Recreating the First Haptic Cinema Experience with Smartphones

By

Jordan Stoessel

A Project Report

Submitted to the Faculty

of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the Degree of Bachelor of Science

in

Computer Science

17 March 2021

APPROVED:

Joshua Cuneo (CS)

Brian Moriarty (IMGD)

Abstract

This report describes the design and implementation of a mobile Web application that recreates the experience of Percepto, a haptic (tactile) cinema technology originally developed for Columbia's 1959 feature film *The Tingler*. Utilizing Google's UI software development kit, Flutter, together with Firebase, React and TypeScript, a client-server architecture was created that allows synchronized control of vibrations across multiple Android or iOS devices. User testing confirmed that the application can reliably activate vibrations on command for an arbitrary number of users, and is compatible with a wide range of smartphones and tablets.

Acknowledgements

The author would like to thank Panhavuth Lau for his many contributions to this project, and the IQP upon which it is based.

Contents

1. Introduction

The application created for this project is a continuation of an Interactive Qualifying Project (IQP), *Percepto The Birth of Haptic Cinema,* completed by the author in collaboration with Panhavuth Lau in AY19-20 (Lau, Stoessel). This IQP studied the history and technology of *The Tingler* (Columbia, 1959), the first Hollywood feature film to incorporate haptic (tactile) sensations.

Figure 1. A poster for *The Tingler* featuring its haptic technology, Percepto. Source: URL.

The Tingler's haptic technology, marketed as Percepto (Figure 1), required the installation of an elaborate network of specialized equipment in the theater. This makes it impractical to recreate the experience for modern audiences. Our IQP concluded by proposing

a method for using smartphone technology to achieve a similar effect with minimal cost or inconvenience.

The idea for the project came from the observation that smartphones, which are now ubiquitous, incorporate vibration devices for user notification. This feature makes it possible to economically recreate *The Tingler's* haptic "shock" effect without the effort and expense of installing electric buzzers under theater seats.

To make a Percepto app accessible to a large audience, it was decided to support both Google's Android and Apple's iOS mobile operating systems alongside another web application. This choice risked a significant increase in the scope and complexity of the project. Android devices are typically programmed with either the Java or Kotlin programming languages, while iOS requires Objective-C or Swift. The project therefore faced the prospect of building, testing and deploying two separate versions of a networked mobile application within a timeframe of two academic terms (14 weeks).

To meet this challenge, the possibility of using a cross-platform development framework was investigated. This would not only save development time, but also provide insight into efficient ways to develop apps in the future. As mobile development was an unfamiliar topic, a significant amount of time was spent researching the available frameworks to determine the best fit for the project.

2. Background

Mobile developers, or developers in general, are forced to become accustomed to multiple programming languages and frameworks in hopes of supporting multiple platforms. To create a Percepto application, it was important to learn how each platform deals with the vibration API on smartphones. For example, many Android browsers, such as Samsung Internet or Firefox, allow access to Android's vibration API, whereas iOS Safari does not support it.

The app required the ability to transmit vibration cues to multiple users from a single controlling device. This functionality requires a server to connect users, and a way to collect and manage user connection data. It was decided to utilize a database to facilitate connection logic rather than a real-time peer-to-peer approach like WebSockets because it seemed like a safer and more efficient approach that would reduce network load.

There were multiple strategies that could be used to implement Percepto as both the web application and the mobile application. To keep variability minimal, Google Firebase's real-time database and Firestore systems were selected. The real-time database was used to support the web application, while Firestore was used for the mobile applications. Google's Flutter development kit was chosen to serve as a bridge between Android and iOS devices.

Dart, Flutter's programming language, appeared straightforward to learn. However, it soon became apparent that it contains many deprecated or improperly documented functions, which made development rather tedious. Despite this, Flutter proved to be a reasonable solution, as it has a lot of support for Firebase, great packages for widget and UI modification, and a short learning curve.

Percepto would also prove to be a means for learning how to deploy a mobile application on both the Google Play and Apple App stores. These virtual markets require a developer account and license, and involve significant documentation and review to get developer status approval. Percepto could have been released as a downloadable APK or IPA file for Android and iOS respectively, but an app store release is much more convenient for users.

3. Design

To save time, the final app's design was kept similar to the mock UI design previously developed for the Percepto IQP, adding functionality as needed to produce the final deliverables. An HCI-friendly approach was chosen to make the app as responsive as possible.

Figures 2 & 3. The dashboard screen that greets users, and the host's view of the lobby screen which shows a list of connected users. Source: Original UI mockups.

3.1. Revisiting the IQP mock UI

The mock UI that was created for the IQP was a minimal design intended for both mobile and desktop devices. The mock UI consisted of 4 screens per platform: the main dashboard screen, for allowing users to host or join a lobby (Figure 2); the join screen, for allowing users to join a lobby with a code; and the lobby screen, for showing hosts the connected clients (Figure 3) and for showing clients the lobby to which they are connected. The mock UI sported a maroon background with a light brown accent. The intent was to make the app simple and intuitive to use for both the host and connected users.

The mockup images created for the original UI design are presented in Appendix D.

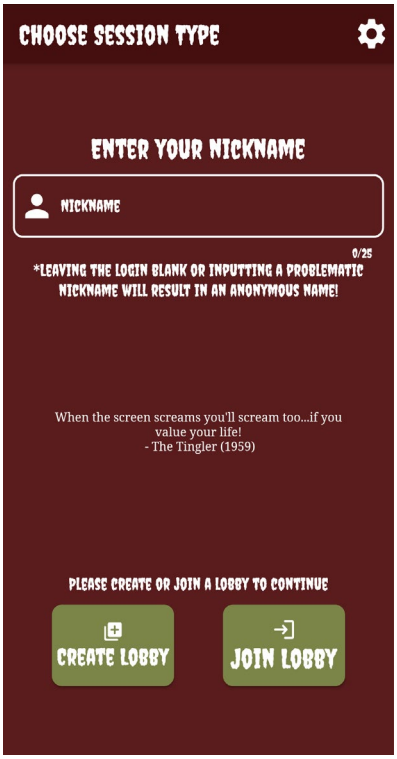


Figure 4. Final implementation of the main dashboard screen. This screen is used to host or join a lobby, collect user nicknames, and reach the setup page. Users are greeted with quotes related to William Castle (director of *The Tingler*) and his films. Source: App screen capture.

3.2. Final UI design

Figure 4 shows the dashboard screen that greets users. Changes made from the original UI include the use of the Creepster font, the addition of a nickname feature, and the settings button. We chose Creepster because it evokes the campy horror of *The Tingler* (1959). The settings button takes users to a screen so they can check their device settings and enable all the necessary permissions to use Percepto.

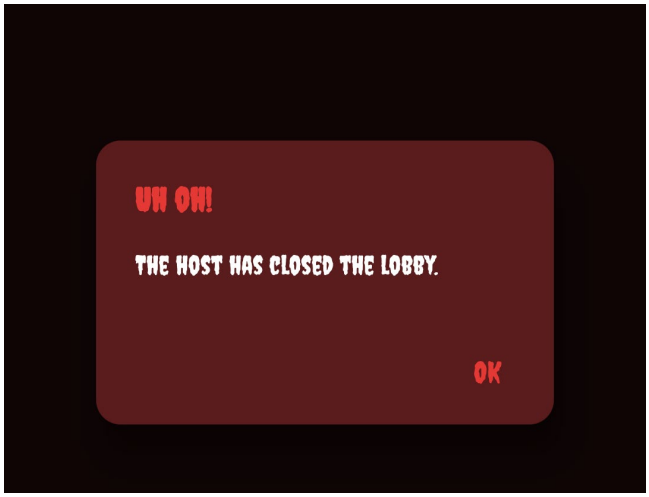


Figure 5. An error modal which appears throughout the app to inform users of what has occurred. Source: App screen capture.

The main dashboard screen has a significant amount of error handling to prevent users from causing unwanted actions from happening. The nickname feature has a 25-character limit (which is denoted by the subscript under the input box). Upon entering an invalid name by either leaving the name blank or using illegal characters, a dialog modal appears which asks the user if they would like a randomly generated name. Modals were also employed displaying status reports (such as lobby closure, shown in Figure 5), preventing unwanted presses, and conveniently handling data upon user input.

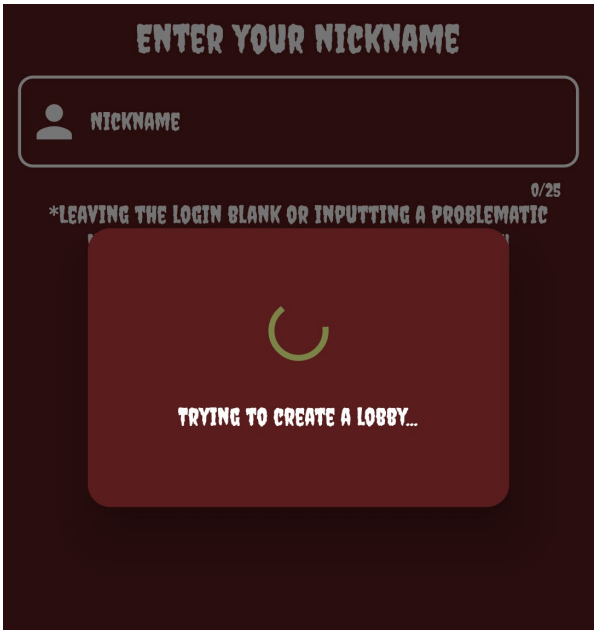
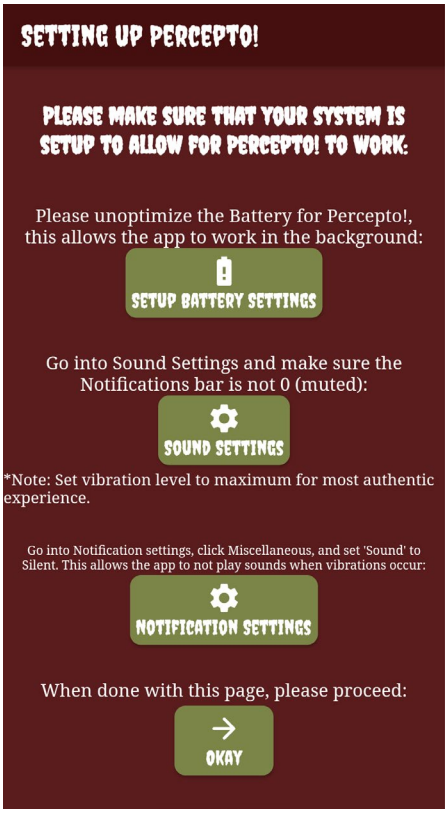


Figure 6. The loading modal that appears when creating a lobby. We also made use of modals like this for other processes that involve waiting, such as sending a vibration signal from host to clients. Source: App screen capture.

