Gather 

Team Sweet Tooth

Tyler Lichten, Christine Kim, Chelsi Hu, William Edgecomb

Gather is a crowd-sourced social platform for sharing and exploring public events.

The Gather app is designed to provide a resource of categorized public events. Any logged in user may add an event to a category, and it will be public to other users subscribed to that category. A user may choose which categories to subscribe to and all events of those categories will be available to the user. Navigation is controlled by a bottom menu with four tabs. The first tab is the home tab where users may view upcoming events in which they have indicated interest, as well as discover top events displayed in a rotating animation. The second tab is the "my events" tab, which displays a detailed list of events the user is interested in. The third tab is the "explore" tab, where a user may browse events that match the categories he or she has subscribed to. This tab contains a calendar, and upon a date in the calendar being clicked, a dialog is displayed that shows the events occurring on that day. The final tab is the settings tab where the user may select categories to subscribe to. Logging in and logging out is also managed in settings..

Motivation:

Sometimes fun events in your area pass you by without your even knowing. Our app connects users to the information they need to never miss out on their favorite events, and to always have options when it comes to the matter of *just what should we do tonight?* And who better to provide the information on events than your fellow locals?. Gather leverages the collective knowledge of the greater community to help everyone make the most of their leisure outings.

Other features:

Besides allowing you to filter events by category (i.e. Music, Sports, Business, Community, and Other), Gather tracks the number of users who are currently interested in an event to help make popular events more visible. The events suggested on the home tab are selected based on popularity, and in the explore tab, events on each day are sorted by popularity. Furthermore each event can be viewed in a separate page that displays the number of users interested. Also from this activity, users can set their status relative to the event as "interested", (they can also rescind their interest, so no commitment required!). The home and my events tabs will automatically adjust to display the events user has indicated interest in. To add an event, there is menu item that persists in the upper-right corner across all four tabs. Clicking it takes one to a

separate activity where one can enter all the required fields for an event and add it to the public calendar.

Our Code Organization and Cool Things We Used:

Our app utilizes Fragments. Each “tab” (Home, My Events, Explore, Settings) is a fragment, and navigation between fragments is managed by a Roughike (third-party) BottomBar menu object attached to the activity *Main*. We also use Google Firebase to manage user authentication, data storage and retrieval.

1. Activities:

- *Main*: sets up BottomBar object and directs user to settings if no valid login, otherwise to home
- *AddEvent*: takes user input to create event and add it to public calendar
- *CompleteEvent*: shows full detail on single event; allows user to indicate interest in event, or rescind interest.

2. Objects:

- *DateTime*: represents date and time using integers; a field of *Event* and used for extracting date information from the database;
- *Event*: represents single public event; events can stored and extracted from database

3. Fragments:

- *HomeFragment*: (see section below)
- *MyEventsFragment*: (see section below)
- *ExploreFragment*: (see section below)
- *SettingsFragment*: (see section below)
- *TimePickerFragmen*t: for picking the time of an event when adding a new event to the public calendar.
- *DatePickerFragment*: for selecting the date of an event when adding a new event to the public calendar.

4. Support Classes:

- *DatabaseUtility*: instance in *HomeFragment* and *MyEventsFragment*; has one principal method that iterates through Firebase database to get events user is interested in and populates GridView or ListView respectively using *MyEventsHomeTrackerAdapter*. Also sets a listener so that views will be automatically updated upon user’s selected events changing in the database.
- *MyEventsHomeTrackerAdapter*: custom adapter for populating GridView in home fragment and ListView in My Events fragment.

Each bottom bar tab:

1. Home Fragment

The principal elements of the home fragment are the GridView of events user is interested in and the ViewFlipper of suggested events. The GridView is populated using a DatabaseUtility object, but only if there is a user logged in (userID not null). Otherwise the GridView is not set.

The GridView entries are each a relative layout of three elements of the Event object, that is the date, title, and category. If no upcoming events are found, a button is displayed that directs the user to the explore fragment.

