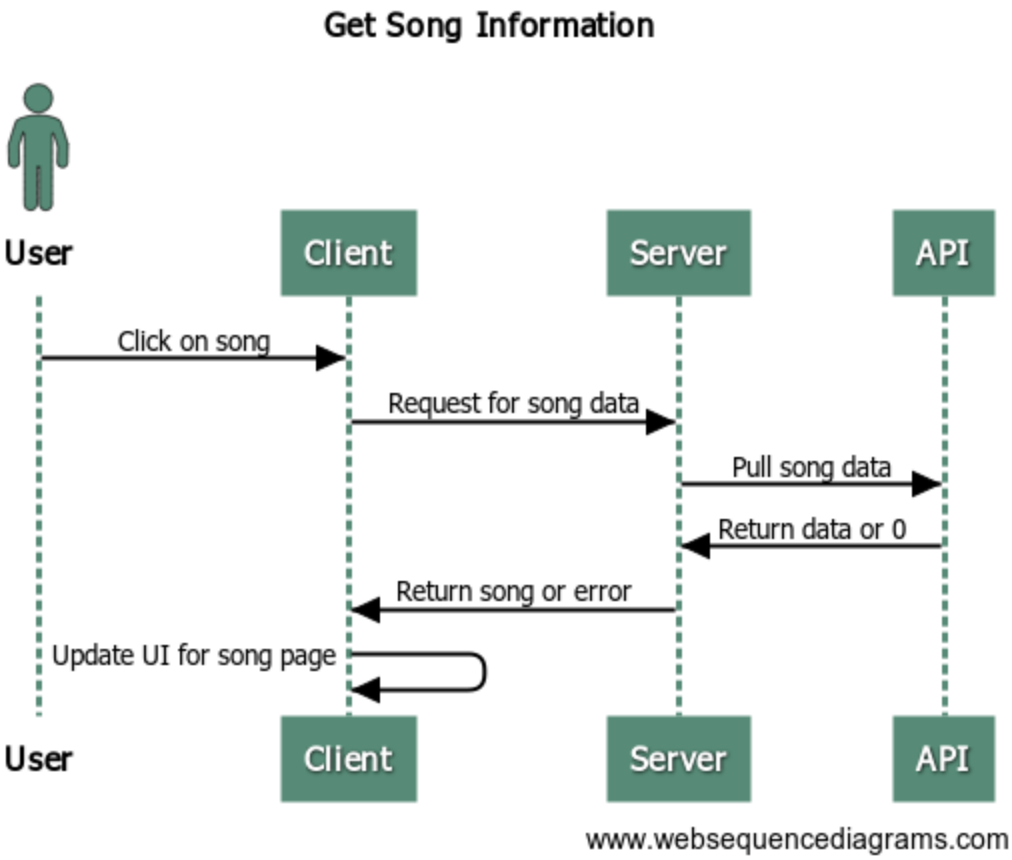
1. **User**
    a. Created when a User signs into our platform.
    b. User fills out appropriate information including: username, password, email, birthday, gender, location and University.
    c. Their location will be used for getting distances to events and location based service assistance.
    d. Users will have groups that they can join or create and share music with a generated userID.
2. **Group**
    a. Name of the group is set by users.
    b. Made up of specific users.
    c. Can be set to private or open and specify what type of group this is.
    d. Can invite users to join group if privacy option is selected