During testing, it was noticed that if a user clicked the button for creating or joining a lobby, the lack of a prompt caused some users to repeatedly click buttons due to impatience. As a result, loading modals were implemented which appear when a button is pressed, while also disabling all buttons with a temporary invisible overlay (Figure 6). The finished application maintained the visual style of the mock UI while improving on it with an HCI-friendly approach.



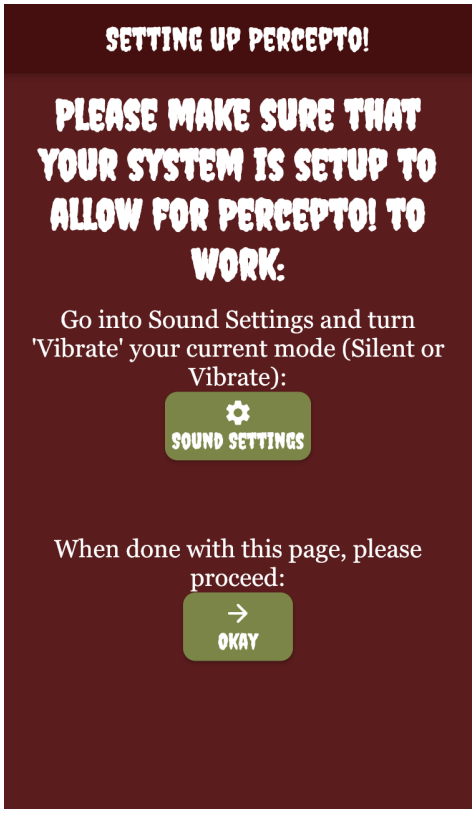


Figure 7 & 8. The Android (left) and iOS (right) settings pages. Android has a lot more settings to alter because of user customizability; some of their preferred settings may impede with app performance. Due to the lack of device privileges, we require users to check device settings manually.
Source: App screen capture.

A new page added which did not exist in the mock UI was the settings page. It displays the steps to setting up Percepto on either an Android or iOS device. Each device is shown their own specific instructions to help the app function on their mobile OS (Figures 7 & 8). An attempt was made to keep the manual editing of settings to a minimum. However, due to the nature of app idling, some user-initiated changes were required.

The settings screen is intended to be a one-time setup phase (given the user does not revert changes after app use), and as such, users are only brought to the screen on first launch or if they manually press the settings button.

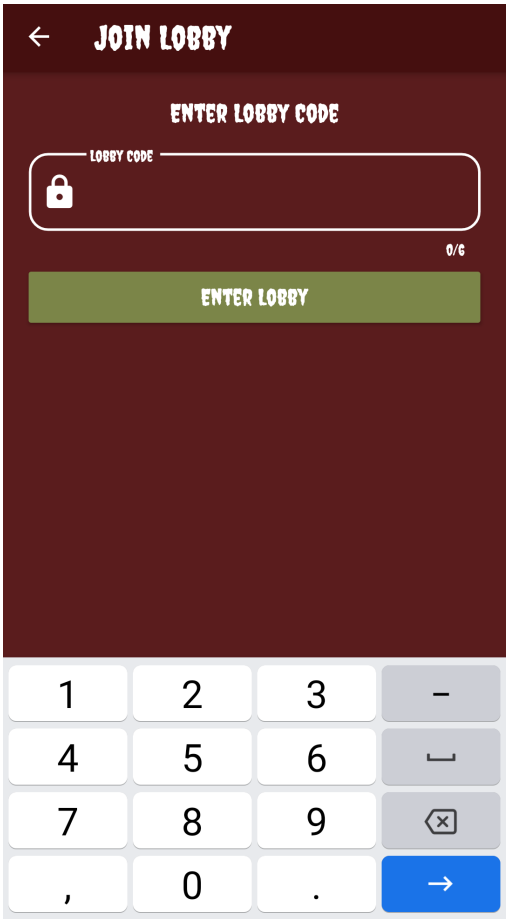


Figure 9. The join screen which users access prior to joining a hosted session. Any user may join a session that is currently in use. Source: App screen capture.

The join screen utilizes similar concepts to the nickname text input from the dashboard screen (Figure 9). Clients are prompted to input the code associated with the lobby they are trying to join. For simplicity and readability, the lobby code was changed from letters to numerical inputs only. Lobby IDs consist of 6 digits, which allows for a million concurrent lobbies. It is unlikely that anything close to a million users will attempt to screen *The Tingler* simultaneously. It can therefore be safely assumed that randomly generating a 6-digit code will produce an ID that is not currently in use. If a code is in already in use, the app will generate another. This could potentially continue till success or a third failure. When creating a lobby, hosts are given three attempts prior to receiving an error modal to “try again.”

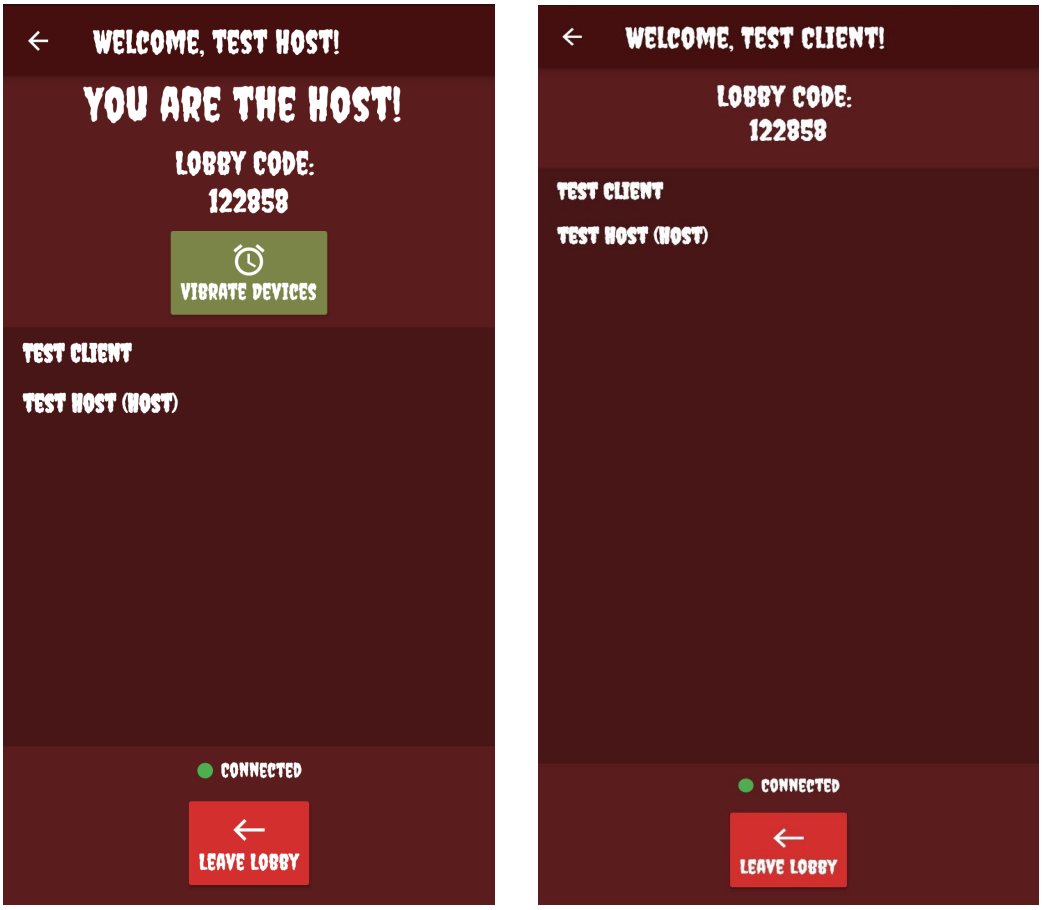


Figure 10 & 11. Hosts (left) and clients (right) have similar lobby screens, which show only what is necessary for them to see. Source: App screen capture.

The last page created for Percepto is the lobby screen, which is the focus of the app. Similar to the mock UI, hosts and clients are shown differing views, based on their permissions (Figures 10 & 11). Hosts are greeted with a message acknowledging their status, and provided with a button to send vibrations cues to connected clients. Clients have a simpler view which hides these visuals.

