# Technical Specifications:

**Important Version Info (see package.json form more info): NPM, Firebase Web Version 8, Expo 43.0, React 17.0.1.**

* **Integrating Firebase Cloud Functions with Google Calendar API**

1. Setting up your Firebase App

The first thing that the needs to be done for the project to run, is to create a Firebase App. Visit [Firebase Console](https://console.firebase.google.com/) , and after signing in to your Google account, you should get a list of all your Firebase projects and an **Add Project** button. When you click **Add Project**, you’ll be prompted to fill in your project details. The **Project Name** can be anything you like, but the **Project ID** must be unique through all Google projects. Be sure to accept all terms and conditions in the window. Once doing so, click **Create project** to finish creating your Firebase project. You can now enter your Firebase project.

1. Initializing our Firebase Project

First, open a command utility in the root of the reop and run *npm install* to download all the relevant node packages.

Then, now that you have a Firebase project initialized on your account, you’ll be able to add/deploy Cloud Functions. Open up the command prompt/terminal and navigate to the project workspace directory. First, you’ll need to download the **Firebase CLI** with *npm install -g firebase-tools*. Once installed, login to your Firebase account with the command *firebase login*. This will grant the CLI access to your account so that you can manage and deploy code to specific Firebase projects. Run *firebase init* to begin the setup of the project architecture for your cloud function. Select **Functions** by navigating with arrow up/down, then press space to select and enter to confirm. Next, the Firebase CLI will want to know which project you want your Cloud Function to be associated with. Select the correct PROJECT\_ID for the Firebase project and press enter. After pressing Enter, you will be given the option to write the Cloud Function in JavaScript or TypeScript. Choose JavaScript since the functions are already written in JavaScript. Continue with the rest of the project setup and DO NOT overwrite any files. For ease of use later, be sure to install the npm dependencies and a linter during your project initialization.

1. Create Calendar API Credentials

In order for the Google Functions to work properly with the Google Calendar API, you will need to generate API credentials. First, go to the [Google Developer Console](https://console.cloud.google.com/) and select the project you created, then select **Go to APIs** **overview** in the section for APIs. In the API Dashboard, select **Credentials** from the menu on the left and then on top select **Create Credentials** and then **OAuth Client ID.**

(You may see a warning banner near the top of the page saying:

“To create an OAuth client ID, you must first set a product name on the consent screen”

To fix this, click the Configure consent screen button on the alert and pick a name for your “product”. Once you name your product, you may need to select an email for a Support Email before the Save button is enabled. Once enabled, click Save to be redirected to the OAuth Client ID creation screen.)

Next, on the credentials screen, select **Web Application** under **Application Type** to bring up the other criteria needed for OAuth credentials. Name the OAuth Client ID something meaningful so you can keep track of the credentials for your project.)

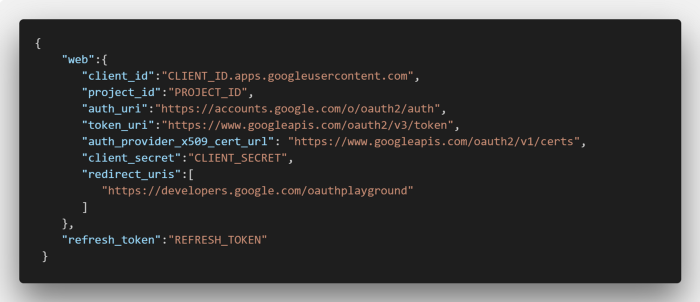
Under **Authorized redirect URIs** add the URI <https://developers.google.com/oauthplayground>. Once the redirect URI is added, click anywhere outside the input to ensure the URI was entered correctly. If everything looks correct, click the Create button at the bottom.

Now go back to the **Credentials** page, and under the section **OAuth 2.0 Client IDs** you should be able to see your credentials. On the left side of the entry you should see a download icon, click that and it will allow you to download your credentials in a JSON file. Rename this file to *credentials.json* and save it in the *functions* directory. You should overwrite the existing *credentials* file.

1. Configure OAuth2 for your project

Now that we have your OAuth Client credentials, you’ll need to actually go into your redirect URI and configure the OAuth2.