To set up the viewflipper, all events are pulled from the database and the five events with the most checks are created and used. The viewflipper then consists of three elements: information about the event, image corresponding to event category, and button to pull up more information about the event. Using the five events' information, we update the imageview and textviews. Then the event is bundled and used to create the activity to hold the additional event information. Onclick listeners are added to each of the buttons to start each of the activities. Then, the viewfliper is set to flip automatically by a time interval, or by swiping, with a touch listener to check the x coordinate of the event.

2. MyEvents Fragment

This fragment's only significant element is populated in parallel fashion as the GridView in My Events, that is using a DatabaseUtility object and MyEventsHomeTrackerAdapter. The underlying xml element is different however. In MyEvents a relative layout is also used, but each event is represented with full detail.

3. Explore Fragment

The calendar is a fragment that is embedded within the explore fragment. The user is able to click on a date and see what events are scheduled for that day. When the user clicks on a date, a dialog pops up with a list of events for that day, but only if they meet the user's category subscription preferences (see Settings Fragment for more info). The events are sorted by popularity and show the most interested in events first. The dialog is built with an ArrayAdapter and each cell is a TextView that displays either the event title with the number of users interested. The first TextView simply displays "Event on MM/DD/YYYY" and is unclickable. The user can click on any of the other events and it will startActivityForResult and start the AddEvent activity. If user clicks that they are interested, a boolean true is returned to the exploreFragment upon pushing the back button (onBackButtonPressed). The true value is used to avoid inadvertently showing the dialog list again (onDataChange) that occurs in the Firebase database and is looking for data changes in the exploreFragment. The exploreFragment builds hash maps from the database that contain <event key,Event object> pairing in order to have each event be unique, since the key is unique to each event, and hold the Event object in order to extract field data such as number of interests, event title, event date and event category. When months are changed (onChangeMonth), the calendar will make an update to the data. The calendar also colors the grid view cells depending on the most popular event of that day. The color is coordinated with the event category, so users can quickly identify which days have events that they prefer. Under the calendar, there are six text views that are have background colors that match those in the calendar as a reference.

The calendar is built using the libraries of a popular open-source project on GitHub called Caldroid, which provides the general framework of the calendar and allows for updating

changed months by the powerful Java date4j libraries. The libraries were added to build.gradle in our project.

4. Settings Fragment

Google login is implemented to tie each user to events in the database. The Google Repository is downloaded to be used with the sdk, which allows the use of the GoogleSignInOptions, which retrieves a unique token for the user using their Google Credentials, and GoogleApiClient which builds Google's Sign in activity to handle those google credentials. The Google token is passed to FirebaseAuth which authenticates the token with Firebase. On subsequent app launches, the data for the two are retained, and the user stays logged in. Then, we tie user specific info in the database to the each user's User ID, which is requested from FirebaseAuth. To be able to use Google login in the debug app, a custom keystore was created for the app, and the SHA1 fingerprint for the keystore was passed to Firebase.

A user may subscribe to categories in the settings fragment. The Subscribe to Categories button creates an Alert with checkable options. The state of each check is handled by a boolean array, and each category string is handled by a string array. Both arrays were created from pulling information from the database by checking which categories are tied to which user, and all categories in the database. When the user finishes checking, the updated boolean array is used to update the database with which categories they are subscribed to.

Firebase Database:

The database is organized is in JSON form and has a flat hierarchy. There are three main nodes: *Events*, *CategoriesToEvents, UserToEvents, and UserToCategories*. Events stores event objects with unique auto-generated keys. UserToCategories maps unique auto-generated user keys to users' subscribed categories. CategoriesToEvents maps categories to their lists of events by event ID. And UserToEvents maps users to their lists of events by event ID. A example file of the database is provided in the project in the app folder, titled GatherDatabase.json. Also note that access to database is made possible via dependencies added to build.gradle, i.e.:

```
compile 'com.google.firebase:firebase-database:9.6.0'
```