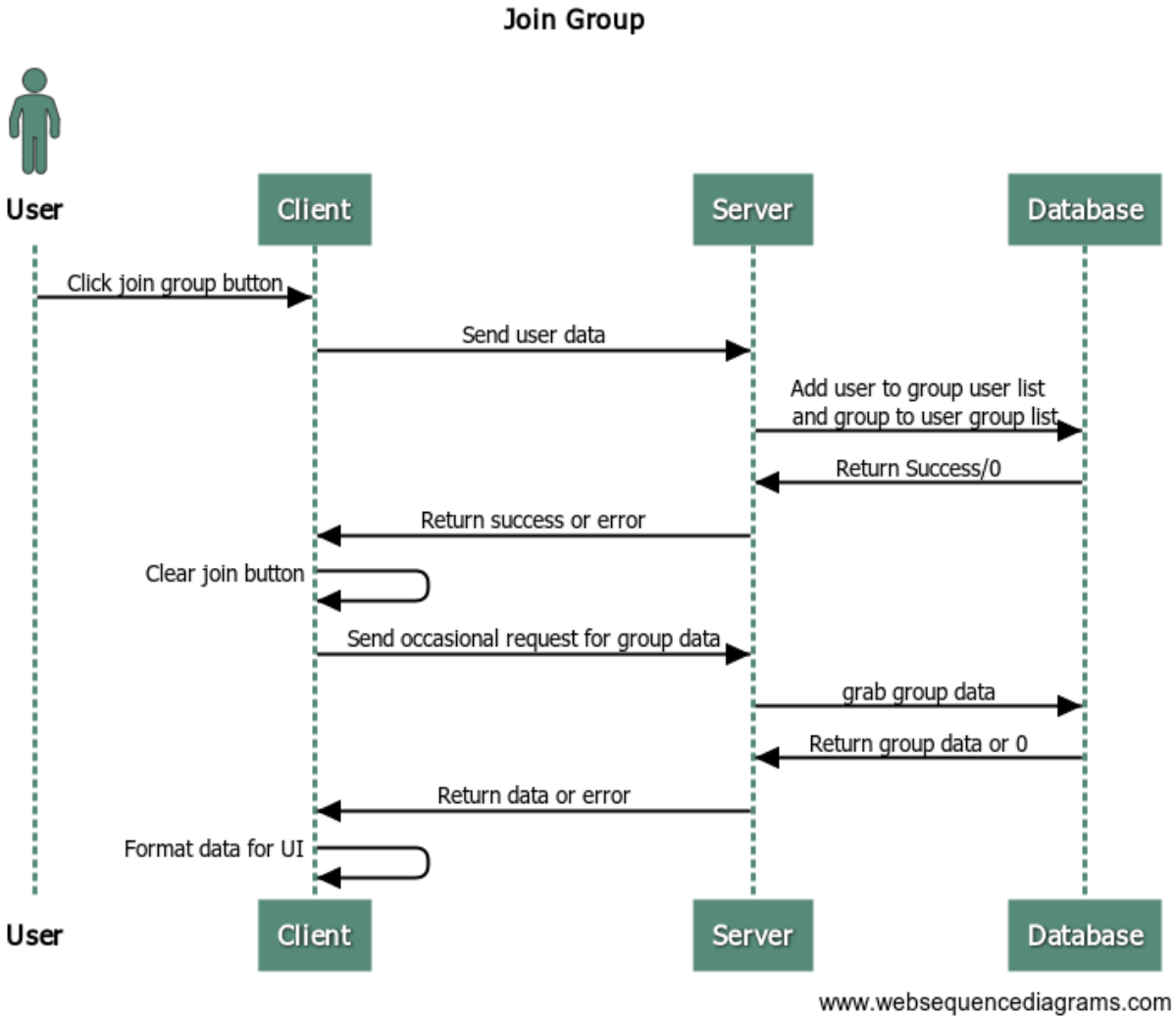
3. **Event**
    a. Each event will have a unique eventNum.
    b. Each event will have relevant information - such as its name, date and location for when and where it takes place.
    c. Each event will be allowed to set a people limit.
    d. Each event will have a string description of it in eventDetails.
    e. Each event will have eventSponsor or eventArtist that describes who is hosting the event.
    f. Events may only be held by verified clubs through our platform.
4. **Privacy**
    a. Checks if the invite being sent from a private/closed group is valid and then adds the users to the group/club
5. **Club**
    a. Validates the key a user provides and is used for more tightly controlled users and groups.
    b. This would include university clubs and groups, as well as businesses.
    c. They must create information about their business and all necessary info that is relevant to the Event being held.