To do this, go to <https://developers.google.com/oauthplayground>. On the page, click the **Settings** icon in the top-right of the screen to pull up the configuration settings. Select **Use your own OAuth** **credentials** at the bottom to bring up inputs for the Client ID/Client Secret you just created. After entering both the Client ID and Client Secret from your credentials.json file leave the above window open. On the left-side of the screen, search for the Calendar API v3 and select *https://www.googleapis.com/auth/calendar* as the scope for the cloud functions. Once that scope is selected, click **Authorize APIs** at the bottom of Step 1. You’ll be taken to a Google sign-in page where you sign in to the Google account who’s calendar you want to access. Once signed in, you’ll be asked to confirm access for your credentials to manage your Google calendars. Approve of this message to continue. This will redirect you to Step 2 of the OAuth2 Playground form. You should have the **Authorization code** populated. This code is only valid for a short period of time (3600 seconds precisely). While it’s active, you’ll need to click Exchange authorization code for tokens to populate a refresh token that we will be able to use later in our cloud function. Then you will need to add this refresh token to the *credentials.json* file*.* An updated version is below, and your file should be the same format.



1. Deploy Cloud functions

Now you will need to deploy the functions in the index.js under functions directory to Firebase Functions by running the command *firebase deploy --only functions* in the root directory of the project. After a minute or two, the Firebase CLI will say that the Cloud Function is deployed.

1. Instantiate Firebase Firestore

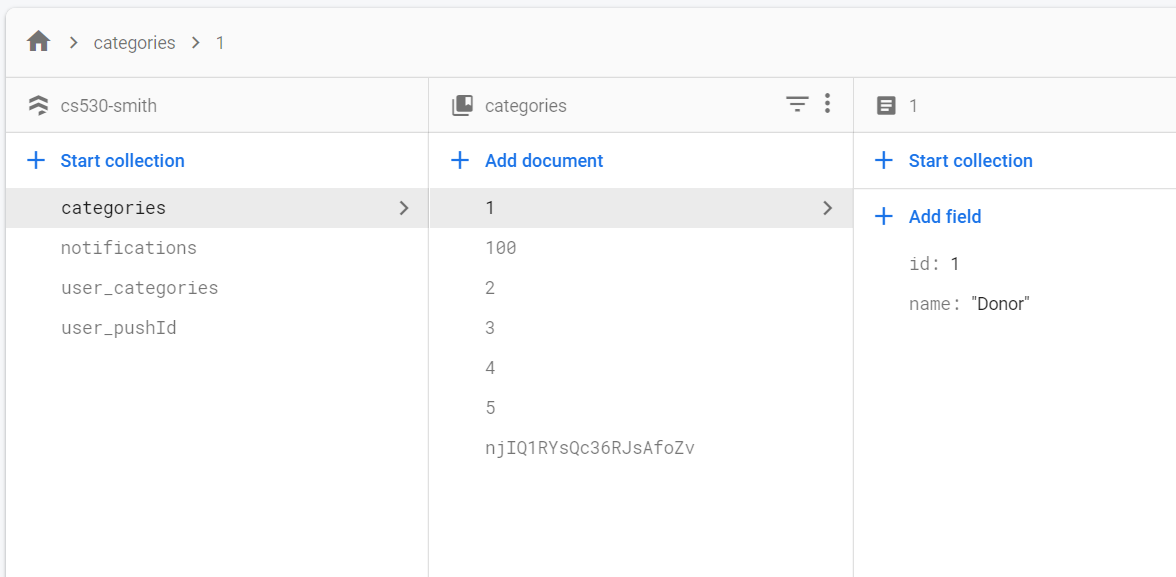
From the [Firebase documentation](https://firebase.google.com/docs/firestore/quickstart#create)

