Team Proposal

INSO 4116: Software Design
Professor: Marko Schütz-Schmuck
Created by: Team F

Communikey Web App

September 17th, 2021

Team Members

- Carolina G. Santiago Pérez
- Christian E. Rosado Feliciano
- Gabriela N. Flores Velázquez
- Santos O. Martínez Braña
- Sebastian O. Negrón Seda
- Songiemar S. García Curbelo

Informative Part

Project Introduction

Communikey is a web app with the primary intent of serving as a medium for music listeners of all genres to come together and bond over their love for music. Users will be able to share their favorite tunes as well as contribute their thoughts. From sparking discussions with friends, to creating a community for casual listeners to music buffs alike. Communikey provides a space to truly forge and appreciate the connections created by music.

Current Situation

As it stands, music streaming platforms allow users to listen to their choice of music, as well as discover new and similar artists. These may also provide the functionality of viewing what their friends are listening to, even creating personalized playlists blending two users' musical preferences. Nonetheless, this social aspect is a secondary focus to their main goal: streaming music. Music is able to bridge the gaps between people by transcending personal differences. By prioritizing this social facet, users will be able to interact with others, create new bonds, and even widen musical horizons based on others' shared tracks. Additionally, in September 2021, YouTube took down music bots on Discord, a platform often used by people who want to stream music together. Communikey would allow users to listen together once again, by providing a platform to communicate, share music, and create listening parties.

Outline

Needs

- Users can benefit from components found in social media platforms such as chats, groups, profiles but do not have to worry about violating copyright laws for songs that are available to stream in Spotify. At the moment, there are no platforms that allow users to freely stream music with others. Discord, one of the most popular platforms, was recently revoked of this capability.

Ideas

- Spotify's API could be incorporated into the web application to seamlessly access, stream and organize songs. By doing this, music would be easily accessible in different contexts such as messaging and live sharing, without having to leave the platform and worry about copyright laws.

Log Book

1. August 25, 2021- Initial meeting, discussing individual A3's.
2. August 26, 2021- Choosing which A3 to pursue.
3. September 8, 2021- Dividing the parts of the team proposal.
4. September 16, 2021- Reviewing and editing team proposal.

Assumptions and Dependencies

Assumptions

- **Resource assumption:** It is assumed that all team members will have the availability to work on the project a minimum of 5 hours per week.
- **Budget assumption:** It is assumed that the overall project cost will be zero (Is it?)
- **Scop assumption:** It is assumed that the scope of the project will not change at any time.

Dependencies

- **Logical dependency:** Project implementation can't start until the domain has been properly defined
- **External dependency:** For the music playing feature, the team must select an external music library before the implementation of this feature starts.
- **Cross-team dependency:** Both backend and frontend must finish their task implementation before testing is done of that particular feature.

