Alike!

Amartya Sanyal Kriti Joshi

November 16, 2016

Abstract

Alike! is an android chatting application which consists of a server built using Node.js, a real-time database maintained by Google Firebase and multiple Client programs. After a client logs in, he is intelligently matched with another online client. The matched clients can send textual data to each other and can also chat in a global chat screen before matching occurs. After a chat, the user rates his experience and using this data future matches are made.

Components

1. App: The android application consists of the listed important parts:
   (a) Login and Register: Google account of the user is taken for Login to avoid fake chat bots and objectionable content during chat.
   (b) Chat : Global chat screen where all online user can send messages before getting matched. A private chat screen for conversation between matched clients.
   (c) Like Or Dislike : Rate the chat experience after exiting a private conversation.
2. Server : The server uses Firebase-admin and Firebase Queue Node.js libraries to interact with Firebase. Following four important functions are handled by the server.
   (a) Wait Function: This function periodically waits for a period of 5 seconds after which it sends the list of active users to *Match Clients* function.
   (b) Match Clients: Using the list of active users and the *Like Table* maintained at the database, the function assigns matching clients to an active client.
   (c) Update Like Table: After completion of a private chat, the app sends a task to the database queue with the user ids of both the clients and the conversation rating. This function updates the Like Table according to the sent data.

(d) Delete Last Conversation: To maintain privacy, after completion of a chat the message table for that conversation is removed using this function. It also makes the *'match'* field null for both the clients indicating none of them is matched to any client after the conversation gets over.

3. Firebase Database: Google's firebase database is used which interacts with both server and client which in turn do not interact at all with each other.

   (a) Firebase Queue: It works as a job scheduler. The tasks related to the above mentioned four functions get enqued either by server or client and are then handled in FIFO order by multiple workers (manual setting)

   (b) Database: A real-time no sql database is maintained which contains all conversation messages, active-user list, task queue and like table.

Architecture

Let us first show two pictures that describe the model view and the component view briefly and then we will go on to describe how the architecture is implemented and highlight the important features for the same.

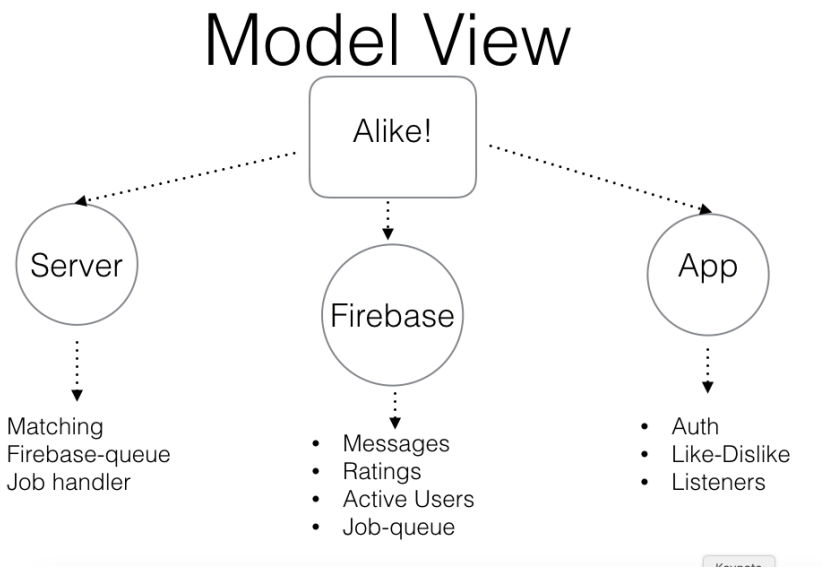

Figure 1: Model View.

The process view of the application could very simply be described as follows : What we are trying to stress here is the fact that the app server and the

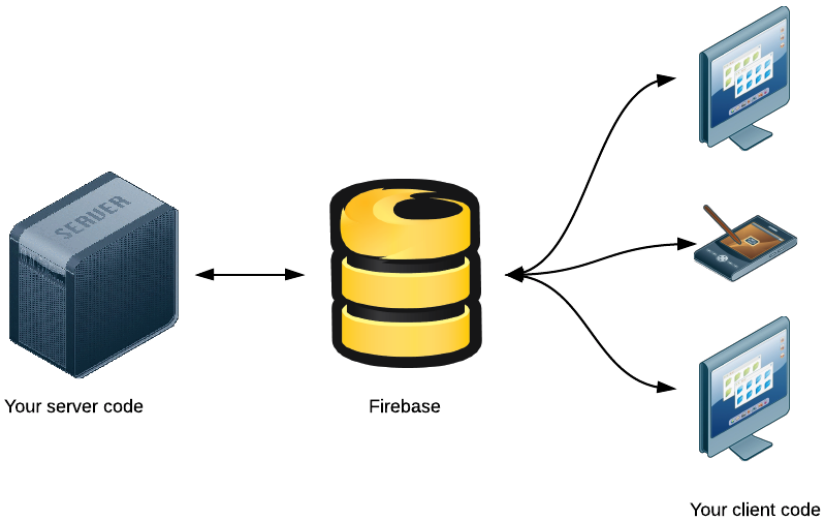

Figure 2: Simple depiction of Process view.

android app are completely decoupled. This means that all communication between the application and the server are handled by an intermediary that is the firebase.