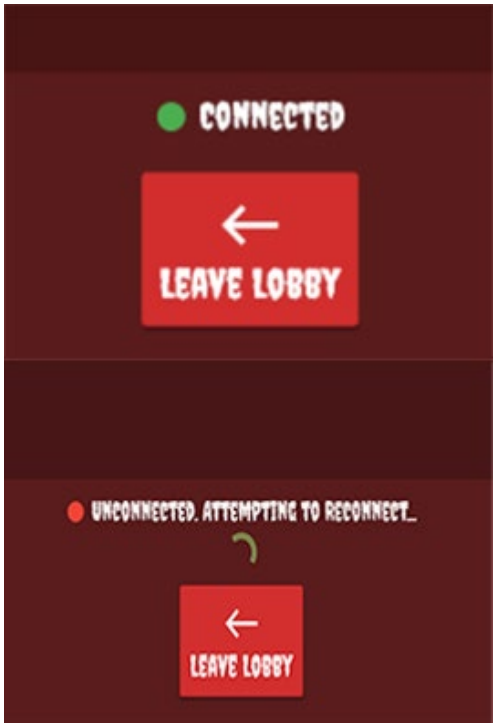
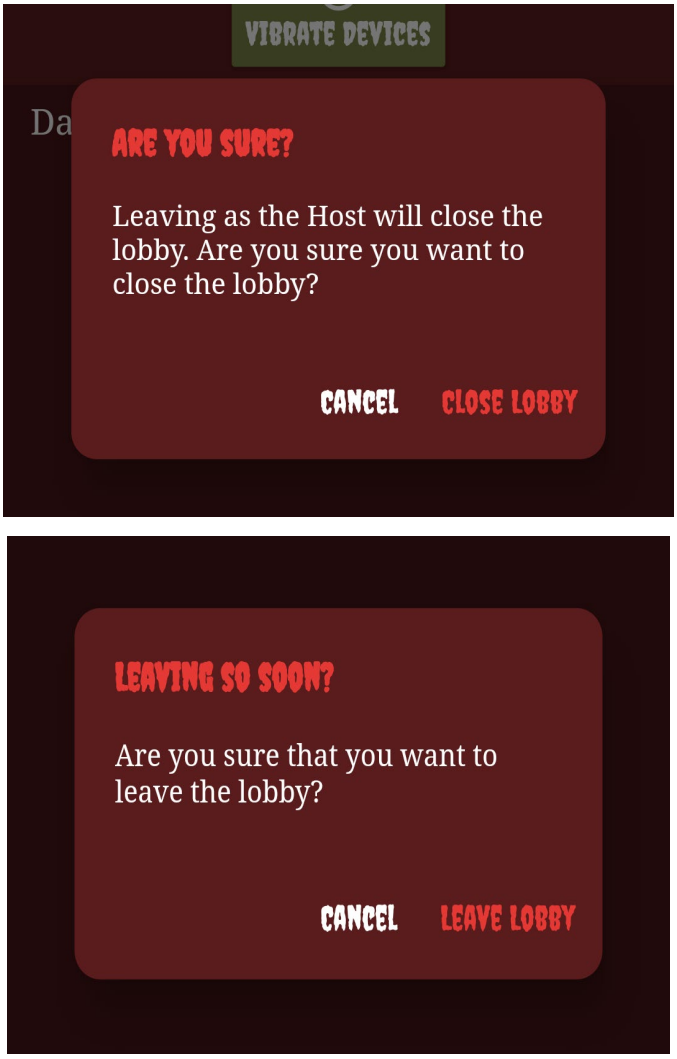


Figure 12. The nominal connection status displays against the failed connection status. We wanted to make sure users knew that their device was connected adequately. Source: App screen capture.

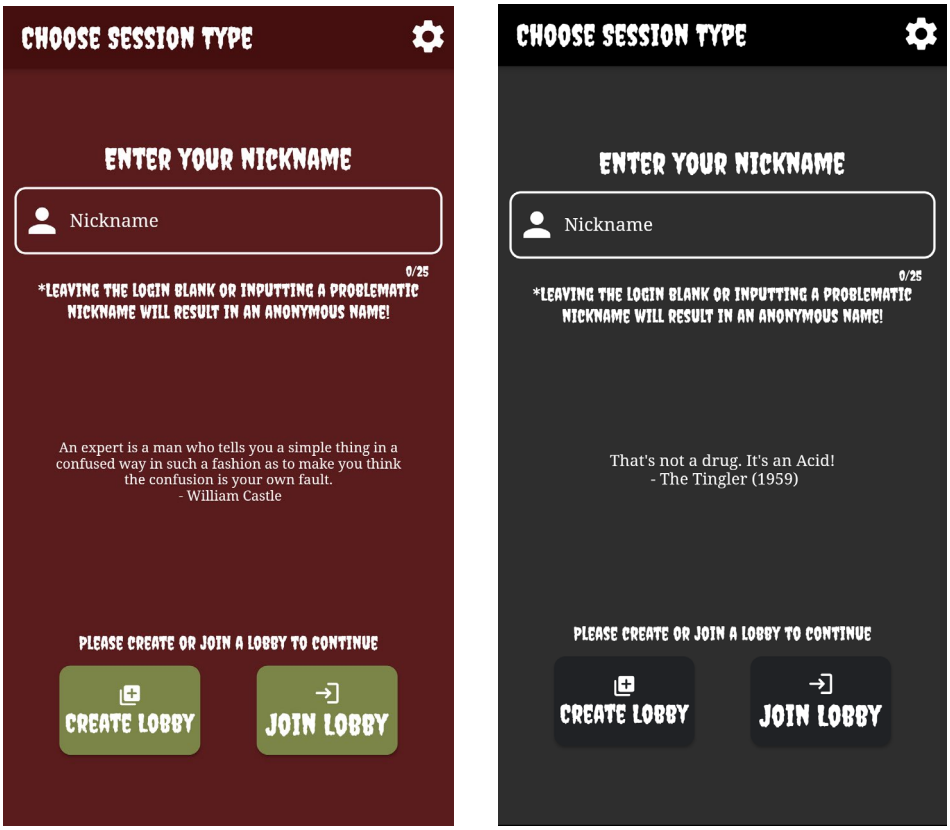
The important part of the design aspect of this screen was showing responsive modals and text on certain conditions. Users can see their connection status to Google servers (the database). When this connection fails, users see a status change along with a spinner to indicate that the connection is trying to be reestablished (Figure 12).



Figures 13 & 14. The confirmation modal (top), added to prevent human error from closing or exiting lobbies prematurely. The error modal (bottom) allows clients to take notice that the host has closed the session. Source: App screen capture.

Users can exit the app at any time, but in case of accidental presses, a confirmation modal was added to prevent hosts from accidentally closing lobbies, while also saving clients from having to rejoin because of accidental screen touches (Figure 13). When a host leaves/closes their lobby, clients are informed of this change with another error modal (Figure 14). This functionality produces a lobby experience that provides users with clear feedback about what is happening within a session.

Percepto is intended to simulate the haptic gimmick from *The Tingler*. It is intended to support group screenings in a cinema. “Dark mode” is a popular app feature, so it seemed appropriate to provide an adaptive theme suitable for a dark viewing environment.



Figures 15 & 16. The dashboard screen side by side in light mode (left) and dark mode (right). We tried to use colors that weren’t too dark so buttons could be clearly distinguished. Source: App screen capture.

The Percepto app can switch between light and dark mode based on the device’s setting. This change can even be done in-app and cause the screen to instantly rebuild itself with a new color palette (Figures 15 & 16). Though not an indispensable addition, it is hoped that this feature will keep theaters darker when the app is in use. Other dark mode screens are shown in Appendix E.

3.2.1 Branding graphics

The Percepto mobile application's icon is made from the Tingler prop which appeared in the original film. An attempt was made to remain faithful to the spirit of William Castle's horror theme in *The Tingler* (1959) by showing an iconic silhouette of its mysterious monster "crawling" over the screen. The icon itself is a darker shade of the maroon color that was utilized as the primary color of our app (Figure 17).

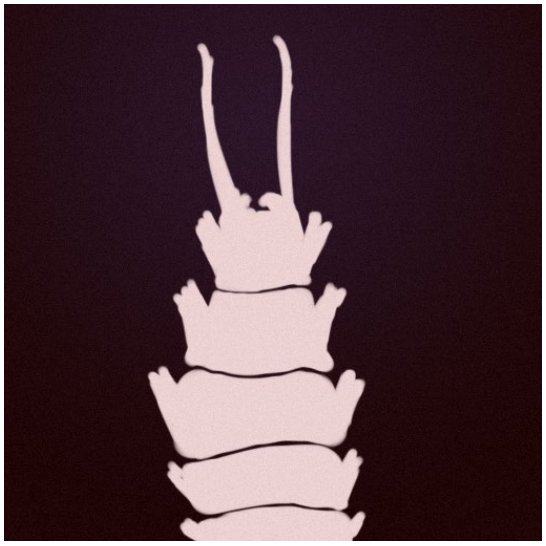


Figure 17. Percepto's app icon which features the Tingler. The icon is intended to reference the scene where Percepto was initially used. Source: Original art.

For deployment on the app stores, a feature graphic or banner is required to serve as a quick preview suggesting the purpose of the app. Without one, an application page on the store might look incomplete or unprofessional. The feature graphic created for the Percepto app is shown in Figure 18 below.

Figure 18. Percepto's feature graphic for the Google Play Store and Apple App Store. Source: Original composite art. The Creepster font is available from Google Fonts, and the royalty-free screaming man graphic was sourced from Shutterstock.

4. Development

The process of developing the Percepto consisted not only of learning the new framework, Flutter, but to also understand the inner workings and limitations of the Android and iPhone operating systems. Relevant multi-platform packages were used whenever possible to reduce the amount of platform-specific code. Firebase was an important tool for managing the backend information such as storing data, handling data, and sending push notifications. There were a lot of caveats with working with specific mobile operating systems as well as working with documentation of specific packages that were major crutches to the project. Routinely analyzing these limitations led to a better understanding of mobile development.

Google's Dart language is made to be as easy to learn as possible, with a syntax similar to C# and Java. The widget and visual designs were straightforward to understand, as everything is built around inheritance. The difficult part was trying to read documentation that explained what each function did or how to deal with states of an application. The Google team tries hard to teach people about new widgets over time; however, Google's documentation

contains a significant amount of outdated information. This inadequacy proved a significant time waster for the development of the Dart code, Firebase handling, and even for some Xcode issues.

Google's Android Studio was used to edit Dart code as well as handle Android changes (Figure 19). Version control of the source code was managed with GitHub.