* 1. Navigate to the **Cloud Firestore** section of the [Firebase console](https://console.firebase.google.com/project/_/firestore). You'll be prompted to select an existing Firebase project. Follow the database creation workflow.
  2. Select a starting mode for your Cloud Firestore Security Rules:
  3. Test mode
     1. Good for getting started with the mobile and web client libraries, but allows anyone to read and overwrite your data. After testing, **make sure to review the** [**Secure your data**](https://firebase.google.com/docs/firestore/quickstart#secure_your_data) **section.**
     2. To get started with the web, Apple platforms, or Android SDK, select test mode.
  4. Locked mode
     1. Denies all reads and writes from mobile and web clients. Your authenticated application servers (C#, Go, Java, Node.js, PHP, Python, or Ruby) can still access your database.
     2. To get started with the C#, Go, Java, Node.js, PHP, Python, or Ruby server client library, select locked mode.
  5. Select a [location](https://firebase.google.com/docs/firestore/locations#types) for your database.
     + This location setting is your project's [*default Google Cloud Platform (GCP) resource location*](https://firebase.google.com/docs/firestore/locations#default-cloud-location). Note that this location will be used for GCP services in your project that require a location setting, specifically, your default [Cloud Storage](https://firebase.google.com/docs/storage) bucket and your [App Engine](https://cloud.google.com/appengine/docs/) app (which is required if you use Cloud Scheduler).
     + If you aren't able to select a location, then your project already has a default GCP resource location. It was set either during project creation or when setting up another service that requires a location setting.
     + **Warning:** After you set your project's default GCP resource location, you cannot change it.
  6. Click **Done**.
* The security rules for the framework as delivered should be set to:



These rules ensure that only authenticated users can read and write from/to the database. All users of the app are forced to authenticate anonymously through existing code.

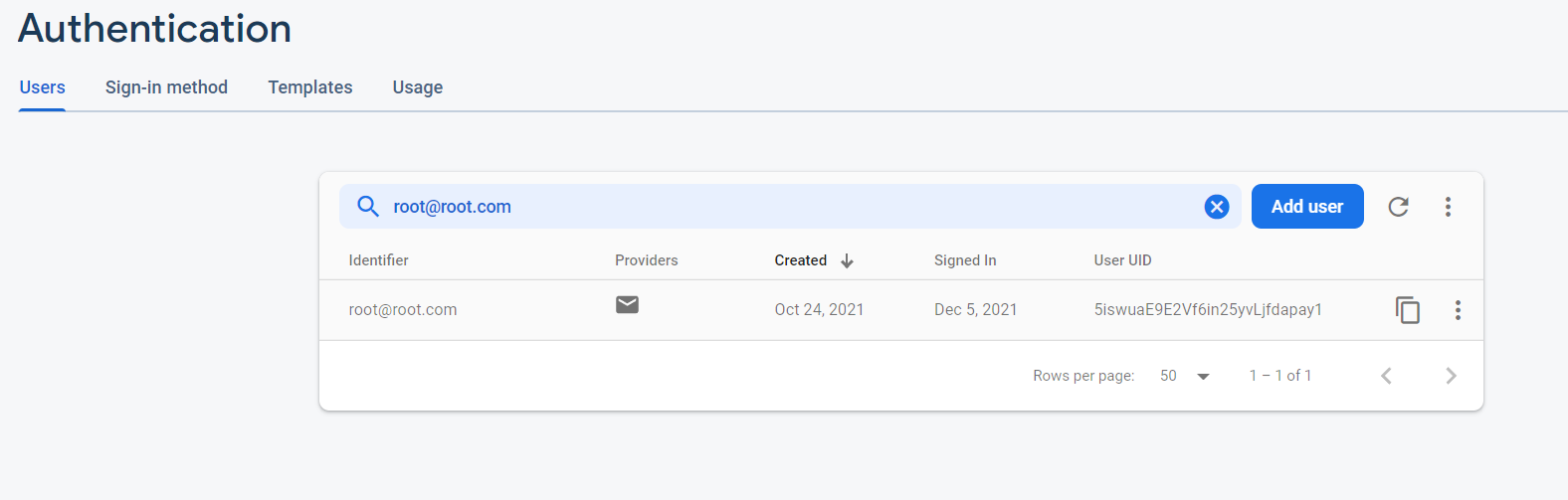
* You must create only one Firestore Collection manually, name it “categories”, and fill with desired subscription choices. The document name is unimportant due to code logic, but each must have a distinct “id:” of type number and “name:” of type string.