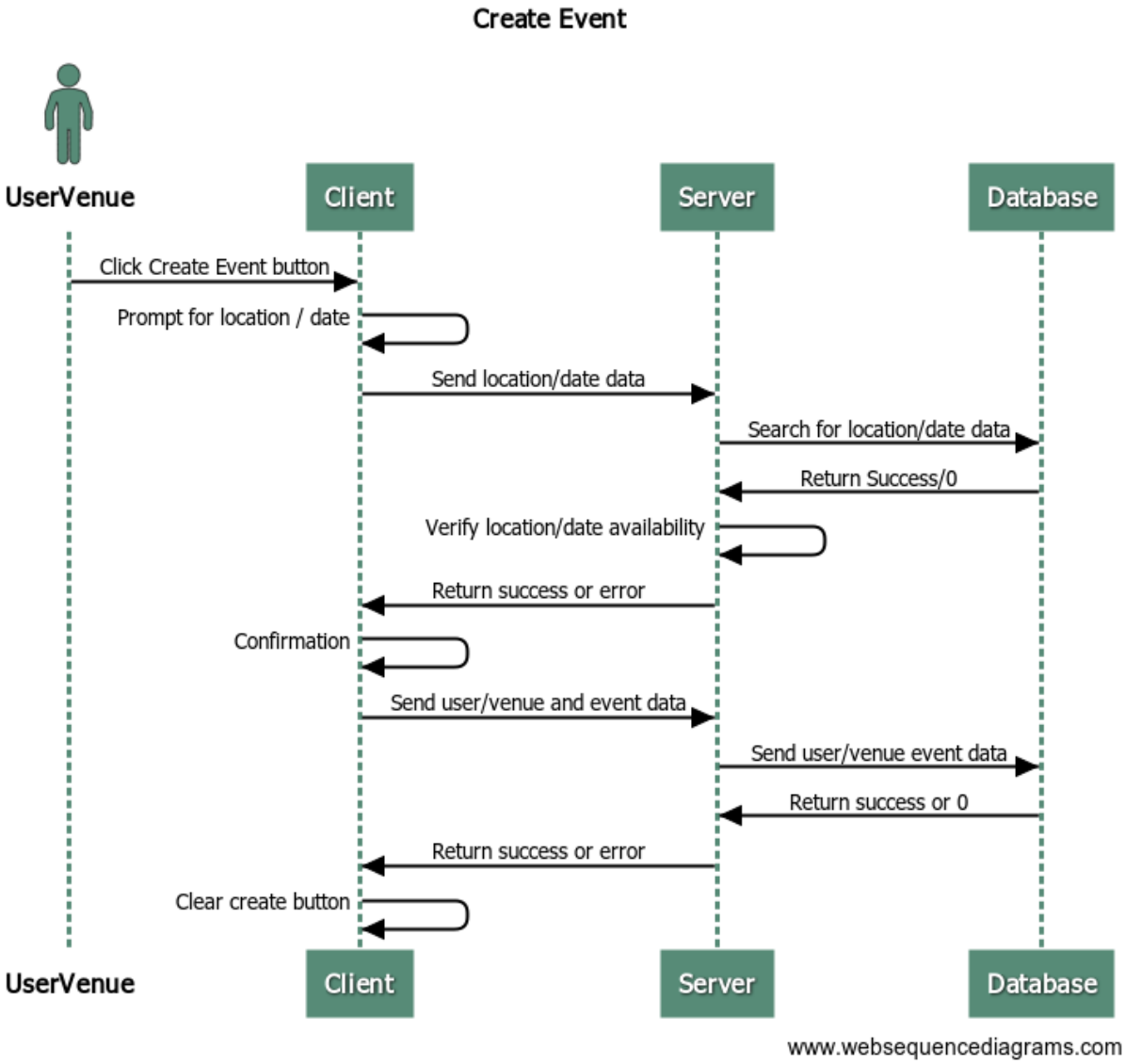
6. **Notification**
    a. Informs users of live events and playlists that are relevant to their interests and tastes.
    b. Also for notifying users if they are invited to a private group/club
7. **Hot Songs**
    a. Generate a playlist by taking the most played songs across all users within the campus location.
    b. Updates the playlist weekly/biweekly to keep up with new and popular music.
8. **Location**
    a. Used to gather user GPS data location for easy venue distance tracker and to help limit artists playing too far away from them to show up on their screen.