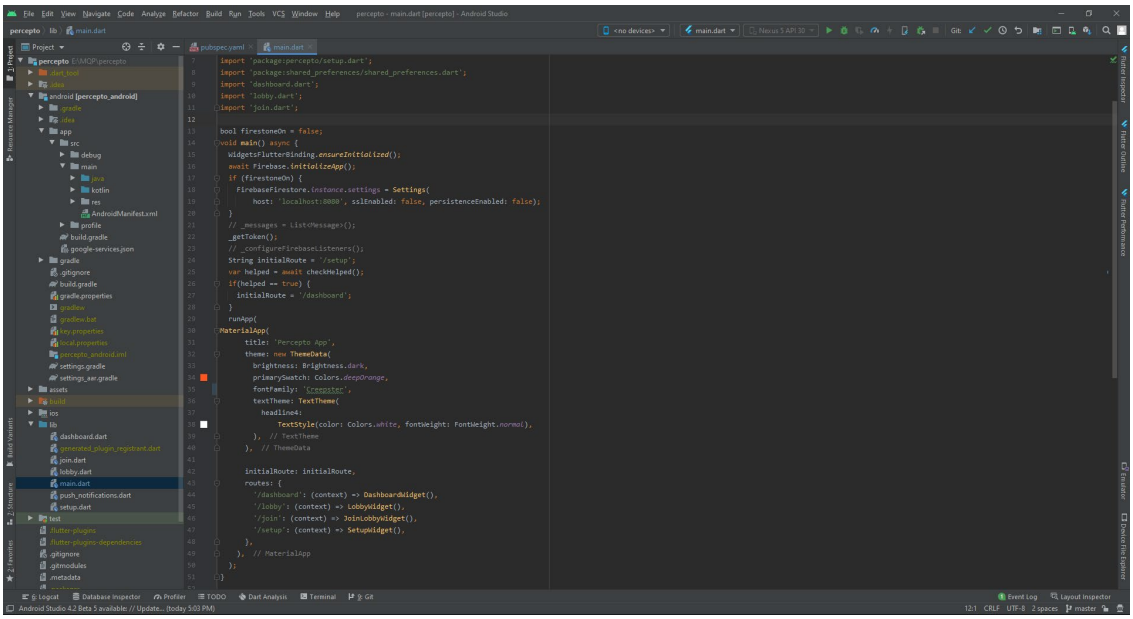


Figure 19. The Android Studio IDE. This is where we made most changes to Percepto. Source: Android Studio screenshot.

The program initially was created with barebone connection to Firebase's Firestore, which also suffers from outdated documentation. Hosts can create a lobby where a 6-digit password or lobby code is generated. Codes are number-only because letters were difficult to type in and to distinguish from each other. It was decided to use 6-digit passwords, allowing up to 1 million concurrent lobbies. The lobby code would double as the password for other users to join. If a lobby exists within the database (See Figure 20), the user's information will be written in the database, adding them to the lobby.

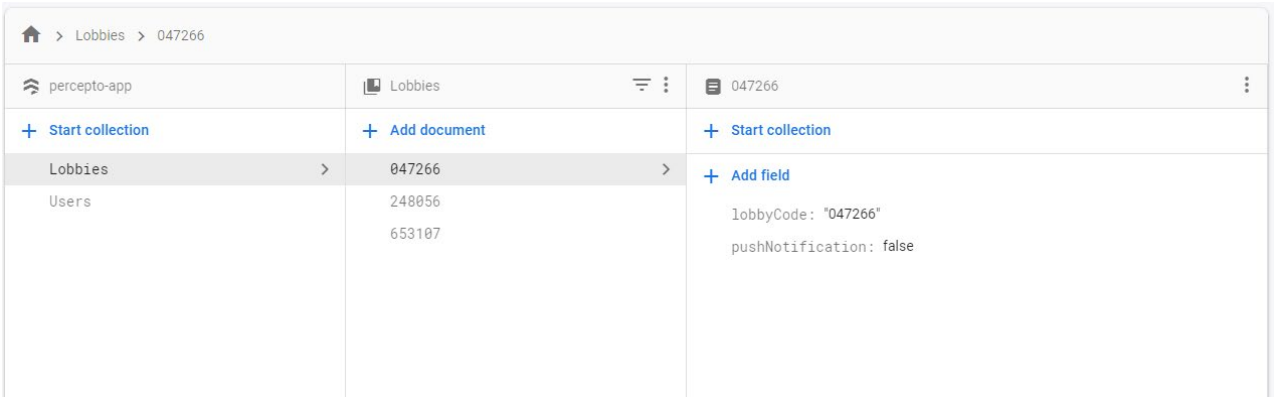


Figure 20. A screenshot of Percepto's Firestore database showcasing the "Lobbies" collection for open lobbies. Source: Firestore screen capture.

Most of the time after this was spent with error handling and dealing with the vibration API on different operating systems.

Smaller quality of life changes were added to the app to make the application seem more functional from a human-computer interaction standpoint. When creating a new lobby and sending a vibration request to other users, a dialog modal appears which shows the user that a process is being done. Additions like these were helpful because the application then responds to users with more information as well as preventing users from doing unwanted actions. Nickname and lobby code inputs have a character limit which prevents users from inputting too many characters. These inputs also check for illegal entries such as invalid characters or empty inputs. The most tedious HCI implementation was the confirm dialog for leaving or closing a lobby which was done with a confirmation modal from a local click or an error modal from an extra Firestore parameter. The tediousness came from having to delete Firestore data while also giving devices enough time to receive the lobby's closure signal.

The focal point of the project was to implement William Castle's gimmick, *Percepto*, into a mobile form. As such we needed to figure out how to get the vibration API not just working on the devices, but to also be triggered from the input of the host. In a peer-to-peer connection this would be simply the sending of a signal (ignoring all the security and packet handling that WebSockets entail); however, with a connection from a database to an idle device, this is a lot

harder than it seems. Users are “connected” by sharing a lobby code field in Firestore. As such, our way of implementing synchronized vibrations was to add a vibration Boolean field and have mobile devices listen for updates to said field. Hosts upon clicking the vibration button would get and update all docs (or users) to change the vibration Boolean to true. The mobile devices would check if the user’s (seen by their unique identification code) vibration Boolean was true and would utilize the vibration API to send a custom vibration before resetting their vibration Boolean back to false. If a device were then listening to Firestore changes while sleeping in a user’s pocket throughout the duration of a movie, this wouldn’t be a problem.

4.1. Android OS limitations

Percepto was initially developed and tested on an Android phone, meaning that most solutions were created and analyzed for the Android version first. Once Firestore was set up, the Android version was up and running after giving the application specific permissions to the Android device such as Internet and Vibration permissions. This had to be added to a data file called the AndroidManifest.xml. Essentially, this defined the location of assets as well as what resources the app would require.

Many problems were encountered when testing the vibration API after the correct permissions were applied. To test the experience of sitting through a complete screening of *The Tingler,* the app was run in the foreground, in the background, and in sleep mode (suspended). It became quickly apparent that the vibration API would not be triggered when the app was running in the background, or if the device was asleep for too long. Eventually, documentation for Android’s “Doze” or “App Standby” mode was located. Android created a system where applications that were not in focus while remaining in memory would go into sleep to conserve battery. Devices that were running Percepto and were listening for changes to Firestore were unable to listen while asleep.

To correct this issue, a high-priority push notification from Firebase’s Cloud Messaging system is used to jolt the application awake, and the change of status is recorded in Firestore.

This correction permitted conservation of battery life along with a means of the awakening of the app from doze mode. However, because push notifications from Firebase's Cloud Messaging service are made for throughput rather than latency, it was decided to also implement a request for the user to manually disable battery optimization, which bypasses doze mode. This assures that users will receive a vibration at the correct time from either staying awake with battery optimization disabled or reawakened from a high priority push notification.

When configuring the app, the Setup screen also instructs users to unmute (and preferably maximize) their notification volume, and unlink Percepto's notification sound from the default ringer.

4.2. Apple iOS limitations

Apple works hard to maintain exclusivity on their platform. Because a little under half of the population uses an iPhone, there was no choice but to play along with Apple's rulings. To create an iOS app, it is necessary to use Xcode and a physical iOS device to run tests that cannot be run on a simulator.

Xcode is a programming environment available only on Apple's Macintosh PCs. Its use was required to bridge the Flutter application onto TestFlight and the AppStore. A mid-2012 MacBook Pro was borrowed for development, which served adequately after adding more RAM.

Xcode suffers from poor documentation and many functional errors. Most of the problems related to dealing with the Mac OS dependency manager, CocoaPods, and reintegrating auto-generated files to fit the current Mac's environment. Extra certificates and authentication keys had to be generated from the developer's Mac along with the developer website. This was necessary to get through Apple's APNS service, which requires up-to-date authentication. The APNS authentication key was then uploaded to Firebase to allow for messaging iOS devices as well as requesting permission to send notifications to the user.

Like Android's doze mode, iPhones are constantly watching which apps are currently in use while putting other apps to sleep. Apple, however, is much more aggressive with this process, even going so far as to terminate processes that try to run after being suspended. To make matters worse, if an app is suspended for too long, iPhones give the app less and less priority till eventually it cannot be re-awakened from headless functions. The only exceptions to this rule are VoIP connections or timers with alarm parameters.

Additionally, when defining permissions to be used on the device, all potential uses for the permission, even the ones that you are not using in the app, must be given a disclaimer. This is done to try and give users more information on what is happening, as well as preventing developers from misusing Apple features.

Because Percepto makes use of a system where the host gives an input for users to listen for, and devices are put into suspension for not being in use, a timer is not feasible. VoIP might have been a solution, but misusing features, such as a Voice over IP connection, could result in Percepto being pulled from the AppStore. When in this state, the app might be considered terminated, because not even high-priority push notifications can force an app to awaken.