* The other collections will create themselves dynamically when the application writes to them.

1. Allow anonymous sign in and create admin username/password accounts.

You must enable anonymous and email/password authentication to use the full framework as delivered. All users writing to and reading from the Firestore database will be required to sign in anonymously to Firebase. All users created manually with email/password authentication will be able to login to and access the Admin functions in the app.



# Main Features:

There are four screens in this app: Calendar, Settings, Notifications, and Admin. The Admin screen needs special authentication to access, while the other three can be accessed by anyone who has the app.

Note: when the user is running the app in the foreground or in the background, they may receive push notifications from the app. Details on how to receive and send notifications are covered in the sections for the Settings and Admin screens.

### Calendar Screen:

The top section of the screen shows a calendar. A user can click the arrows in the top left and right corners of the calendar to change months. Selecting a day will cause the list of events scheduled for that day to display in the bottom section of the screen (switching days switches what events get displayed). Displayed events are ordered by event time with the earliest ones at the top of the list. Days that have events scheduled are marked in the calendar via filled blue circles surrounding the day numbers.

### Settings Screen:

This screen contains a list of categories, each with a checkbox to its left. The user may check checkboxes to subscribe to the corresponding categories and may uncheck checkboxes to unsubscribe from those categories. Users subscribed to any categories will receive push notifications targeted at those categories. A user only needs to be subscribed to one of the categories a push notification targets to receive the notification.

### Notifications Screen:

This page shows a list of all notifications that have ever been sent associated with the categories the user is subscribed to. The notifications will show up even if the user was not subscribed to an appropriate category when the notification was originally sent out. The notifications are ordered by time sent with the most recent ones at the top.

### Admin Screen:

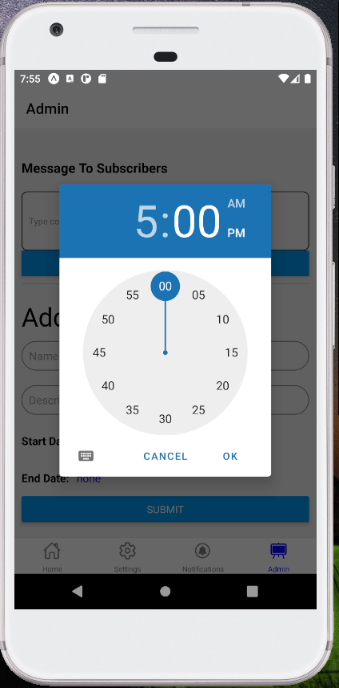
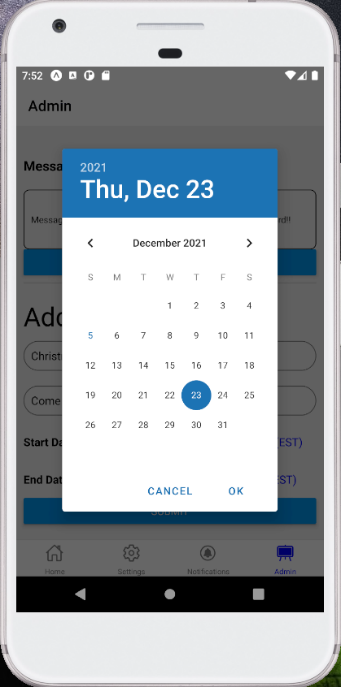
When the user first accesses this screen, they will be prompted for their username and password (this dialog prompt is technically another screen, but we don’t deem it important enough to include the list of screens).

After successfully logging in, or after returning to the Admin screen after having already logged in recently, the user will be presented with the “real” Admin screen. Thís screen consists of three sections: a list of categories at the top, a form for sending messages in the middle, and a form for adding events at the bottom.

Each category in the list has a switch to its right. The user may turn the switch on (or off) to select (or unselect) a category. More on selected categories shortly.

