Q&A Design Document

December 15, 2017

Gururaj Shriram

CSC 411: Project Implemetation

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the design and implementation of the Q&A Android application.

1.2 Problem Description

In today's age of technological innovation, knowledge is, for the most part, easily accessible and available. However, an immense amount of knowledge exists solely in the minds of people, unconnected to the internet. Both questions and answers to questions differ across geographical boundaries; what may seem to be a correct answer to a problem in one region may be egregiously wrong in another region. Further, there are many queries whose solutions require a localized knowledge; for instance, this could range from asking for the best Italian restaurant in a certain city to figuring out where a garage sale selling electronics may be taking place. The question then arises: how can we share all of this localized knowledge?

2 Project Overview

2.1 Goal

The goal of the Android application, Q&A, is to provide a forum that will connect people with localized knowledge to people who need it. The application will allow users to post questions or issues and receive answers from other knowledgeable users in the same general location in realtime.

2.2 Technologies Used

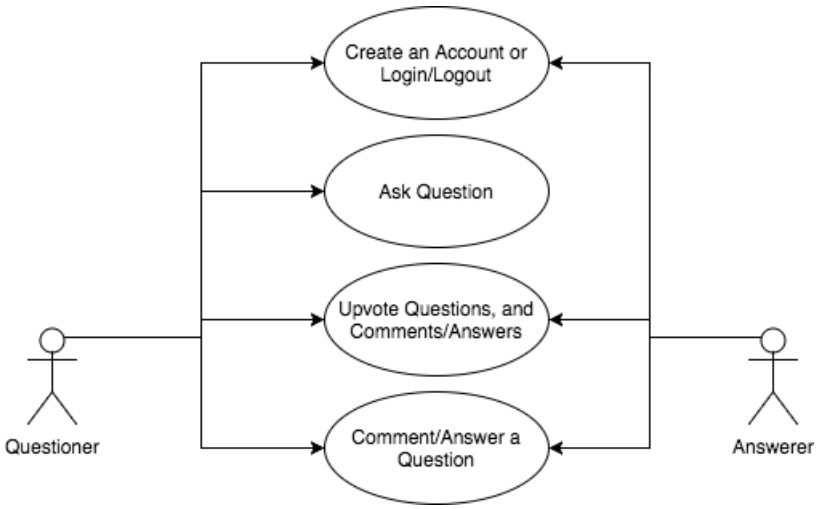
As Q&A is an Android application, it will be written in Java for the app logic and XML for the user interface. The primary backend for the app will be Firebase, which is a mobile and web application development service. Firebase includes services such as cloud functions, messaging, authentication, and a realtime database, which will all be invaluable for the application. To push notifications to the user, Firebase Cloud functions written in Node.js will be used to listen to database triggers and send messages using the Firebase Cloud Messaging service. Further, for the location aspect of the app, Q&A will use GeoFire, which is a set of open source libraries that use Firebase's realtime database to store and query data based on geographic location.

3 Technical Design

3.1 Use Cases

The core functionality of the application should ideally include a mechanism for users to create and sign into accounts, ask questions, view questions from other

users in the area, comment on and answer questions, and vote on interesting questions and answers. The following Use Case Diagram summarizes the basic user flow with two users, the "Questioner" and the "Answerer":