Fortunately, a way was found to enable vibrations during the suspended state. Push notifications were still able to be received, and typically those can make iPhones play an alert sound and vibrate. By enabling sound notifications, we're able to also enable vibrations from a push notification. We added a custom alert sound in a CAF audio file with 1 second of no sound. That way the notification would play its silent alert "sound" along with the vibration that we originally wanted. Unfortunately, this workaround doesn't support custom vibration patterns, but it gets the job done.

The iOS vibration API wasn't able to be fully tested during development, because CoreHaptics, a package that gives iPhones access to customizable vibrations (and other haptics), wasn't added until iPhone 8. Users will still be able to receive a vibration on their iOS devices despite these many caveats.

4.3. Frontend technologies

4.3.1. Flutter

Flutter is an open-source software development toolkit offered by Google for creating UIs for mobile devices such as iOS and Android, and web browsers such as Safari and Chrome. It was chosen as the development environment because of its strong support for packages and widget design, as well as providing a bridge between Android and iOS devices. Flutter could also be used for Google Fuchsia, Linux, macOS, and Windows.

Although relatively new (it was first released in 2017), Flutter is stable and works quite efficiently. Widgets are managed with inheritance, simplifying the management of an application's UI. Multiple Flutter packages were used to serve our requirements for the front and backend of our application.

4.3.2. Vibration plugin

The vibration plugin is a package that gives Flutter access to the device's vibration API. The current version (1.7.3) works with iOS and Android devices as well as mobile devices on the web. The vibration plugin allows Flutter to control vibration amplitude, duration, delay, intensities, and patterns. The plugin requires the requesting of a permission for Android devices and has limited customizability for non-CoreHaptics devices such as iPhones older than the iPhone 8. This plugin was the most crucial installment to the development of Percepto.

4.3.3. App Settings plugin

The "app settings" plugin for Flutter (v. 4.0.4) allows an application to send users to specific settings pages on Android and iOS devices. This was useful for making setting up the application easier on users by sending them to the correct settings page from a single button press. Percepto could be made without it, but it improves usability.

4.3.4. Permission Handler plugin

The “permission handler” plugin (v. 5.1.0+2) allows Flutter to ask users for permission on specific utilities such as access to the camera, contacts, location, microphone, photos, etc. This was used to disable battery optimization for Percepto in the Android application; iOS devices will not see the button to request this permission, since Apple doesn’t allow it. This plugin (among others) forced us to add extra documentation to the iOS info.plist and write disclaimers for permissions we did not actually need to request. When a user agrees to the request, the process is automatically done. A check is run to determine if the battery is optimized or not whenever the user presses the battery setup button or tries to join a lobby. We don’t bother checking for the host, because the host must awaken the application in order to push the vibration button.

4.3.4. Cupertino icons

Cupertino Icons (v 1.0.0) was an essential package for the visual design of Percepto’s buttons. The Cupertino Icons are icons that were based on the visuals of Apple’s icons brought into Flutter. This allowed the Android and iOS versions of Percepto to have a good HCI implementation. The key was to allow for immediate recognition for the buttons and inputs that we have.

4.3.5. Wakelock

Of all our plugins, Wakelock (v, 0.4.0) was one of the least valuable for Percepto. Originally, it was hoped that Wakelock would be able to keep the app awake while in the background, but it only prevents the device’s screen from falling asleep or dimming. Nevertheless, we decided to keep it on the lobby screen so the host can, if they choose, leave their device screen on throughout a lobby session.

4.3.6. Auto Size Text plugin

The “AutoSizeText” plugin (2019-08-13) was added to make text dynamically resize to fit the device in use. It adaptively forces text to occupy a given number of lines while also defining the maximum and minimum text font size. This plugin was not a critical addition to the project, but it made designing Percepto a lot easier.

4.4. Progressive Web app

One option was to develop a Progressive Web App (PWA). A PWA is similar to a browser web app but with the option to install on the desktop or mobile phone. Most PWAs leverage the offline capability of a native app and can be built easily from a JavaScript framework such as ReactJS.

There have been several recent developments for PWAs. Large companies such as Twitter and Google have made their own PWA frameworks. YouTube, Spotify, and Twitter are some examples for their apps published on their site and can be easily installed from the browser. The benefits of PWAs are that they bypass publishing platforms such as Google Play Store and Apple’s App Store, and offer performance comparable to a native app. However, there are still some native features that PWAs cannot yet access, such as the Vibration API on iOS. Over time, there will be support for iOS since Android already has more features accessible for PWAs, such as Push Notifications.

4.4.1. Push API

The Web Push API is supported by many Android phones, Windows OS, and MacOS, and many other devices, except for mobile apple devices (iOS). Apple’s security measures prohibit third-party applications from the browser from using the vibration feature (reserved mainly for haptic feedback).

4.4.2. Vibration API

This API seemed like the most straightforward solution for the Percepto app. However, this module is not supported by the most recent iOS firmware (iOS 14), and can only work while the device is active (not screen locked). To use this function, the app would need to remain active at all times to trigger the vibration, and continuously listen for trigger events (either by keep-alive connections or WebSockets) from the database.

4.5. Backend technology

4.5.1. Firebase

Google's Firebase Backend-as-a-Service (BaaS) is a platform with many services that aid in the development of mobile and web applications alike. The number of services Firebase offers makes it quite versatile. Firebase has three separate database options, server hosting, cloud functions, and even a machine learning service.

Firebase has two entry-level payment plans, Spark and Blaze. Spark is an entirely free plan which allows for the use of their services minus the Cloud division. The Blaze plan is a "pay as you go" service which is free till after 1GB of data is regularly used along with the use of the Google Cloud service. It was decided to use the Blaze plan to implement various Cloud functions for automated maintenance of Percepto's databases. As such, one of the downsides to Percepto would be that the app has a 1GB database limit, which is unlikely to pose a problem unless a lot of users try to use the app at once, or employ it maliciously.

Firebase also allows developers to get statistics of Firebase usage within their app which is useful for tracking different pieces of information (Figure 21).

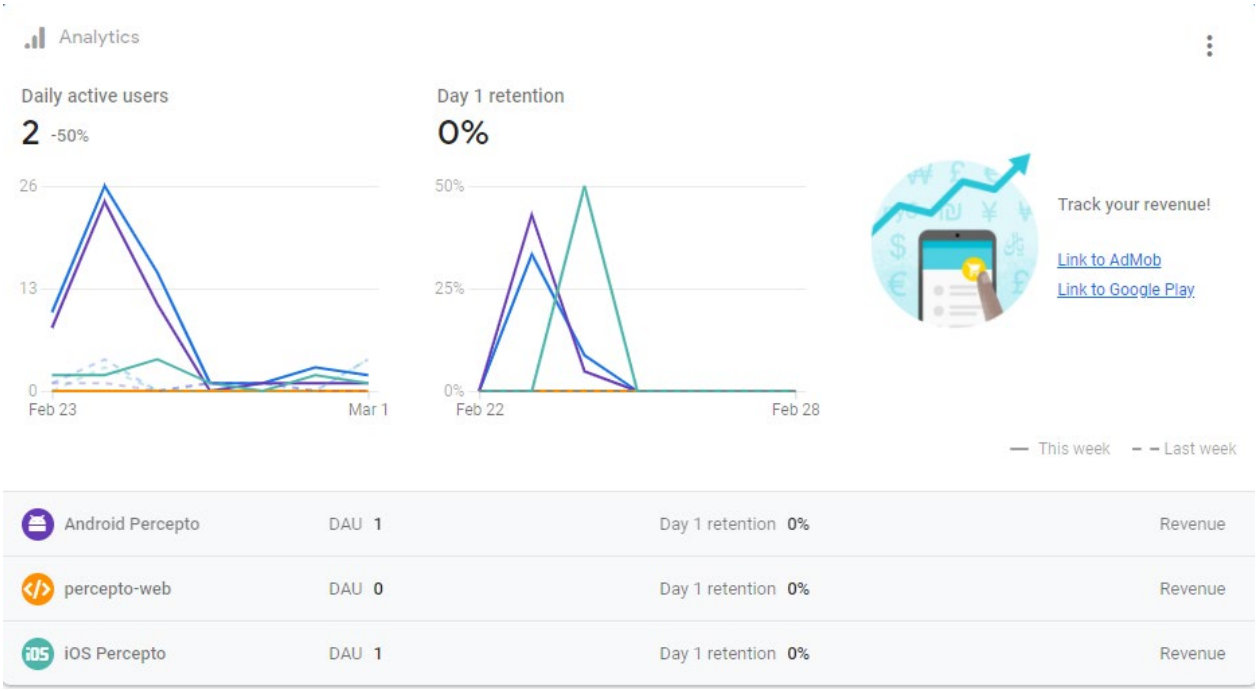


Figure 21. The Analytics page of Firebase which gives a quick rundown of Firebase service usage across implemented apps. Source: Firebase screen capture.

4.5.2. Realtime database

This NoSQL database specializes in quick changes, and is metered based on the amount of data transferred. Under the current quota, the free plan allows up to 10 GB/month of data transfers as reads/writes. The entirety of one Realtime database data can be represented in one JSON tree.

4.5.3. Firestore

Google's Firestore is a NoSQL database similar to the Realtime Database. However, Firestore is more optimized for queries. In comparison to the Realtime Database, Firestore has a higher uptime for stability and is metered based on the number of reads and writes. Under the free quota, Firebase allows up to 20,000 reads and 50,000 writes per day. Instead of a JSON tree representation, Firestore is represented by a series of Documents and Collections, where a Collection contains multiple Documents, and the Documents contain data. With the help of Cloud Functions, the Firestore database automatically clears data that is over 24 hours old. This is done by having a Cloud Function check for old data when a new user is written to the database. Users that are written to the database and forcibly close the app are never manually cleared. This required error handling to prevent lobbies from turning into "ghost rooms" or hosting "ghost users."