In the middle section consists of a text area and a “Send Notification” button. The user may edit the contents of the text area at any time. When the user pushes the button, a notification whose content matches the text area’s will be sent to all users who are subscribed to at least one category selected in the above list. Right now, all notification titles are hardcoded to “Message from Charity”, but this can be easily made dynamic with future development.

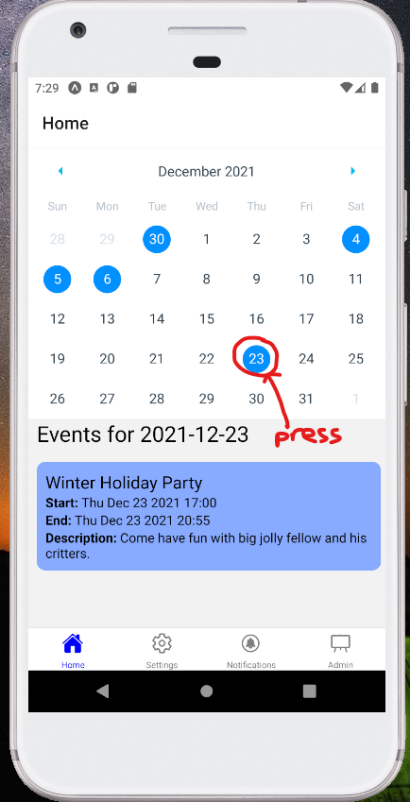
The bottom section is the form for adding events to the calendar. It has text inputs for the event name and description as well as special date pickers for the start and end datetimes. To select a datetime, a user clicks the blue underlined text next to the label a very intuitive (see below screenshots) dialog will appear allowing the user to select a day and time. The user clicks the Submit button at the bottom to add the event. The Name, Start Date, and End Date fields must be filled for adding an event to trigger. If at least one of these is not filled, a message will display above the button informing the user that an unfilled field (the first of the above required fields that is not filled) is required.



Note: the events and categories have no associations whatsoever in the current incarnation of this app. The calendar always shows all events regardless of what categories to which the user may be subscribed. We made it this way because we suspect that while people would not wish to be bothered with notifications for categories that do not apply to them, they may desire to brows events of all categories.

# Main Scenario Walkthrough:

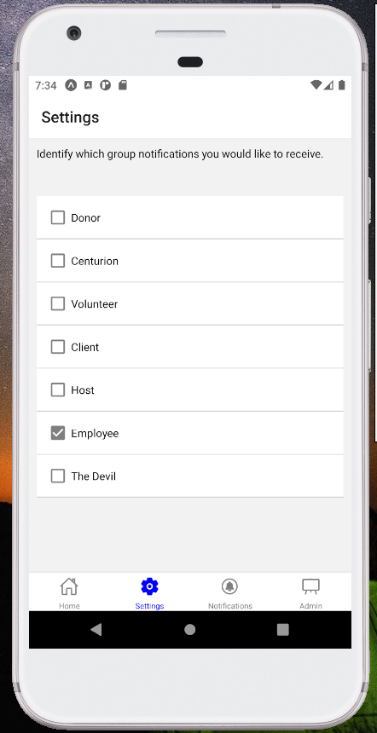
A user running the app on their phone in the background receives a notification via the app that their company is hosting a holiday party on December 23rd. The user has received this notification because they are subscribed to the Employee category. The user wants to know more about this event, so they open the app, and select December 23rd of the current year on the app’s calendar. Then, in the list below the calendar, they see the event with its description, start time, and end time.



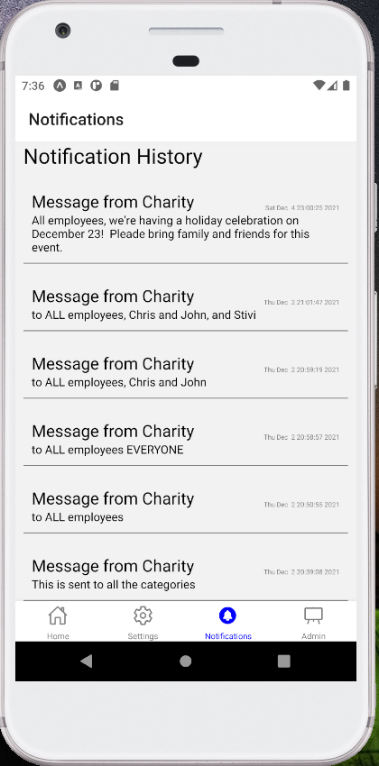
# Alternative Scenario Walkthroughs:

**Scenario #1:**

A user wishes to subscribe to the Volunteer category because they are now involved in volunteer activities related to their company. The user goes to the Settings screen, and checks the checkbox next to “Volunteer “.

