Spring 2021

Proximity-Based Video Communication with CocktailParty

Addison Fabry
University of South Carolina - Columbia

William Heffernan
University of South Carolina - Columbia

Andrew Shroyer
University of South Carolina - Columbia

Director of Thesis: Jose Vidal, Ph.D.

Table of Contents

Thesis Summary

CocktailParty is a video communication application designed to uniquely and efficiently solve the problems faced by traditional online video communication. The proposal for this start-up project was accepted in August 2020. CocktailParty enables users to join a video chat room overlayed on top of a virtual house, in which each user can move their own video feed around the house. Each room in each house layout hosts a different video call, allowing users to easily host large virtual gatherings that facilitate multiple conversations taking place simultaneously. This functionality makes important steps towards effective simulation of the real-life gathering experience.

Introduction

In March of 2020, amid the onset of the COVID-19 pandemic, many private and public institutions closed their doors. Many states and countries instituted limitations on social gatherings, forcing many socialites to find an alternative to meeting in person. The world, consequently, saw a massive migration from day-to-day face-to-face interactions to video conferencing applications such as Zoom, Microsoft Teams, and Cisco WebEx. The transition to online working, learning, and socializing has not gone smoothly for all involved. Awkward Zoom meetings, boring virtual birthday parties, and similarly underwhelming video chats have highlighted a significant vacancy in the online video communication market.

The CocktailParty web application is a communication software built around the idea of making virtual communication as easy as in-person conversation. Our goal in the planning and development of CocktailParty was to emulate the role that proximity has in the real world on group conversation: if two groups are in two different rooms, they should be able to hold independent conversations, but participants should have the freedom to migrate between the groups with ease. CocktailParty addresses this experience by seamlessly switching users between video calls whenever they move their video feed into another room in their virtual house.

Methods & Approach

Feature Overview

When hosting a call, users can choose from three different layouts, as shown in Fig. 1. Other users, in turn, can use a uniquely generated code to join a call. Once joined, each user can move their own video feed around the house using the arrow keys. All layouts are partitioned into different video call sessions by room—if a subset of users desire their own space for conversation, they can simply move their feeds to another room of the house. Doors to each room can be closed, preventing other users from entering without permission. CocktailParty also features user friend list management, direct messaging, as well as account and device management capabilities.

Fig. 1. All three CocktailParty layouts.

Technologies

Vidyo.io

Vidyo.io is a video chat service that exposes a public API enabling developers to connect multiple devices in one video chat room. This service was chosen due to its low cost and ease of setup; each incoming and outgoing video feed can be rendered to an HTML element [1], which we determined would ease the challenge of integration with Lance.gg player movement.

Lance.gg

Lance.gg is a Node.JS-based real-time multiplayer gaming server and client-side library that facilitates network synchronization between connected participants. To provide users with a smooth visual experience, Lance.gg optimizes networking, position interpolation, user input coordination, and pseudo-physical movement. The simple development model provided in the Lance.gg documentation simplifies the processes of code analysis and debugging [2]. CocktailParty utilizes Lance.gg for much of the player movement, player collision, and wall collision features.

Google Cloud Platform

App Engine

App Engine is a cloud computing platform that we used for hosting the web application on a publicly accessible server [3]. We initially planned on using Google's Firebase Hosting for our public website, but we migrated to App Engine upon discovering that Firebase Hosting does not support WebSocket connections, which is essential to facilitating player movement with Lance.gg. Hosting with Amazon Web Services was also investigated, but we decided against it due to its lack of free HTTPS protocol configuration (a requirement for Vidyo.io connections) for custom domains (a preference for ease-of-access) [4].

Firestore

Firestore is a storage service that we used to facilitate user authorization, store user account information and preferences, and to track currently running video call rooms as well as their participants [5]. The database was partitioned into Collections by Users, Conversations, and Rooms. The Users Collection is organized by user ID, and it stores data for all the users of the website. For each user, these data include first and last name, their unique friend code, a list of user IDs of each friend, a list of user IDs of each inbound and outbound friend request, and the URL of their profile picture. The Conversations Collection contains the data for the messages between users. These data are organized by conversation ID, and each conversation document contains all messages, as well as a variable for the last message sent, which is displayed in the app in the conversation list. The Rooms Collection stores the data for all the call rooms, which

includes the layout identification number, a list of the user IDs of each participant, and a subcollection of the participants in each call zone. For more information, see Database Schema in the Appendix.