Descriptive Part

Rough Domain Sketch

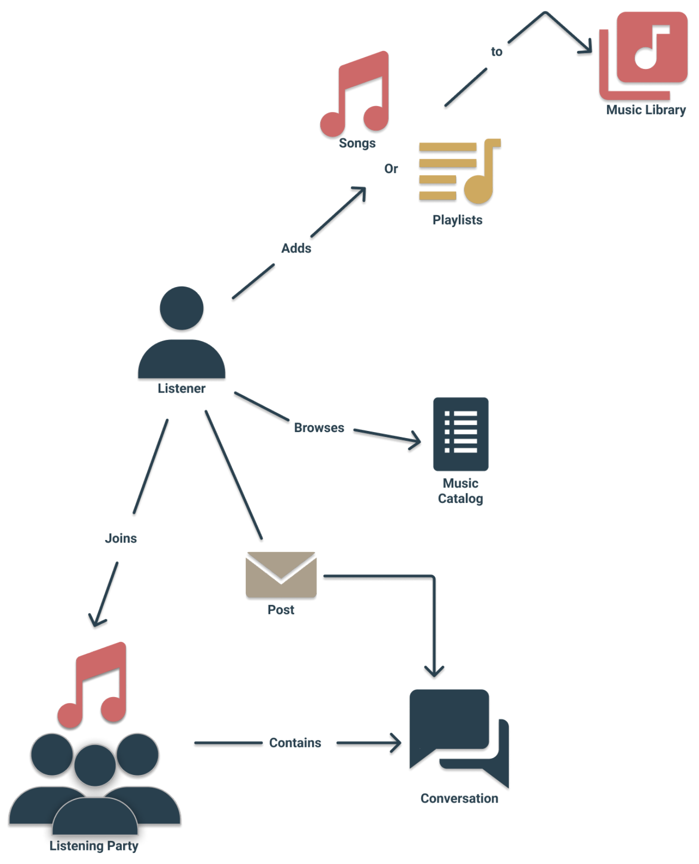

Figure 1. *Rough Domain Sketch Flow Diagram*

Domain Narrative

The domain is encompassed by music spaces composed of music enthusiasts. Music enthusiasts create discourse related to their music tastes, or distastes, and discuss opinions, ideas, recommendations, etc. There would be music catalogues to be accessed and listened to, discussion threads where the topic would be discussed, chats to have direct conversations with other users in the platform. There would also be playlists users could add songs from the music libraries available.

Domain Terminology

Since our platform will be a digital music service that allows people to connect and interact while listening to their favorite tunes, we need to define the following domain terms:

Term	Phenomenon of Domain	Definition
Song	Entity	Content that will be consumed by listeners.
Listener	Entity	Streams songs and interacts with other listeners.
Music Room	Entity	Represents the place where listeners gather to hear songs.
Playlist	Entity	collection of songs stored by the listener.
Library	Entity	Where listeners store their favorite playlist and songs.
Message	Entity	A text created by listeners.
Conversation	Entity	Collection of messages that two listeners shared between themselves.
Post	Entity	A public thread-like message that other posts can be added to.

Domain Entities (Remove if staying with table)

- Listener - streams songs and interacts with other listeners.
- Song - content that will be consumed by listeners.
- Music Room - represents the place where listeners gather to hear songs.
- Playlist - collection of songs stored by the listener.
- Music Library - where listeners store their favorite playlist and songs.
- Message - a text created by listeners.
- Conversation - collection of messages that two listeners shared between themselves.
- Post - a public thread-like message that other posts can be added to.

Domain Functions

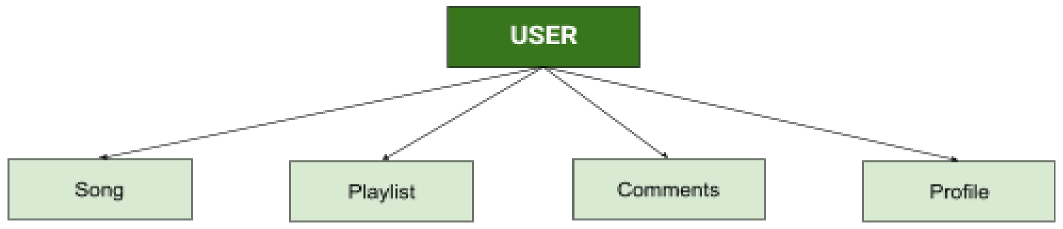

Figure 1. *Domain Functionalities Description Flow Diagram*

- The user plays plays a song
  - PlaySong: User x SongList -> Song
- The user searches for a song
  - SearchSong: User x NameOfSong -> Song
- The user searches for a category
  - SearchSongByCategory: User x Categories -> Categorized Song Results
- The user creates a playlist
  - CreatePlaylist: Array x Song -> Playlist
- Display how much left is remaining from the song
  - ShowSongDuration: SongLength - TimeElapsed -> TimeRemaining
- The user comments on the song
  - MakeComment: User x Comment -> Comment Section
- The user interacts with other users that like the song.
  - UserInteractions: User x User -> Interactions

Functionality Descriptions

- **Play Song**:
  This functionality is almost self explanatory, however it consists of downloading a song from the library that contains thousands of songs.

- **Comment**:
  Users want to express their ideas and be able to communicate with other users that enjoy the same things. We want to make meaningful interactions between users.