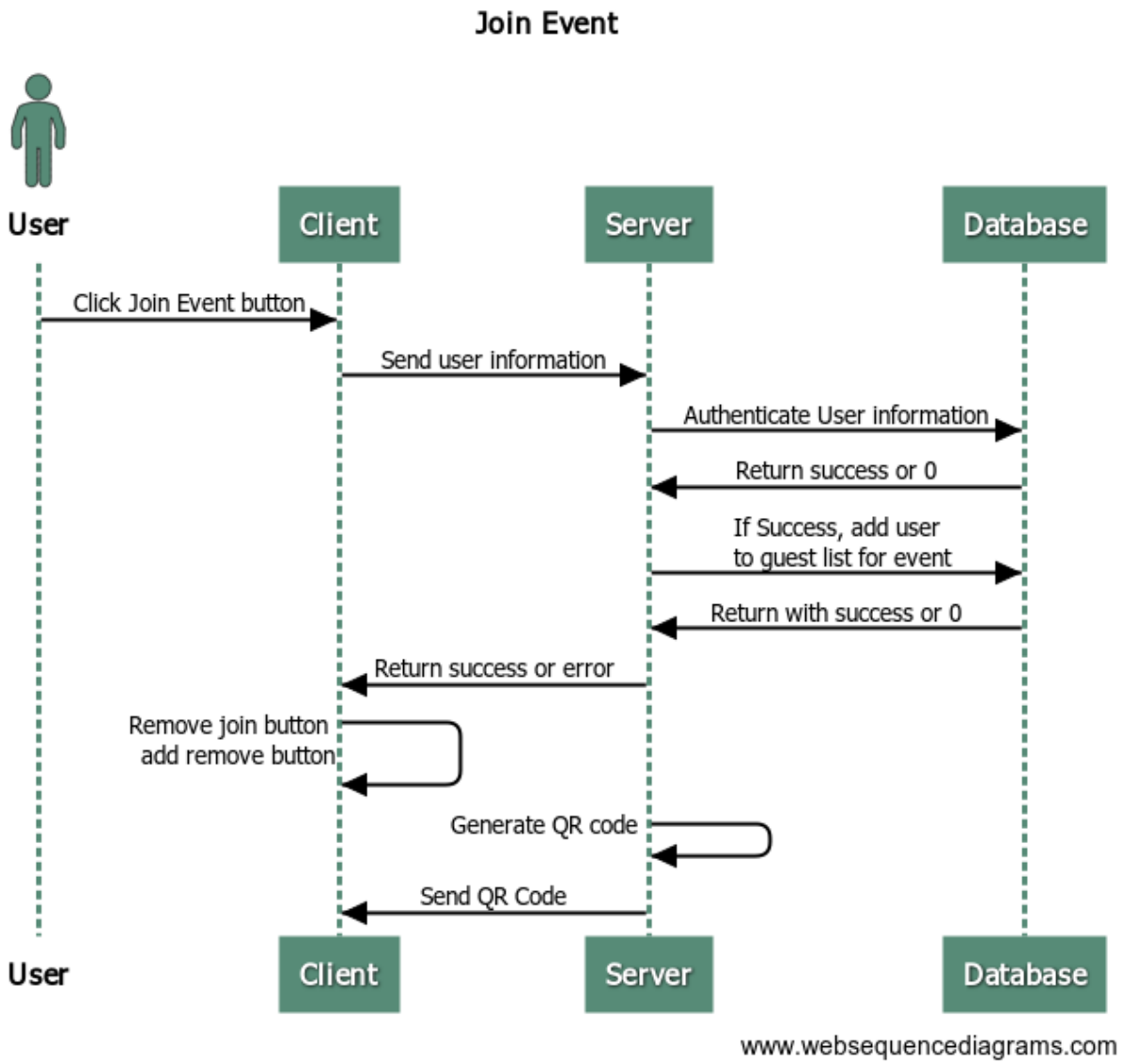
Sequence of Events




Get Song Information
User
Client
Server
API
Click on song
Request for song data
Pull song data
Return data or 0
Return song or error
Update UI for song page
User
Client
Server
API
www.websequencediagrams.com