Bootstrap

Bootstrap is a commonly used library for use with Cascading Style Sheets (CSS) of the front-end components of web applications [6]. Bootstrap made it easy to design and customize the user interface with extensive pre-built components, allowing the team to focus on back-end development.

GitHub

GitHub is a source control tool we used to host our project repository. GitHub allows developers to iterate upon their own local repositories and later integrate them into the shared repository, enabling our team to develop independent features simultaneously [7]. In addition to storing all branches of our application code, GitHub provided us with tools to track open development issues and log our planning documentation as well as our own development progress.

Selenium

Selenium is an open-source framework we used for running behavioral testing on our web application [8]. Selenium's WebDriver infrastructure enables the automated control of a web browser, allowing us to automatically test app functionalities in the same manner that an end user would [9]. Our usage of Selenium is discussed in further detail in the Behavioral Testing section of Testing and Validation.

Application Interface

CocktailParty consists of several different views that are accessible to all users. The first view is the Login page, and a user must first login or create a new account to access the rest of the views. Once the user is logged in, a navigation bar appears on the left side of the screen with tabs for the Home, Friends, Messages, and Settings views.

Login / Register

These pages are used to login to an existing account and register a new account, respectively. The Login page includes fields for the user email and password, both of which are validated upon clicking the Login button. If authenticated, the user is signed in and navigated to the Home page. Conversely, the user can use the “Sign up now” link to open the Register view. In addition to email and password, this page requires user input for name and password confirmation. Provided that the passwords match and there is not already an account with the inputted email, clicking the Register button creates a new account, signs the user in, and navigates to the Home page.

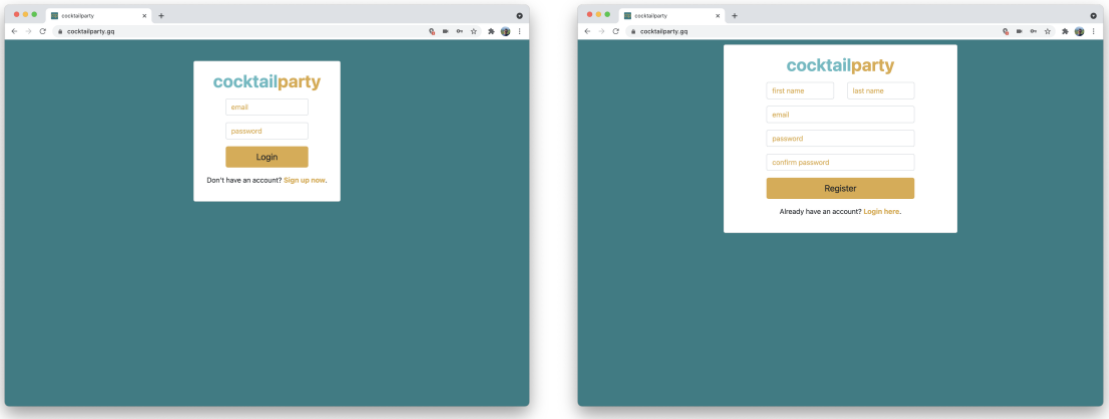


Fig. 2. The Login (left) and Register (right) pages.

Home: Host / Join

The Home tab consists of two buttons that allow the user to host and join a call, respectively. Clicking the Host button opens a pop-up that accepts user input for display name as well as call layout choice. The Create button, in turn, opens the Active Video Call view and generate a corresponding room code. Clicking the Join button opens a pop-up allowing users to input their display name and the code of the room to be joined.