- **Share Songs**:
  A great idea to obtain new users is by sharing songs with other people. In addition, it's a great way to discover new music and interact with others.

Domain Events

- Song searched
  - Trigger: In order for the user to play a song, it needs to be searched.
  - When a user searches for a song, it gives him the opportunity to hear the song he/she/they want to listen to.
- Song loaded

    - Trigger: In order for the user to hear the song, it needs to be loaded.
    - After a user selects the song, it needs to be loaded before it can be heard.
- Song played
    - Trigger: User finishes hearing the song.
- Register user
    - Trigger: User decides to register for the application.
    - The user creates a profile where the application can save songs attached to his user account.
- Login user
    - Trigger: User enters credentials in form and access his/her/their profile.
    - Fill the form with credentials.
- Comments on song
    - Trigger: User enters a message taking his opinion on the current song in a textbox.
    - Gives the ability to engage with other future listeners.
- Playlist created
    - Trigger: User creates an array of songs that he likes and wants to hear together.
    - Gives the ability to create a curated array of songs the he/she/they want to listen to.

Domain Behaviors

- Listeners that want to listen to music will use the application, stream it from our library and then they can listen to the desired song.
- Listeners that want to make a comment about a specific song will use the comment section and express their opinion through a textbox that will display the comment to all users.
- Listeners that want to engage with other listeners in the application can search for a song and find comments tagged to it. Therefore, the users can read or even engage with their own opinions or statements.
- Listeners that want to hear songs in a certain order can select a button that will organize the queue in an established order made by the user or the record label.
- Listeners can create a playlist by selecting various songs that they want to listen to and add to the playlist, filling up the playlist with great selections.
- Listeners that want to share a song can do so by selecting the song they wish and share it.

Domain Requirements

- Landing Page:
    - The system must check if the user is already logged in.
    - The system-to-be must recognize if the user already has an account or if they are a new user in the system.
    - The system must allow the new user to create an account.
    - If the user does not remember the password of the account, the system-to-be must let the user change the password and store it in the system.

- Home Page: The system must allow the user to navigate between the options the Home Page has. This options are:
    - Chat / Direct Messages
        - The system must allow the user to see and enter to their stored messages
        - The system must recognize if the incoming message is from a friend or an unknown person.
        - The system-to-be must let the user navigate through the list of friends and messages to anyone.
        - The system must let the user add or remove a conversation.
    - Music / Library / Discover
        - The system must play the music the user chooses to hear.
        - The system-to-be must let the user search and choose the category of music they want to hear.
        - The system-to-be must let the user add or remove music from a playlist.
        - The system must store in the Library section the playlist the user creates or likes.
        - The system must allow the user to see and comment in the posts that are in the Discover section.
        - The system-to-be must recognize the music the user likes (either because they liked it or because they listen to it often) and based on that, the system must create different playlists with random music.
    - Profile
        - The system must allow the user to change their profile and customize it.
        - The system must let the user search the list of friends they have in the profile section.
    - Settings
        - The system must allow the user to change the username and the password in this section.
        - The system must allow the user to modify the notifications, account privacy, storage, music quality, devices connected to the account, etc.

Interface Requirements

- Landing Page:
    - The system must display the Landing Page only for users that are not logged in.
    - If the user is not logged in, the system must show the login or sign up screen.
    - In the Landing Page, the system-to-be should display the description of the product aside to the login/sign up section.
- Home Page
    - The interface must update the screen to the Home Page section.
    - The interface must display the options that allow the user to navigate through the app.
    - The system-to-be must change to the correct page when the user clicks on a desired option.
    - The interface must update and show the cover of different music everyday.
    - The interface should display messages:

        - Success messages: the interface should display the actions that the system performed was successfully completed, such as creating a playlist
        - Error messages: the interface should also be able to display actions that were not done successfully. An example of this is when a message was not sent.
    - The interface should show when the user adds or removes a friend.
    - The interface must display when the user adds or removes a song from a playlist.
    - If the user lost the internet connection, the interface must change the status to disconnected.
    - When the user (or a friend) changes something in the threads, the interface must update and display the change.