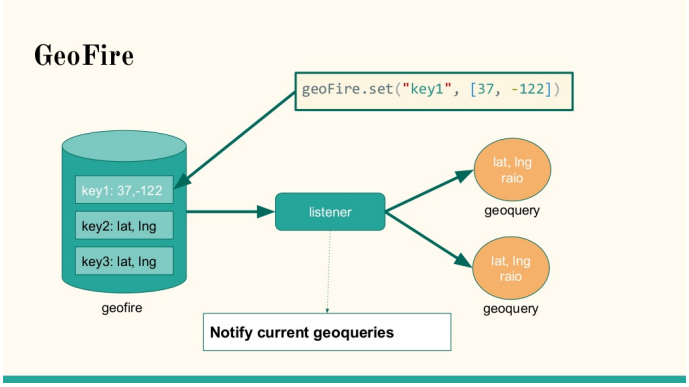
3.2 Authentication

Having a user's identity will allow Q&A to securely store necessary user data in the database, such as a user's name, saved questions, and so on. Firebase Authentication provides the backend services and UI libraries, collectively named Firebase UI, needed to authenticate users. The authentication utilizes OAuth 2.0, which uses transport layer security for added confidentiality, and OpenID Connect, which is an identity layer on top of the OAuth 2.0 protocol. It also supports authentication using quite a few identity providers, but the initial version of Q&A will use email-password authentication as well as Google single sign on.

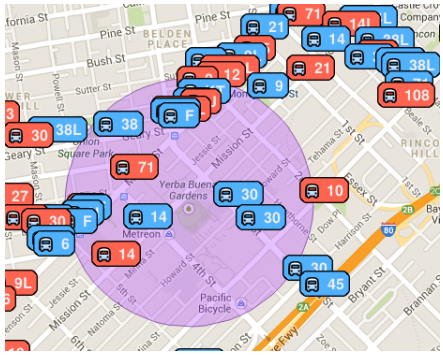
To sign a user into the app, the app will send either the user's email address and password or an OAuth token into the Firebase Authentication SDK which will then verify credentials and return a response to the client. If the login is successful, Q&A will be able to retrieve basic user profile information and allow the user to write to the database in the form of questions and comments on questions. If the login is unsuccessful, the user will be redirected back to Firebase UI's sign in page. Finally, if the user does not have an account and wishes to create one, Firebase UI will automatically detect this during login (by checking whether the identifier used already exists in the Firebase Authentication console) and prompt the user to enter their chosen email address and password.

3.3 Realtime Location Queries

A key aspect of Q&A is the ability to associate location data with questions as well as retrieve questions posted within the same location in realtime. For this, Q&A will use GeoFire. In its simplest form, GeoFire stores locations with string keys, which in this case will be the unique ID of a question thread. GeoFire optimizes the location queries using a GeoHash, which consolidates latitude and longitude pairs, or a GeoLocation, into a hash.



To query the database, a GeoQuery is created and given a center, in the form of a GeoLocation, and a desired radius. The GeoQuery then returns all of the keys and locations within the bounds of the aforementioned circle. These keys can be used to retrieve data from question threads and display relevant information for the user. The GeoQuery has an asynchronous event listener which gives a callback whenever a key, or ID of a question thread, enters or exits the user's search radius. Therefore, the user will always view question threads that were posted within a certain radius of the user's location. A more in depth explanation of the mechanism for data storage follows in the next section.



3.4 Data Model and Storage

All of Q&A's data will be stored in the Firebase Realtime NoSQL Cloud Database. This is to allow data to be synced across all clients in realtime without modifying networking code. Because data is also persisted locally, when the app goes offline, the previously loaded question threads will still display. As soon as network connectivity is reestablished on the client, the database will synchronize the local data with the current data.

Because the Realtime Database is a NoSQL database instead of the standard relational database, the database is designed for speed, performance, and scalability. However, unlike relational databases, the design of the NoSQL database leads to data redundancies, expensive storage of unstructured data, and only eventual consistency, which means that a piece of data will eventually be the same for all clients but it may not be the same at any given moment.

The Realtime Database stores data as JSON. Q&A's database schema will have five objects: Threads, GeoFire, Posts, and Users, and FcmTokens. "Threads" contain the metadata about each question thread including a unique ID, question title, number of likes, and timestamp. "GeoFire" contains the location data of a question thread based on where the user was located when he or she posted the question. "Posts" contain the posts of the question thread including the main question and all of the comments and answers in the thread. "Users" contain basic data about each user. "FcmTokens" contain a list of Firebase Cloud Messaging tokens which are generated to send notifications to devices. The following is an example of the JSON tree structure, with capitalized constant values:

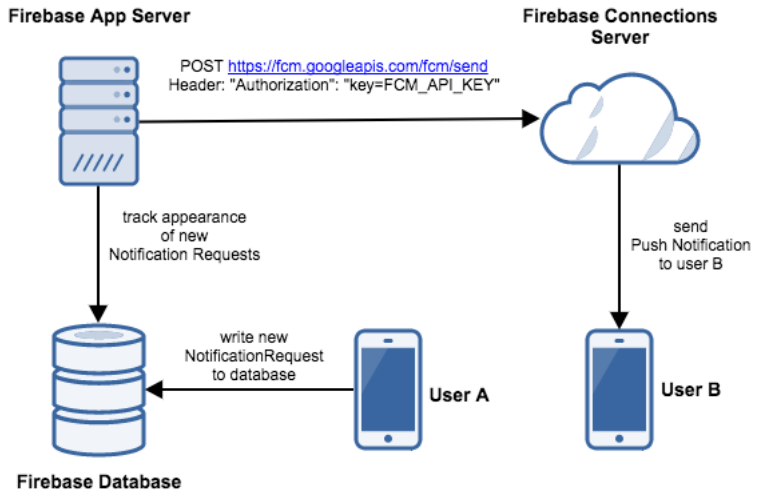
```
{
  "threads": {
    "one": {
      "dateCreated":
          date: "date",
      "dateLastChanged":
          date: "date",
      "likes":
        "user1": true,
      "likesCount: 1,
      "title": "QUESTION_TITLE_1",
      "userId": "user1",
      "users":
          0: "user1",
    }
  },
  "geofire": {
    "one": {
      "g": "GEOFIRE_GEOHASH",
```

```
    "l": {
      "0": "LATITUDE",
      "1": "LONGITUDE"
    }
  }
},
"posts": {
  "one": {
    "p1": {
      "dateCreated":
         date: "date",
      "dateLastChanged":
         date: "date",
      "likes":
        "user1": true,
      "likesCount: 1,
      "title": "QUESTION_TITLE_1",
      "userId": "user1"
    },
    "p2": {
        "dateCreated":
           date: "date"
        "dateLastChanged":
           date: "date"
        "likes":
          "user2": true
        "likesCount: 1
        "title": "Answer to QUESTION_TITLE_1",
        "userId": "user2"
    }
  }
},
"users": {
  "user1": {
    created_threads:
      0: "one"
    fcm_tokens:
      0: "token_1"
  },
  "user2": {
    fcm_tokens:
      0: "token_2"
  }
}
"fcm_tokens": {
```

```
    "token_1": "user1",
    "token_2": "user2",
  }
}
```

3.5 Notifications

Push notifications will be sent to the user when a user participates in a question thread and there is some new activity in the thread. The notifications will be sent using a Firebase Cloud Function which triggers on writes to the Realtime Database. This function will use the Firebase Admin Node.js SDK to send a message request to the Firebase Cloud Messaging (FCM) servers, which are provided by Google. The servers will then send messages in the form of a notification (using HTTP) to the client apps running on users' devices. Each user's device has a generated FCM token which is used by the FCM servers to identify various devices and send the messages to the appropriate devices. This notification flow is reflected in the following diagram:



4 References

[1] Google.: Firebase Documentation. firebase.google.com/docs/. (2017)

[2] Google.: GeoFire for Java - Realtime location queries with Firebase. github.com/firebase/GeoFire-java/. (2017)

[3] Udacity.: Firebase in a Weekend: Android by Google. www.udacity.com/course/firebase-in-a-weekend-by-google-android–ud0352/. (2016)

[4] Caio Ariede.:React Native: Developing an app similar to Uber in JavaScript. https://www.slideshare.net/caio.ariede/react-native-developing-an-app-similar-to-uber-in-javascript/. (2017)

[5] Ihor Vitruk.: Firebase Cloud Messaging for Push Notifications. TechMagic, http://blog.techmagic.co/firebase-cloud-messaging-for-push-notifications/. (2016)