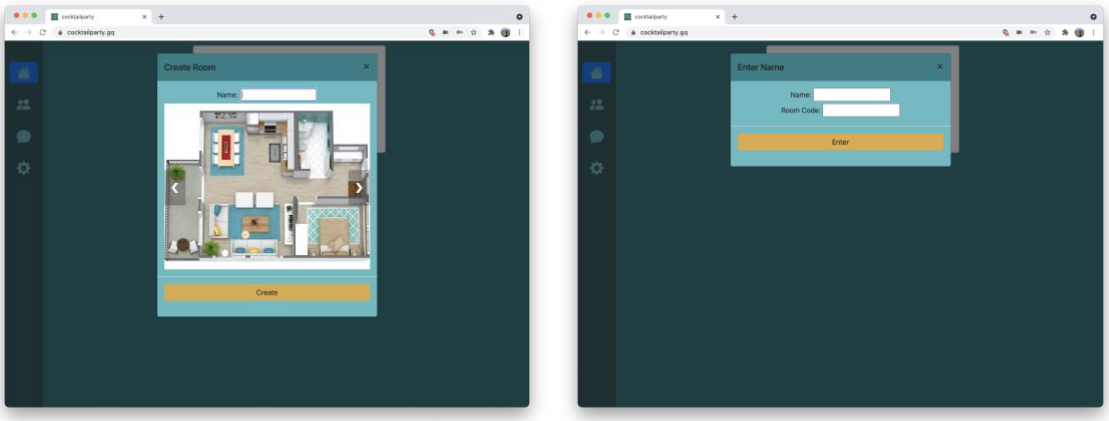


Fig. 3. The Host (left) and Join (right) dialogs.

Active Video Call

The Active Video Call view overlays the Home tab when the user is currently participating in a CocktailParty. This view displays an image of the selected room layout. The participants' video feeds appear in small boxes on the screen, as shown in Fig. 4, with all display names rendering in the corresponding video feeds (except for the current user, who sees a mirrored preview feed in their place). At the bottom of the screen is the call bar, which includes buttons to turn off the user's camera, to mute themselves, and to leave the call. The host of the call also a button to open all doors in the house, which gives the participants the ability to move more freely between rooms.

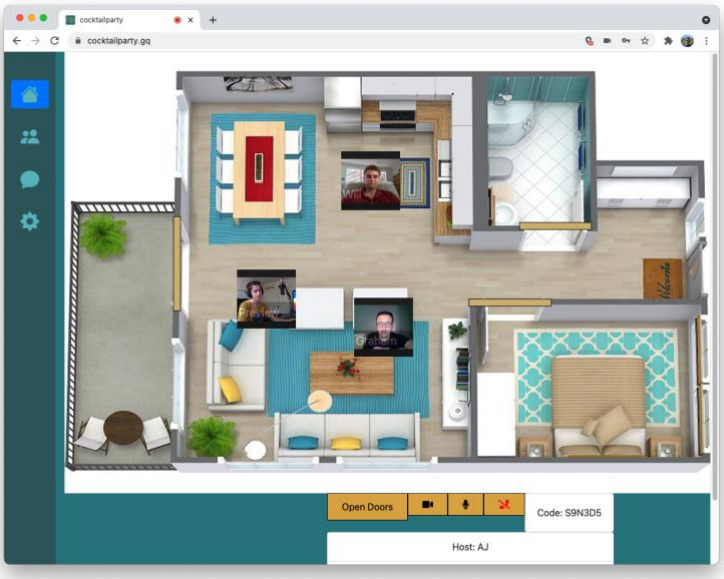


Fig. 4. An active CocktailParty session between three users, with host privileges shown.

Friends

Active users can use the Friends tab to manage their friendships with other users (Fig. 5). The left panel lists all the user's friends. The two icons to the right of the friend's name are buttons to message the friend and remove that user as a friend. The right panel includes a list of the user's friend requests. As shown below, the arrow next to the request to "alpha" indicates that the current user is the one who sent the request. If the user was to receive a request, there would be a green check next to the red X button that would allow the user to accept the request. At the top of the panel is a field to input other users' unique codes to send friend requests. Below that, the current user's code is displayed.