Machine Requirements

- To achieve availability, we must have down times of less than 0.01% in the system in a 24/7 shift.
- Response time has to be kept under 100ms for a connection with 10/1 Mbit down/up-stream bandwidth.
- Having high availability in mind, we must create at least one backup per service that stores state. Examples of this is, creating at least one extra database that stores the data input but does not delete anything so that if the running database gets breached and deleted then we can roll back to a previous version.
- Each message in the message system must allow for a minimum limit of 2000 characters.

Software Architecture Design

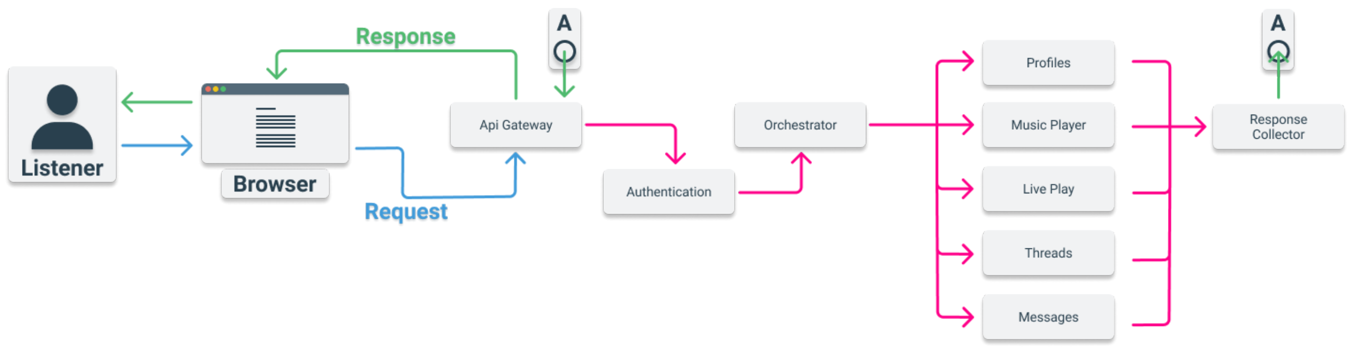

Figure 3. *Software Architecture Design Flow Chart*

Software Component Design

1) **Api Gateway -** Application entry point to the system. We provide all of the possible actions that the application can take.
2) **Authentication -** Receives all incoming requests and identifies and tracks the listener in the system.
3) **Orchestrator -** Focuses on routing the request to the appropriate service if the current listener has permissions.
4) **Profiles -** Stores the information related to the listeners profile page.
5) **Music Player -** Contains the logistics needed to stream and search for music.
6) **Live Play -** Coordinates the streaming of content such as music between subscribed listeners in real time.
7) **Threads -** Real time post driven events preserving relational hierarchy.
8) **Messages -** Group chats or direct messages between listeners are stored and manipulated here.
9) **Response Collector -** Collects all of the individual responses from the main request and converts them into a single response that can be presented back to the listener to consume.

Selected Fragments of Implementation

We will be implementing this web application using React as our web framework to create reusable components throughout the application. GraphQL will be used in the backend as the standard of communication to access the api. The backend will orchestrate its services using kubernetes to help load balance each microservice automatically. Each service is a program that can be written in any technology as long as the input and output of the respective api is consistent with the GraphQL specification. Most of the services will be written in javascript using node to keep the language complexity to a minimum. If a service needs to store state then we may consider between sql or no sql database alternatives such as SQLite or MongoDB.

Analytical Part

Concept Analysis of Descriptions

The main participants in the domain are the users, or the music enthusiasts, who can create, view, and discuss in different circles centered around music. A possible conflict might be that there could be discussions that have little to do with music, or people that could “spam” these threads with irrelevant or inappropriate information. Another possibility is that of people releasing songs illegally, whether it’s unreleased music or music that does not belong in the artist’s account with its proper credit.

Verification

In order to understand how efficient the software is, each section from the rough sketch domain has to be evaluated. The evaluation will have different criterias. Our software will be directed towards users, for such reasons the evaluation should be done by listener users.