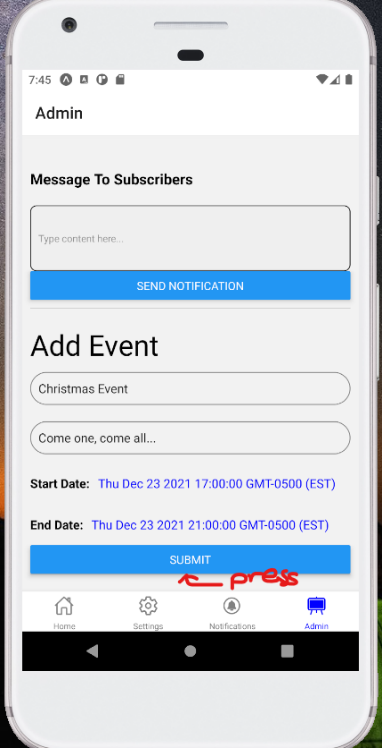
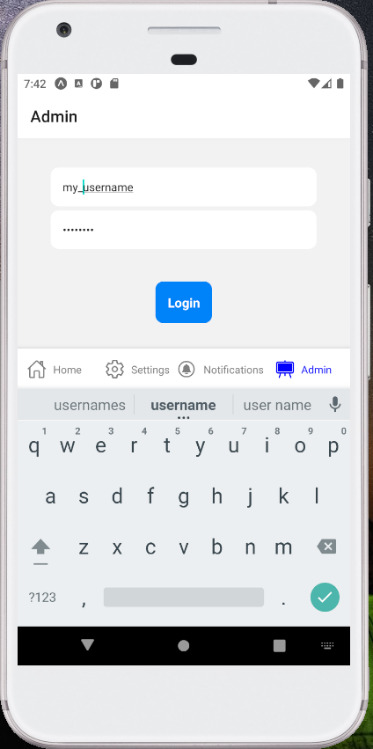


The user also wants to be caught up on any notification they missed while they were unsubscribed, so they go to the Notifications page and look at all the Volunteer notifications in addition to all the other notifications they were able to view.

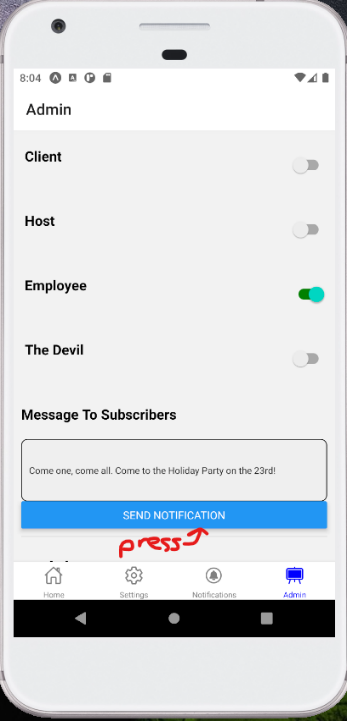


**Scenario #2:**

An administrator wants to schedule a company-wide holiday event to occur from 5:00 to 9:00 pm. on December 23rd. They go to the Admin screen, log in, and enter the event information at the bottom (name, description, start time, and end time), and press “Submit”.



The user also wants to inform all employees of this event, so without leaving the Admin screen, they make sure only the Employee category is selected at the top, type their message in the text area in the middle, and press “Send Notification”.



To make sure everything was successful, the user checks the app’s calendar and notification history. On the calendar, they see their event correctly added under December 23rd. Then, they make sure they’re subscribed to Employee (on the settings screen) and then check the Notifications screen. They see their notification at the top of the list, where it should be.