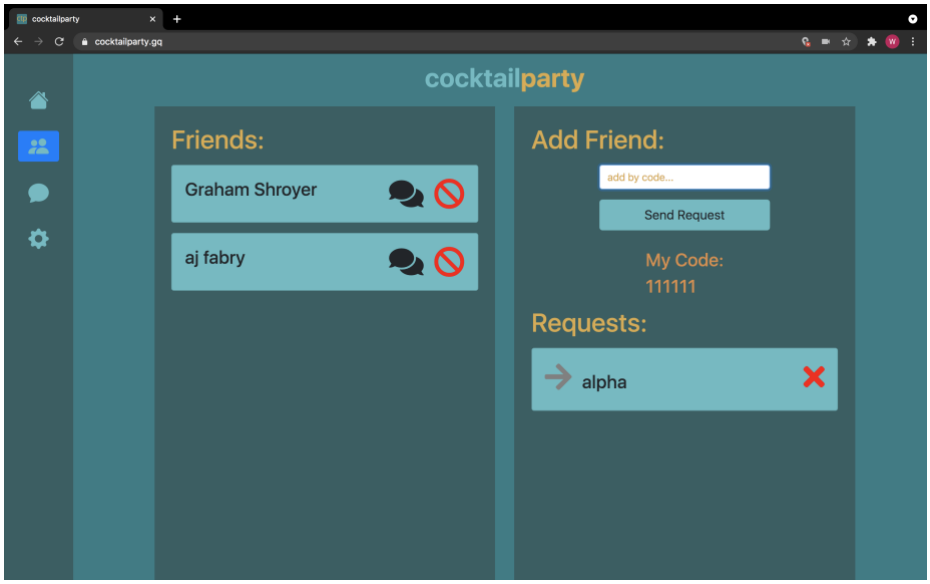


Fig. 5. The Friends page of one user, featuring two current friends and an outbound request to user "alpha."

Messaging

The Messaging tab allows users to exchange direct messages with each other. On the left side of the screen, all the user's current conversations are displayed. The currently selected conversation is highlighted in orange. On the right side of the screen, the messages in the current conversation are shown. The messages sent by the user are displayed in blue on the right and messages received are in orange on the left. The field at the bottom of the screen allows a user to write and send a message.

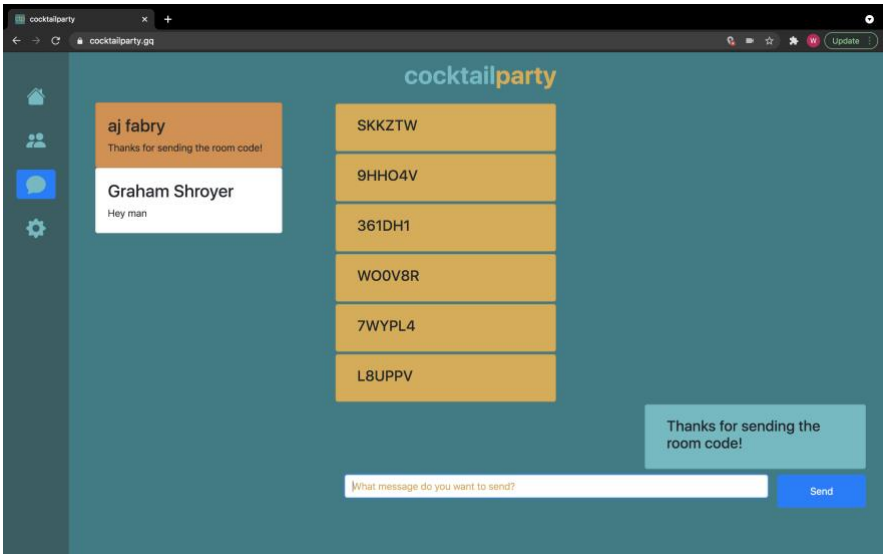


Fig. 6. A direct message exchange between two users. Here, a string of past room codes is shown.

Settings

The Settings tab contains two panels for account and device management, respectively. The top of the Profile panel displays the user's name and profile picture. Below that are buttons that let the user change profile picture, name, and password; additionally, this panel can be used to log out of the app. The Manage Devices panel has drop-down menus to change the video and microphone input devices, as well as the audio output device. Device preferences can only be changed once device detection has been performed, which runs once a user has connected to their first call of the session.