Key things that may have to be addressed

- Users when they are logged in to the music app can access the music selection and play the music they want.
- Once a user logs in, he/she can see the different playlists, threads and friends.
- Users can chat with other users around the world.
- The music recommended playlist is created randomly and the user can play it.
- When the server reaches its limited capacity, the database and backend features will be able to handle the user demand.
- The user can use the music app with ease because of its intuitiveness.

The following evaluation has to be done as soon as possible after the launch of the application. The music app will be evaluated as follows:

Verification Scenario	**Yes/No**	**Countermeasure if scenario failed**
Did the user encounter an error when searching for the desired music?		The back-end for the music database will be verified. Alongside with the verification of using the correct functions to play the music in the front-end.
If a music is no longer available and it was stored in the playlist, is the user notified about the change?		If the user is not notified, the user should verify spam folders for the notification. If it is not found, the tools used to send the email from the front-end and back-end will be fixed.
Can the user write to other users?		Verify if the chat was successfully created and assigned to the correct user.
If a friend is no longer active and the user sends a message, the user receives a notification about the inactivity of the friend?		If the user is not notified, the user should verify spam folders for the notification. If it is not found, the tools used to send the email from the front-end and back-end will be fixed.
Can the user access the music database?		If the user sees no music available. The database will be verified in order to identify the root of the problem.
Can the user access other profiles and see their music playlists?		Verify if the other user has his profile public. If the user has his profile public, then the front-end will fix the function that lets the profile be visible.
Can the user view the public threads?		Verify if the threads are still in the system and were not deleted. If it is still in the system, then the front-end will be fixed.

Can the user successfully add, remove music from the page?		Verify the functions used to access the database, and fix the issue.
Does the user consider the music app easier to use than other music apps?		If the user prefers other music apps ask the user why. Afterwards ask for feedback on where the app fails to satisfy the users preferences or needs.

Validation

How do you plan to determine the project was successful?

User-Scenario	**Yes/No**	**Scenario justification**
Can users with no previous interaction with the proposed system navigate through the app and communicate with other users?		If the user can use the proposed software with ease and can communicate with other users with no problem, the app was a success.
Are the users satisfied with the music presented in each category?		If the user is satisfied with the categories, then it is a success because it has variety.
Can a user remove an unwanted friend or chat after joining one?		This scenario tests the functionality of the software.
Do users prefer to chat, communicate and share their tastes in music or just listen to music?		This scenario looks to validate the whole software because that is the purpose of the music app.
When the user accesses the music section, is the music searched accessible in less than 100 milliseconds?		This scenario looks to measure if the loading time of the music is acceptable. This validation checks the latency between the database and the user.

To access the system, the user must enter his account or create a new one. After login, the user will have seven options to select:

- Chat/Home - This is the home page and also shows the user chats/discussions.
- Music - This section has two options. Music Categories where the user can choose one and the Search Music option when the user writes the name of the song or the author. Also, you can create playlists that are stored in the Library.
- Discover - Shows threads that are public and currently trending. Discover friends and participate in their threads.
- Library - Stored all of your playlists and liked songs.
- Direct Messages - See the friend list and direct messages as well as group messages.
- Profile - You can view and edit your public profile that gets displayed to others. Profiles can have themes and edit as they want to create their own experience.
- Settings - Users can change aspects of their app here. Settings affect the entire app.

Roles

Gabriela N. Flores Velázquez- Domain Requirements, Interface Requirements, Verification, and Validation.

Songiemar S. García Curbelo- Introduction, Current Situation, Log Book, and Roles.

Santos O. Martínez Braña- Machine Requirements, Software Architecture Design, Software Component Design, and Fragments of Implementation.

Sebastian O. Negrón Seda- Assumptions and Dependencies, Domain Terminology, and Domain Entities.

Christian E. Rosado Feliciano- Domain Functions & Descriptions, Domain Events, and Domain Behaviors.

Carolina G. Santiago Perez- Outline, Rough Domain Sketch & Domain Narrative, and Concept Analysis of Descriptions