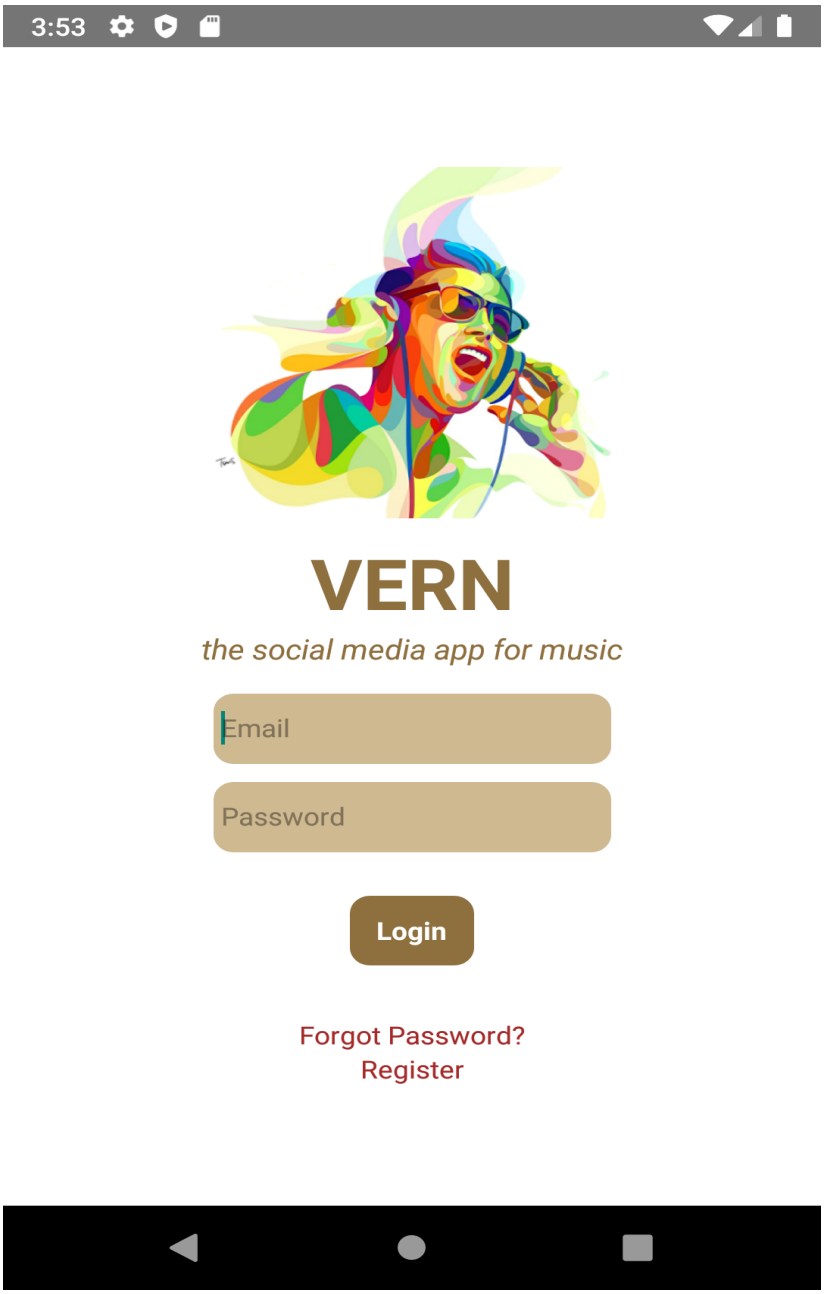

Join Group
User
Client
Server
Database
Click join group button
Send user data
Add user to group user list
and group to user group list
Return Success/0
Return success or error
Clear join button
Send occasional request for group data
grab group data
Return group data or 0
Return data or error
Format data for UI
User
Client
Server
Database
www.websequencediagrams.com


Create Event
UserVenue
Client
Server
Database
Click Create Event button
Prompt for location / date
Send location/date data
Search for location/date data
Return Success/0
Verify location/date availability
Return success or error
Confirmation
Send user/venue and event data
Send user/venue event data
Return success or 0
Return success or error
Clear create button
UserVenue
Client
Server
Database
www.websequencediagrams.com


Join Event
User
Client
Server
Database
Click Join Event button
Send user information
Authenticate User information
Return success or 0
If Success, add user
to guest list for event
Return with success or 0
Return success or error
Remove join button
add remove button
Generate QR code
Send QR Code
User
Client
Server
Database
www.websequencediagrams.com

UI mock-ups

- Login Page



Server/Database implementation details

- Use an external service like AWS to deploy our server. The server will (hopefully) be under a lot of stress since we'll have to constantly generate playlists and make requests to the Spotify API.
- Our users will interface with our frontend built in React Native and will make requests to the Spotify API and we will store necessary data from requests in the database.
- In addition to storing data returned by the Spotify API, we will also be storing data collected by our app such as the user's location, college, and music interests.
- Our database will also host data along the lines of discussion boards, ratings, etc.