All communication is done by listeners sitting on the android app and the app server. It is basically an event-driven application. Let me explicate the scenario with the help of an example. As soon as an user is active, the app updates the **uid** table in the firebase database, with its *name* and *uid*. The picture below shows it.

The **wait** job clusters the currently active users and launches a matching job, which updates the *match* field in the **uid** table for the corresponding user.

Figure 3: Active user list.

This activates the listener on the android app asking it to start a private chat.

Queuing jobs

When the app requries to update certain items in the database, which requires a bit of computation, it does not do that by itself. It queues a job in firebase under the queue table as *task*. A component, running on the app server, is listening to this queue and as soon as a task pertaining to it is appended to this list, it starts processing. As described above, there are four different types of jobs that can be queued. The matching and the waiting job are queued by the waiting job and the like/dislike and delete-match job are queued by the android app.

Real Time messaging

The messaging is also handled by an event-driven architecture. As soon as a match is created, a new table is created witht eh chatid and both the clients starts subscribing to this new table. All chats are entered in chronological order(this is handled by firebase) i.e. the relative ordering of messages sent by an user is maintained. As soon as the message list is updated i.e. a new message is available, the other person gets an alert on its listener and the message is available on his screen. This completely takes away the necessity of an app server in the task of chatting and hence allows a large number of users to chat in real time by exploiting the high quality real-time updates handled by firebase.

Stopping a chat

There are two ways to end a chat. The first one is by closing the app and the second one is a by a specific *Close Chat* button in the app. A sequence of events happens once this is done and it is triggered by the app by queueing jobs in firebase queue. The first one queues a job to delete the *matchid* field in the uid table as well as the particular message table that stored the messages for the job. This deletion of the *matchid* field initiates a callback in the app, requiring you to enter whether you liked or disliked the job. As soon as you enter the like or dislike, the app queues another job, which updates the rating and the total number of jobs in the likes table in the firebase. Once, these two jobs are done, both the users subscribe to the public chat board and waits to be matched again.

Matching Algorithm

For matching n online users, we use their ordering in the *Active Users* table. We start from the first one and then find the *alikeness* with each non-matched online user. The one with the maximum alikeness is chosen. The alikeness is calculated using given formula.

Here alikeness(i,j) is the alikeness between the two users i and j.
$r_i(j)$ is the rating given by user i to user j.
$tc_i(j)$ is the total number of chats between user i and j.
k are the users which have had a chat earlier with both i and j.

$$alikeness(i,j) = \frac{r_i(j)}{tc_i(j)} + \sum_k \left(\frac{r_i(k)}{tc_i(k)} * \frac{r_j(k)}{tc_j(k)}\right)$$

Note that the given function not only takes into account the rating given to user j by user i but also the ratings both of them gave to same users.

Figure 4: Like Table

Screenshots from App

Here we are giving some screenshots from the app. The first one shows what a screen looks like when it is quitting a chat and the second one shows what a regular chat screen looks like.