For Percepto, it was important to prevent Firestore's 20,000 read and 50,000 write limit from being reached. Though unlikely, code was added to shut down the database if the limit is reached until the daily limit is cleared.

4.5.4. Cloud Messaging

Firebase's Cloud Messaging is a cross-platform way to send messages to users via push notification, emails, and other data payloads. It can be used to allow devices to sync with recent changes of some form as well as give custom updates mandated by the developer. Cloud Messaging was utilized to send automated push notifications to users. This was done by creating a Cloud Function that fires a predefined message to all users within the same lobby as the host whenever the host presses the vibrate button.

4.5.5. Universally Unique Identifier plugin

The UUID generator and parser plugin (v. 2.2.2) was added to the Flutter project so all users could be given a unique identifier. This was especially useful for making sure users could confirm changes were happening to their specific data on Firestore; allowing for multiple users with the same name as well as an anonymous name generator. Therefore, while the users can set their own nicknames, they are identified in the database by their unique id.

4.5.6. Shared Preferences plugin

The shared preferences plugin (v. 0.5.12+4) was a very useful plugin for both the Android and iOS sides of Percepto. Both operating systems use different means for implementing persistent storage (local data), with Android using SharedPreferences and iOS using NSUserDefaults. The shared preferences plugin handles both persistent storages automatically, depending on which platform the Percepto app is installed on. This plugin was used to store user data onto the device so it could be sent to Firestore and for listening to specific changes for parameters on Firestore that matched local data.

4.5.7. Connectivity plugin

The connectivity plugin (v. 2.0.2) was needed to add a check for whether the user was connected to the Firestore database or not. An attempt to connect to Google's servers is made over both Wi-Fi and mobile data networks. If Google's servers are down, it is assumed that Firebase is also down. This plugin was chosen for its ease of use.

5. App publication

A decision had to be made whether to creating a website to host the application, or publish it on the Google and Apple app stores. A website for allowing users to access and download the APK and IPA files for Android and iOS respectively would be an easy route, but it was decided that the app would be more accessible if posted on the stores. A basic requirement for using these services is obtaining the appropriate developer licenses.

5.1. Google Play store

Google Developer requires a $25 one-time fee to receive a license, which allows developers to not only customize their posting on the app store, but also to submit the mobile application. Subjectively, the Google Play Console is quite bloated as a service. The console is divided into many sections and subsections, each with similar purposes and/or restrictions. Many forms of testing are supported, such as pre-registration, internal testing, closed testing, and open testing, all of which could be better defined as a parameter for an application rather than a separate subsection.

Appbundle builds are used to upload the app for testing or publication. However, these builds cannot be deleted or removed from the console. This just led to the test feed becoming overly cluttered with builds. Other than issues with the appearance of the console, builds that are uploaded onto the store are given a 1-2 day review period. The Google Play Console, after getting used to the scheme, was a straightforward process that required minimal extra documentation on the application. The only real requirements were the $25 license, an app icon and feature graphic, country availability, and the application description.

5.2. Apple App store

The Apple App Store was a seemingly more straightforward process, but required a lot more supplementary documentation prior to app uploading. The first step, like the Google Play Store, requires an Apple developer license. The license, instead of being a one-time fee, is a $99 yearly subscription tied to the developer's Apple ID. Apple divides their developer website into two sections. The first is the "Certificates, Identifiers, & Profiles" page, which handles permissions to all developer applications. The second is the "App Store Connect" page, which provides control over individual applications and their builds in relation to the App Store and TestFlight. The certifications page is used to certify devices to connect to the developer page as well as specify the Apple devices that are being used. This procedure verifies that official Apple products are in use prior to uploading an app. In the certifications page, an APNS authentication key was created to allow Firebase to communicate with iOS devices over APNS.

App Store Connect was where most of the requirements for the publishing of Percepto occurred. It allows testing to be defined as a parameter within TestFlight. TestFlight requires a description for what is being tested, along with an approval process to confirm the app exists and is launchable on iOS devices. A bundle identifier was established to allow Xcode to communicate a build to a specific Apple address.

Uploading to the Apple App Store was a more tedious process, as it required a lot of documentation. Screenshots for all the device display sizes, app descriptions, promotional texts, a targeted audience age, a customer support document, and a privacy policy were all the public-facing documentation required for uploading. For the review process, which involves assigning an Apple employee to approve the app, it was necessary to provide a note section along with video showing the specific iOS device registered in the developer account being used. With all of this done, Percepto was ready to be published for Apple.

6. Evaluation

The development process of Percepto in Flutter allowed monitoring of visual changes to the app's design, together with fast reloading of the app in simulators. This enabled rapid evaluation of updates prior to further work on bug fixes and improvements, and was used to support multiple rounds of user testing of the app.

6.1. AlphaFest

In the early stages of development, a significant amount of time was spent on Flutter UI and development, packages, and Firebase. About two months into the process, the vibration plugin was functioning through the Firestore database, allowing for synchronized remote vibrations to be sent. Fortunately, WPI's Interactive Media & Game Development program was sponsoring an annual testing event, AlphaFest, just a few weeks after the synchronous vibration was implemented. After the outbreak of COVID-19 in 2020, AlphaFest became entirely virtual, meaning that many different locations would be available for testing the synchronous operation of the app.

WPI's Institutional Review Board (IRB) requires certification by the Collaborative Institutional Training Initiative (CITI) before running tests with human subjects, together with documents explaining the research methods, consent agreement, and survey instruments to be used. All surveys and research were conducted anonymously. Documents related to the IRB application are provided in Appendix A and B.

Before AlphaFest, the Apple developer license and Macintosh PC required to run Xcode had not yet been secured. Therefore, the app was distributed as an APK file for Android users only. Because of this limitation, and a non-physical test environment, only three testers were recruited during the event.

A post-test survey was conducted to rate the performance of the test. The results were surprising. The survey (included in Appendix C) consisted of two Likert scale questions, a

multiple-choice question, and an optional open response question. All three respondents answered similarly, rating the app 4/5 for functionality and 1/5 for navigation difficulty (1 = easy). All testers reported received the vibration notification successfully.

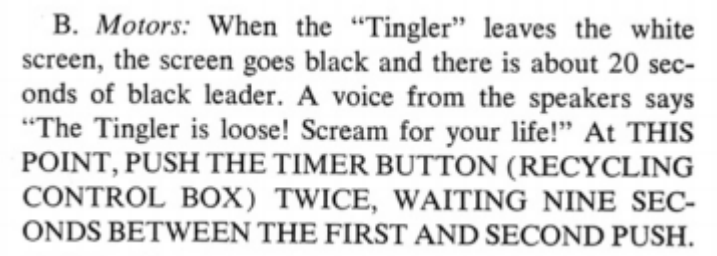
The open responses had comments and criticisms asking for differing patterns, a “voice chat system,” easier password inputs, a more aesthetically pleasing appearance, and the combining of screens for a more streamlined experience. It was decided to work towards implementing the suggestions that best complemented the Percepto theme of the app.

6.2. Subsequent user testing

After almost six months of development, Percepto was in its final stages. More testing was needed to evaluate the many additions made since AlphaFest, which included automated processing, error handling and human-computer interaction design, together with new functionality such as push notifications and custom vibration support.

Builds for both Android and iPhone devices were prepared to support as many testers as possible. Updated documentation was submitted for IRB approval.

Invitations were sent to subsections of the WPI student body. Three testers (one iOS, two Android) agreed to participate immediately, so preliminary tests were run prior to the next visual redesign (and other functional upgrades) build. The iOS user was using an iPhone 7, a model which predates support of CoreHaptics; this improved the breadth of testing. We sent each tester the Informed Consent Agreement form and a link to the pre-test survey so that we would know which devices were being tested.



B. *Motors:* When the "Tingler" leaves the white screen, the screen goes black and there is about 20 seconds of black leader. A voice from the speakers says "The Tingler is loose! Scream for your life!" At THIS POINT, PUSH THE TIMER BUTTON (RECYCLING CONTROL BOX) TWICE, WAITING NINE SECONDS BETWEEN THE FIRST AND SECOND PUSH.

Figure 22. Instructions provided by Columbia Pictures explaining how exhibitors should activate the electric buzzers installed under seats in theaters screening *The Tingler.* We followed these instructions during the second and third test sessions of Percepto. Source: URL.

The test was conducted similar to the way an actual screening of *The Tingler* would be handled. The movie's climactic scene was shown via Zoom screen-sharing. At the appropriate moment, the test host pressed the vibration button as instructed in the original Percepto Manual provided to theaters (Figure 22).

All testers were able to receive the vibration on cue, while some did not receive the push notification that corresponded with the vibration signal. This could be because in-app messaging was disabled. Users who left their device on with the app open would not be able to see the push notification. Testers rated the app's navigation ability at 1.33 out of 5 (1 = easy). Comments including requests for more powerful vibrations, along with praise for the app's low battery drain on the older iOS device.

Over the week following our second test session, bugs were fixed, and the visual design of the app was upgraded with new fonts, more modals, a connection checker for the database, and themes for light and dark mode users.

For the third and final formal testing of Percepto, six test subjects were hosted. Two were using Android devices, three had newer iOS devices, and one had an old iOS device. Survey results included a rating of 2.4 out of 5 for setup difficulty. Four out of six users received their vibration on time, while two experienced delayed vibrations. This was better than expected, as the performance of the iOS app, which most testers were utilizing, is generally inferior to the Android version. A rating of 4.83 out of 5 was reported for the app's functionality, and a 4.3 out of 5 for the visual presentation.

No voluntary comments or criticism for this test session, so it is assumed that the app ran smoothly on all tested devices. Nevertheless, more visual changes to the app were made to improve readability and provide adaptive formatting.