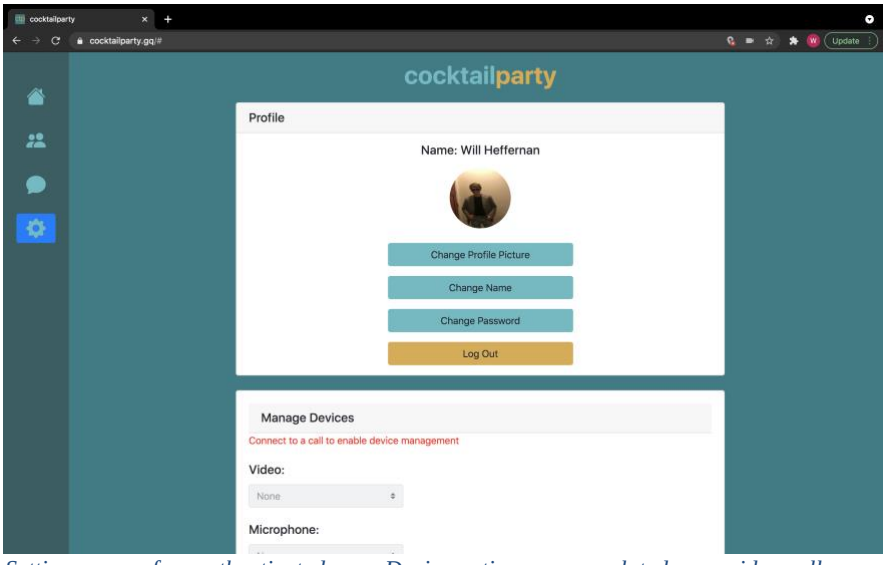


Fig. 7. The Settings page of an authenticated user. Device options are populated upon video call connection.

Technical Obstacles

Synchronization of Player Rendering

While Lance.gg is designed to shift the burdens of player identification and net synchronization away from the developer, the challenge of properly integrating this service in with Vidyo.io remained. When a user connects to a call, they may detect the incoming video feeds of other users essentially in random order. In traditional video chat software, this is no issue—the place on the screen in which each participant is rendered can be different for each client; users are not in control (nor need to be) of their position on any screen. In CocktailParty, however, it is imperative that each participant must be rendered to the element that they are in control of. With little technically identifying information that came with each camera feed detection (a randomly generated ID that changes between every call), this proved to be a challenge.

To ensure proper player rendering, we designed a new procedure that pairs camera feed identification with Lance.gg player identification information in Firestore as soon as a new user connects to an existing call. This algorithm only runs on the machine that is hosting the call, which prevents other clients from mistakenly overwriting the data. While any user is connecting to a call, their client consults the database to determine the HTML elements that each player should be rendered in.

Proximity Functionality

CocktailParty was originally planned with the goal of having the application automatically adjust the volume for each participant in a call relative to the on-screen location of the other participants. The closer a participant was to the current user, the louder that participant would sound with the volume of the participant being proportional to their Euclidean distance from the current user. However, using the Vidyo.io framework, it was determined to be infeasible to isolate and manipulate individual incoming audio streams. Consequently, we were unable to alter the volume of each participant.

Our solution was to separate each room layout into separate room-defined zones, as depicted in Fig. 8, which each zone hosting its own video call. So, when a user enters a new zone, they leave the current call and join a new one. They can no longer see or hear any of the participants in the original zone; they instead see and hear the participants in the newly entered zone. At this moment, the video feeds of the participants in the original zone are replaced by their profile pictures, so the current user is still able to see where all the participants are in the house.