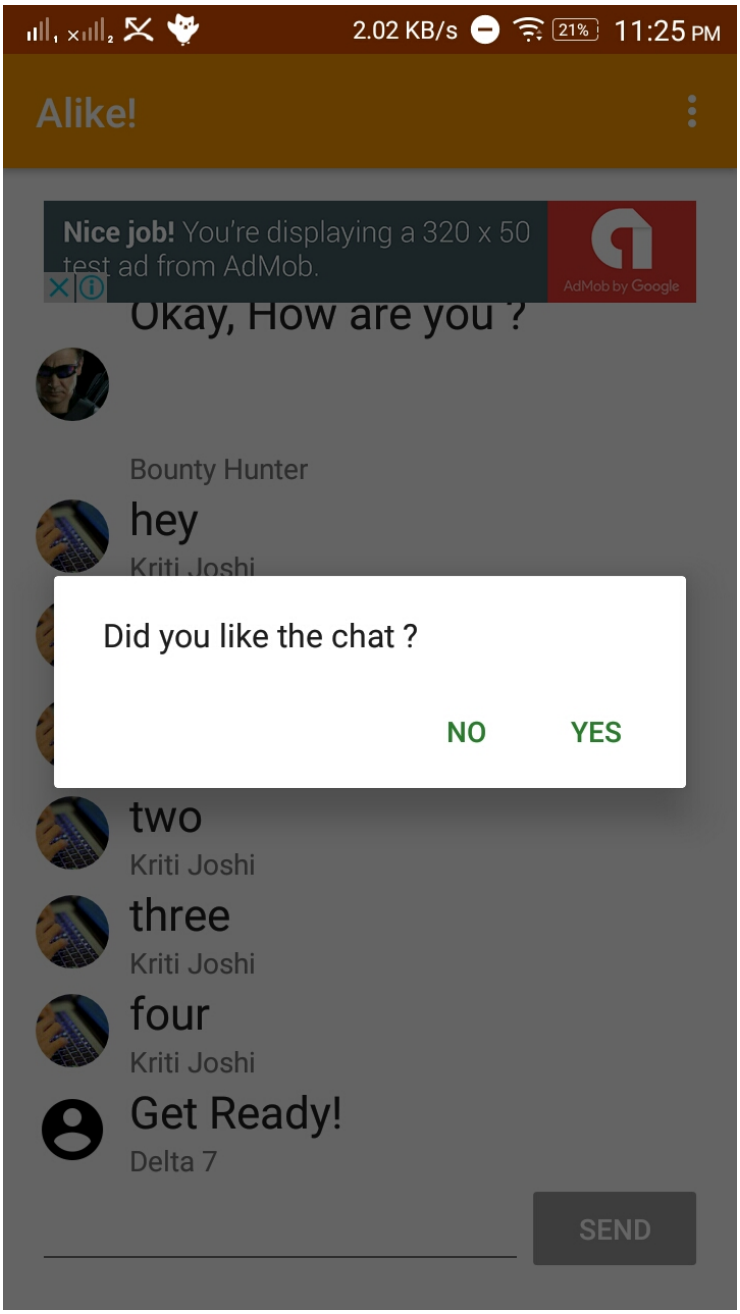

Figure 5: Choosing whether you liked the chat or not.

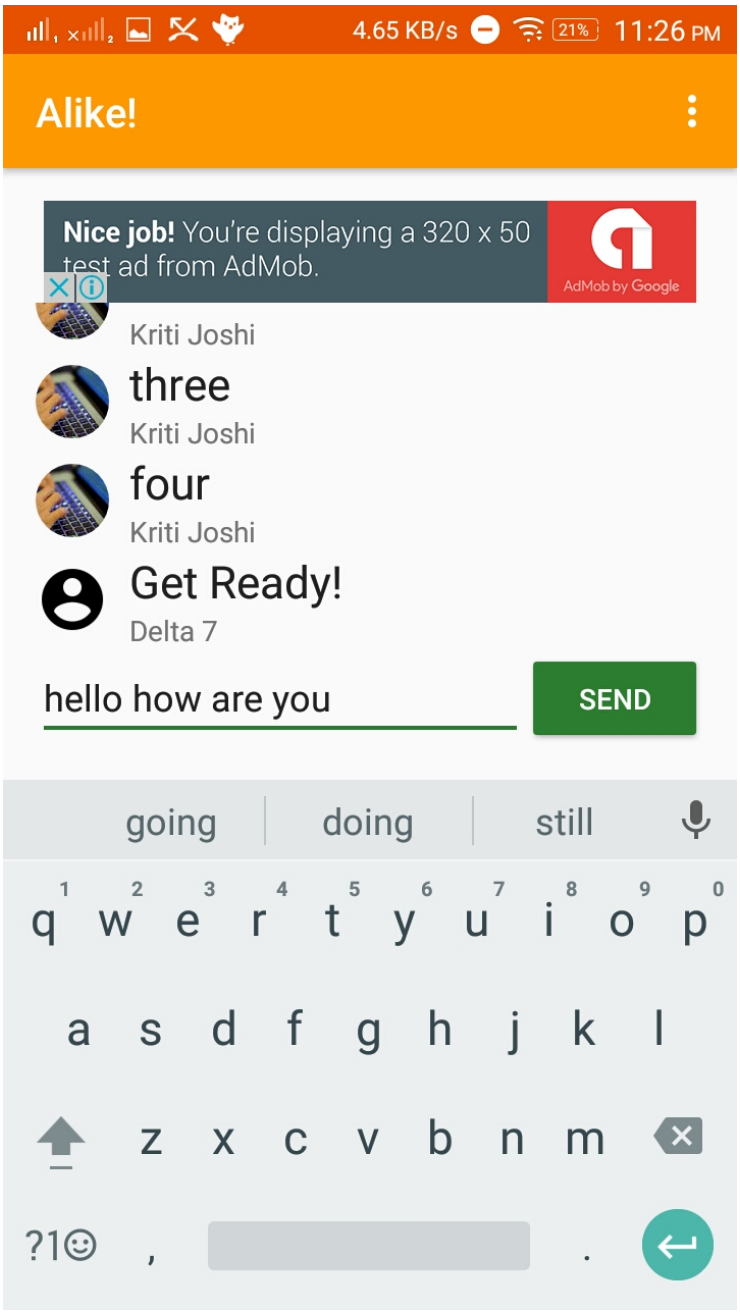

Figure 6: A regular chat screen