7. Conclusion

The purpose of this MQP was to learn how to create a fully functional application through web and mobile developmental processes. This kind of project entails the learning of multiple tools, as well as the devices those tools are made to address, including Flutter and Dart to bridge Android OS and iOS, together with the use of Flutter packages and other services such as Firebase to address the needs of the app. The project needed to be worked on consistently each week to meet academic deadlines.

Much was learned about what it's like to be a mobile/web developer while working through this project. Dealing with the Google Play and Apple App stores established expectations for what to expect in creating future applications.

The final product, and the process required to reach it, far exceeded original estimates for what the project would entail. Had key roadblocks such as App Doze on Android OS and Suspension on iOS been known about in advance, a lot of time and resources would have been saved. Based on what was discovered, peer-to-peer connections through WebSockets would be considered more seriously, given that battery drain was minimal, even after hours of connection. App Doze and Suspension issues would be dealt with during the design phase rather than the end of the developmental phase.

Overall, the project was a great experience, which the recent release of Flutter 2 in early March 2021 (supporting mobile, car, and television app development) will hopefully make even more valuable.

Works cited

Api.flutter.dev. n.d. *Icons class - material library - Dart API*. Available at: https://api.flutter.dev/flutter/material/Icons-class.html [Accessed 9 January 2021].

Apple. “Apple Developer Program.” Apple Developer Program, 2021, https://developer.apple.com/programs/

Apple Developer. n.d. *TestFlight - Apple Developer*. Available at: https://developer.apple.com/testflight/ [Accessed 9 February 2021].

Flutter.dev. n.d. *Use themes to share colors and font styles*. Available at: https://flutter.dev/docs/cookbook/design/themes [Accessed 26 February 2021].

Firebase. n.d. *Add Firebase to your Flutter app*. Available at: https://firebase.google.com/docs/flutter/setup [Accessed 17 October 2020].

GitHub. 2020. *Explain that a grey screen on release is indicative of an error (and how to find the error)*. Available at: https://github.com/flutter/flutter/issues/53482 [Accessed 10 November 2021].

Hasgeek TV. (2016, November 25). Firebase Realtime Database deep dive - Soham Mondal, Triveous [Video]. YouTube. Retrieved from: https://www.youtube.com/watch?list=LL&t=1509&v=1nUSoCZlnDo&feature=youtu.be

How to Install and Setup Flutter for App Development on Windows - Part 1. (2019, May 1). [Video]. YouTube. Retrieved from: https://www.youtube.com/watch?v=Z2ugnpCQuyw

Lau, P. and Stoessel, J. (2020) *Percepto! The Birth of Haptic Cinema*. WPI Electronic Projects Collection. Available at: https://web.wpi.edu/Pubs/E-project/Available/E-project-051820-210051/ [Accessed 17 March 2021]

Seng, L., n.d. *How to Send Silent Push Notifications*. Medium. Available at: https://medium.com/swlh/how-to-send-silent-push-notifications-fef208b24863 [Accessed 18 January 2021].

Shah, D., 2020. *Flutter Tip: Show scrollbar in ListView*. Medium. Available at: https://medium.com/flutterflakes/flutter-tip-show-scrollbar-in-listview-71fc504c71e5 [Accessed 3 March 2021].

Stack Overflow. 2017. *Firestore - Checking The Connection Status Of The Module To The Server*. Available at: https://stackoverflow.com/questions/47397547/firestore-checking-the-connection-status-of-the-module-to-the-server [Accessed 5 March 2021].

Stack Overflow. 2018. *Flutter One time Intro Screen?*. Available at: https://stackoverflow.com/questions/50654195/flutter-one-time-intro-screen [Accessed 24 January 2021].

Support.google.com. n.d. *Create and set up your app - Play Console Help*. Available at: https://support.google.com/googleplay/android-developer/answer/9859152?hl=en [Accessed 10 November 2020].

Appendix A: IRB Plan of Study

Title of Research Study: Haptic Cinema

Purpose of study

To obtain user feedback in order to determine if experience goals are being achieved, locate operational bugs, and identify opportunities for design improvement.

Study protocol

Participants are directed to a Web URL where they view the Opening Briefing (below), complete the Informed Consent Agreement, and fill out a pre-test Survey. The response from the pre-test survey will allow us to contact the participants while also inviting them to specific application tests. They will follow written instructions to join an app session remotely.
After the testing is finished and the session closes, participants will be directed to another URL and asked to fill out a short survey to characterize aspects of their subjective experience and solicit suggestions for improving the experience.

Opening briefing for testers (provided online)

"Hello, and thank you for volunteering to test our project. Before we begin, I need to ask if everyone has read and signed the Informed Consent form? [Testers confirm they signed the IC form.] Thank you. When your session is complete, we will ask you to complete a brief survey about your play experience. At no point during your test session, or in the survey after, will any sort of personal and/or identifying information about you be recorded. Please follow the instructions when you are ready."

Appendix B: IRB Informed Consent Agreement

Informed Consent Agreement for Participation in a Research Study

Investigator: Brian J Moriarty

Contact Information: bmoriarty@wpi.edu, Tel. 508-831-5000

Title of Research Study: Haptic Cinema

Sponsor: WPI

Introduction: You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risk or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

Purpose of the study: The purpose of this study is to obtain feedback on an application to facilitate design improvements and find/address operational bugs. The application's main features include utilizing a mobile device's vibration API along with allowing users to connect to other devices using a lobby system. For clarification, the lobby system works similar to that of a game's chat room without being able to send messages; users instead can have their phones vibrate at the host's leisure. Some operational bugs that we expect to find will be relative to connection errors such as not being able to join a lobby or receive a sent vibration, or functionality issues with navigating the application itself. Feedback is then crucial to inform us of what works within the app and what does not.

Procedures to be followed: You will be asked to launch a mobile application on an Android or iOS device and navigate through the app to join an active lobby/room. The application is a simple lobby system that allows users to host a lobby or join a hosted lobby. The main purpose of the

app is to allow users to join a lobby along with users receiving a vibration effect from their phone (at the host's control). Prior to testing, users will be instructed to enable and unmute their notification volume on their mobile device and unoptimize their battery settings for the application (on Android). The app store for Google Play and Test Flight will allow users to download the app along with minimal instructions to assist the user with joining a test lobby. There will be one or more testing sessions for the purpose of stress testing and testing overall functionality. The only information in the application that is recorded is the inputted nickname of the user's choice which will be anonymously stored along with the device token being stored for push notifications. All nicknames and other information tied to a user will be removed from the database upon leaving or closing of a lobby. No nicknames nor other information will be analyzed in the testing of this app. After the tests, users will be asked to complete a brief, anonymous survey describing aspects of the subjective experience with the app.

Risks to study participants: There are no foreseeable risks associated with this research study. Benefits to research participants and others: You will have an opportunity to enjoy and comment on a new game under active development. Your feedback will help improve the game experience for future players.

Record keeping and confidentiality: Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury: There is no foreseeable risk of injury associated with this research study. Nevertheless, you do not give up any of your legal rights by signing this statement.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact: the project Investigator (Brian J. Moriarty, Tel. 508-831-5000, Email: bmoriarty@wpi.edu). You may also contact the IRB Manager (Ruth McKeogh, Tel. 508-831-6699, Email: irb@wpi.edu) and the Human Protection Administrator (Gabriel Johnson, Tel. 508-831-4989, Email: gjohnson@wpi.edu).

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

_____ Date: _____

Study Participant Signature

_____

Study Participant Name (Please print)

_____ Date: _____

Signature of Person who explained this study

Appendix C: Survey instruments

Study instrument for IMGD AlphaFest (2020-11-20)

1. Functionally, how would you rate the mobile app?

[Lickert scale 1-5]: 1 = Awful, whatever it is, 5 = Pretty good for what it is

2. On a scale of 1-5, how difficult was it to join/create a lobby?

[Lickert scale 1-5]: 1 = Very Easy, 5 = Very Hard

3. For those who joined a lobby, did your phone vibrate/buzz?

[Multiple choice]:

   a. Yes
   b. No
   c. I did not join a lobby

4. (Optional) Please share any of your thoughts about the app. (Technical problems, general comments, suggestions, etc.).

Study instrument for remote testing, 2021-03-02 and 03-09

Pre-test questions

1. What Mobile Device do you have?

[Multiple Choice]:

   a. Android (Galaxy, Note…)
   b. iOS (iPhone, iPad…)
   c. I do not have a mobile device…
   d. Other… [User Response]

If user specified iOS:

2. What is your Apple ID email?

[Short Response]:

3. What model is your iPhone?

[Multiple Choice]:

   a. iPhone 12
   b. iPhone 11
   c. iPhone SE
   d. iPhone X
   e. iPhone 8
   f. iPhone 7
   g. iPhone 6
   h. Other… [User Response]

General Questions

2. What is your contact email address?

[Short Response]:

3. Will you be willing to meet remotely via Zoom on [date]?

[Multiple Choice]:

   a. Yes, I'll probably be there
   b. No, I can't make it
   c. Other… [User Response]

4. (Optional) Do you have any questions or concerns about *Percepto* and it's testing?