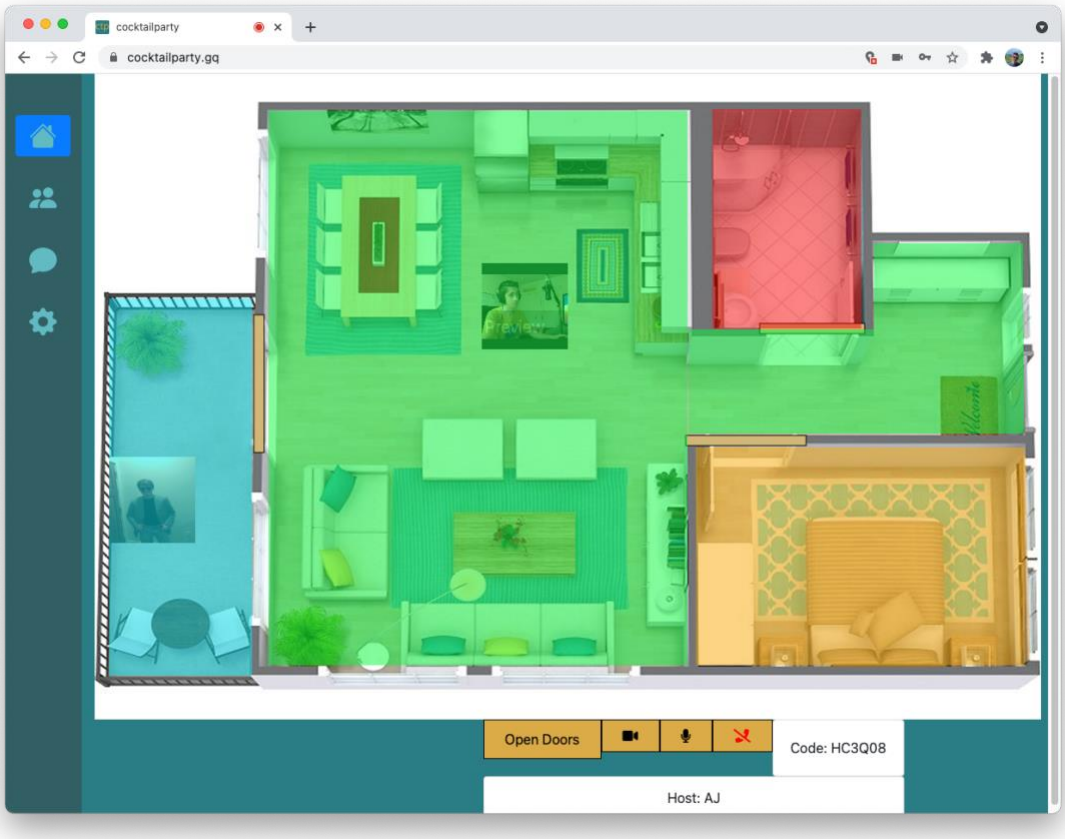


Fig. 8. Proximity chat zones. Each video call area is highlighted in a different color. Note the appearance of one player's profile photo, rather than their video feed, in the blue zone.

Environmental Collision Mapping

One of the core requirements of CocktailParty is that each virtual environment had to be fully interactive; players should not be able to traverse through walls. The virtual environments, however, are displayed using static image elements—this means we

needed to manually map every wall to an in-game object that would prevent players from passing through. Additionally, these objects had to adhere to the same room-based principles that player objects do; they should only be present in calls with their corresponding layout selected.

The first half of this issue was addressed through the development of a game engine method that empowered developers to simply input the dimensions and position of a wall to add it to the world. Client-level functionality was also introduced to detect each wall and render them in green during development, enabling the team to fine-tune the placement of each wall. Assigning each wall to one room proved to be a more arduous task as we found out that the built-in room assignment functionality of Lance.gg was insufficient; it treats all objects on the server as if they are in the same room. Because of this, rooms in Lance.gg, as we discovered, are intended to be run on completely isolated Lance.gg server instances. Since we were working with just one server that had to handle management of all calls simultaneously, we downloaded and directly modified the Lance.gg physics engine to only enforce collisions between objects that are truly present in the same CocktailParty room.

Testing and Validation

Unit Testing

A small number of unit tests were written using Mocha, a JavaScript-based testing framework. Unit tests differ from behavioral tests in that they directly call functions in our code, rather than access our site through an automated browser. These tests were used to verify the status of the website in development and check the validity of the Vidyo.io token generation function (a unique string that allows users to connect to video calls).

Behavioral Testing

Behavioral tests were written with the Selenium framework to cover Login, Register, and Friend page functionalities. Each of these tests simulated user behaviors of site navigation and input, and they also automated behavior validation. All tests, including unit tests, were consistently run alongside the creation of new pull requests to ensure that

no existing functionality could be negatively impacted on the addition of each feature and bug fix.

Debugging

When trying to identify points of failure during the development process, we were able to use Google Chrome Developer Tools to traverse from breakpoints set in our front-end JavaScript code. These tools proved to be invaluable towards the isolation of the causes of unintentional behavior in development. Additionally, they permitted the editing of HTML and CSS, which enabled us to experiment with different front-end layout tweaks as the site was running.

GitHub Workflow

GitHub allows repository administrators to configure a range of actions, referred to as Workflows, that can be automatically performed whenever a user-selected action happens to a branch, such as the pushing of a commit or creation of a pull request. We created a Workflow to automatically deploy an instance of any opened branch pull request to Firebase Hosting. This production-like environment enabled the team to test features in the same manner that they would appear in the public website (once complete and merged into the master branch). Additionally, GitHub checks for potential merge conflicts when comparing two branches in a pull request. These tools aided the team in identifying and fixing bugs in their respective branches, significantly reducing the amount of flawed code that could potentially be introduced into the master branch.

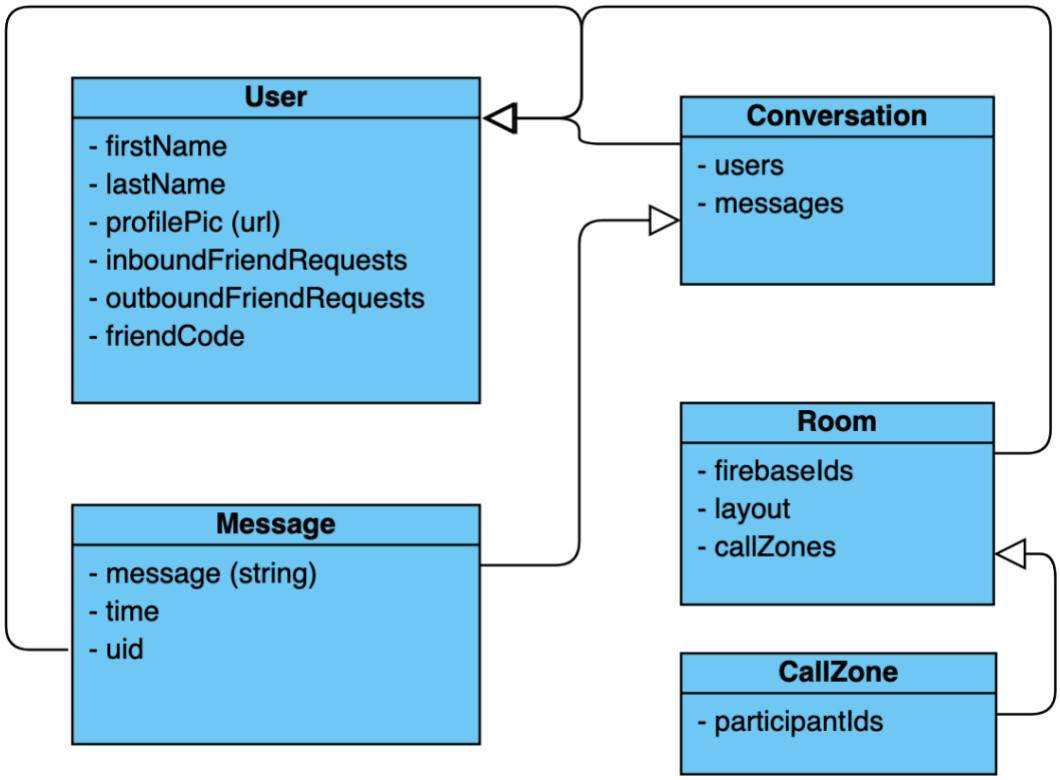
Conclusion

The primary objective of this project was to create an application that meets the requirements constructed by the CocktailParty team: a video conference call that allows users to hear those around them more clearly than those further away in the call room. The application had to be user-friendly, engaging, and reliable in order to establish it as a viable alternative to other video call software. Secondarily, the development of this project aimed to allow group members to apply learned concepts from the computer

science curriculum by tasking each developer with the responsibilities of designing and implementing each application feature from front-end to back-end. The team acquired invaluable experience working with languages, frameworks, and services currently used on the forefront of software development. Furthermore, the opportunity to work in a collaborative environment on a professional-level software application has prepared group members for future careers in software development.

Appendix

Database Schema



References

[1] “Development Center,” Vidyo, Inc., [Online]. Available: https://developer.vidyo.io/#/documentation.

[2] Lance-gg, “Multiplayer game server based on Node.JS,” GitHub, Inc., [Online]. Available: https://github.com/lance-gg/lance.

[3] “Google App Engine Node.js Standard Environment documentation,” Google LLC, [Online]. Available: https://cloud.google.com/appengine/docs/standard/nodejs.

[4] “How do I use CloudFront to serve a static website hosted on Amazon S3?,” 05-Jan-2021. Amazon Web Services, Inc., [Online]. Available: https://aws.amazon.com/premiumsupport/knowledge-center/cloudfront-serve-static-website/.

[5] “Cloud Firestore | Firebase,” 30-Apr-2021. Google LLC, [Online]. Available: https://firebase.google.com/docs/firestore.

[6] “Introduction - Bootstrap,” Bootstrap Team, [Online]. Available: https://getbootstrap.com/docs/4.1/getting-started/introduction/.

[7] “GitHub Documentation,” GitHub, Inc., [Online]. Available: https://docs.github.com/en.

[8] “The Selenium Browser Automation Project,” Software Freedom Conservancy, [Online]. Available: https://www.selenium.dev/documentation/en/.

[9] “WebDriver,” Software Freedom Conservancy, [Online]. Available: https://www.selenium.dev/documentation/en/webdriver/.