   [Short Response]:_____

Post-test questions

1. How difficult was it to correctly follow the instructions given to you?

[Likert Scale 1-5]: 1 = Very Easy, 5 = Very Hard

2. Will you be willing to meet on campus for the testing? (Date and time to be decided)

[Multiple Choice]:

   a. Yes, my phone vibrated
   b. No, I didn't feel a thing
   c. I did not join a lobby

3. What type of Mobile Device did you use?

[Multiple Choice]:

   a. Yes, my phone vibrated
   b. No, I didn't feel a thing
   c. I did not join a lobby

If phone vibrated:

4. When did you receive the vibration/buzz?

[Multiple Choice]:

   a. Right on time
   b. A couple seconds late
   c. I received it, but very late
   d. Other… [User Response]

5. Did you receive a push notification around the time the buzz occurred?

[Multiple Choice]:

   a. Yes
   b. No
   c. I'm not sure

6. What mobile device were you using during this test? (Please be a little specific of the model: e.g. "iPhone 11" note "iPhone 11 Pro Max" or "iPhone")

[Short Response]:_____

If phone did not vibrate:

4. Did you check / unmute the volume for notifications in your device's sound settings?

[Multiple Choice]:

   a. Yes, it was unmuted
   b. No, I didn't check

5. What mobile device were you using during this test? (Please be a little specific of the model: e.g. "iPhone 11" not "iPhone 11 Pro Max" or "iPhone")

[Short Response]:_____

If participant is unable to join:

4. Did you have a stable internet or 5g connection?

[Multiple Choice]:

   a. Yes, my internet was stable
   b. I was on 5g connection
   c. No, I did not have stable connection

5. What mobile device were you using during this test? (Please be a little specific of the model: e.g. "iPhone 11" not "iPhone 11 Pro Max" or "iPhone")

[Short Response]:_____

6. Please be more specific about what you did that prevented you from joining and what you remember happening. Don't be afraid to give extra information.

[Short Response]:_____

General Questions

7. Functionally, how would you rate the mobile app?

[Likert Scale 1-5]: 1 = Awful, whatever it is, 5 = Pretty good for what it is

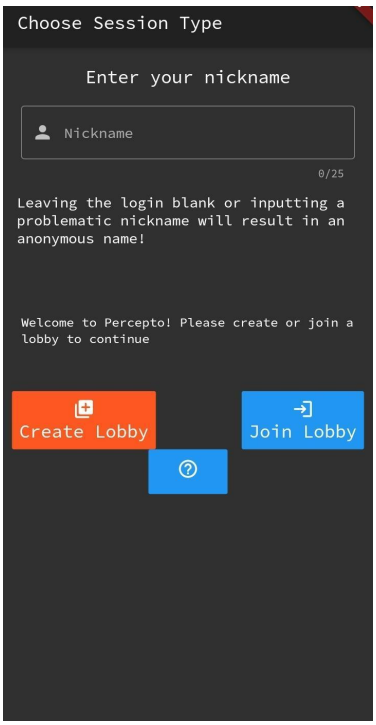
8. Visually, how would you rate the mobile app?

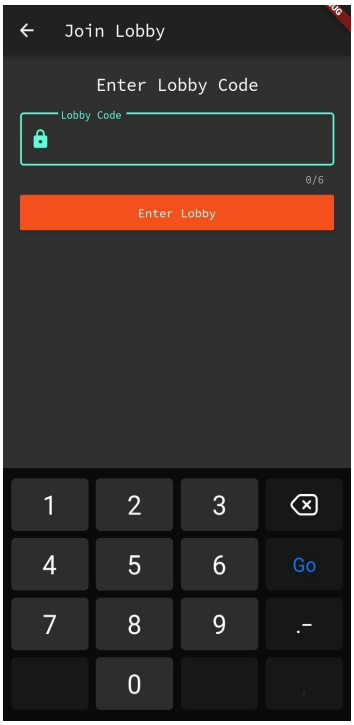
[Likert Scale 1-5]: 1 = Awful, whatever it is, 5 = Pretty good for what it is

9. (Optional) Is there anything else that you'd like to share? Any comments will be beneficial to the development process.

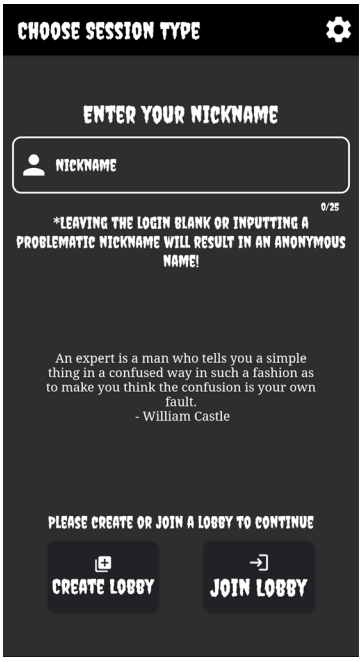
[Short Response]:_____

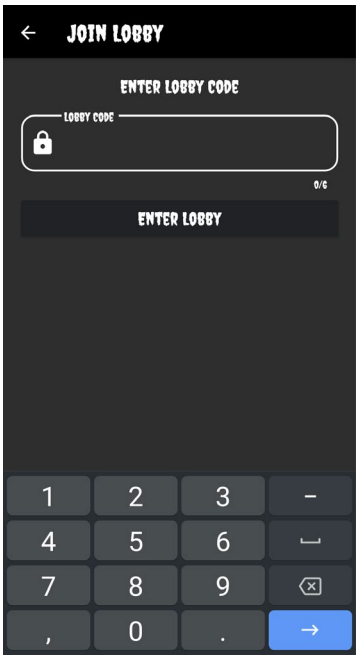
Appendix D: Prototype art

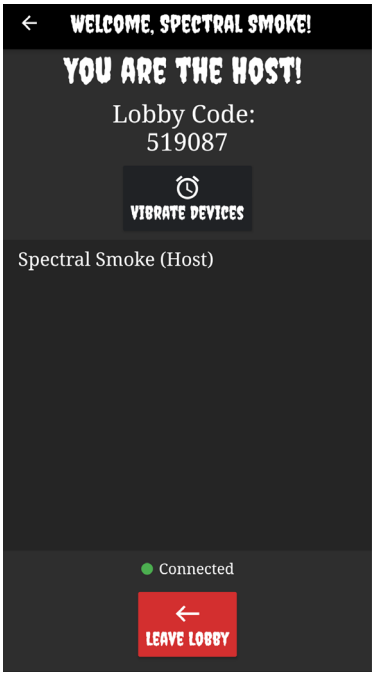


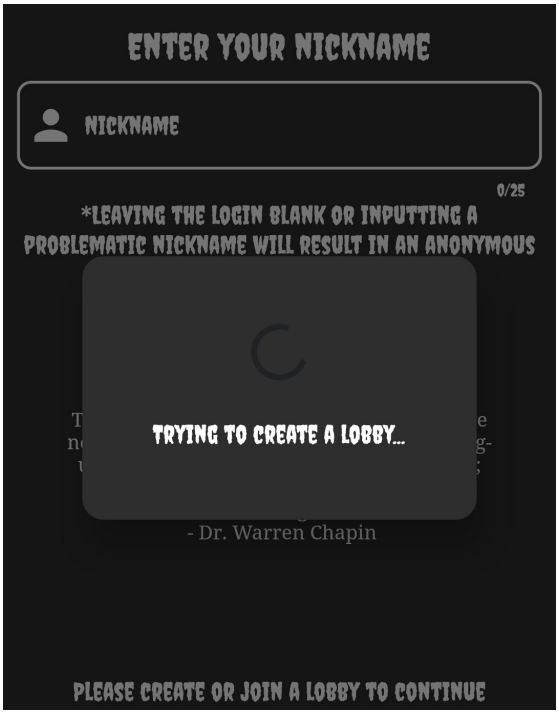


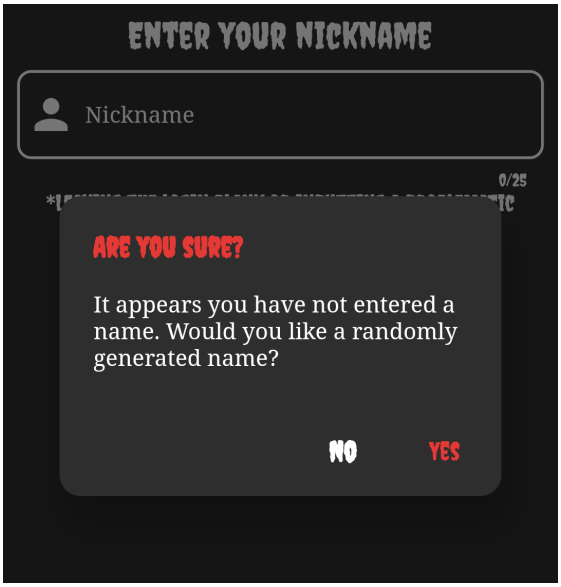
Appendix E: Dark mode interface

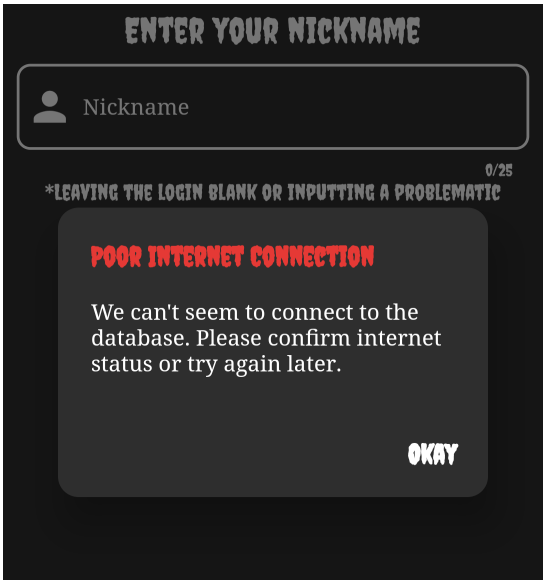







ENTER YOUR NICKNAME
NICKNAME
0/25
*LEAVING THE LOGIN BLANK OR INPUTTING A PROBLEMATIC NICKNAME WILL RESULT IN AN ANONYMOUS
TRYING TO CREATE A LOBBY...
- Dr. Warren Chapin
PLEASE CREATE OR JOIN A LOBBY TO CONTINUE


ENTER YOUR NICKNAME
Nickname
0/25
ARE YOU SURE?
It appears you have not entered a name. Would you like a randomly generated name?
NO
YES


ENTER YOUR NICKNAME
Nickname
0/25
*LEAVING THE LOGIN BLANK OR INPUTTING A PROBLEMATIC
POOR INTERNET CONNECTION
We can't seem to connect to the database. Please confirm internet status or try again later.
OKAY