Got It Diabetes Management: Final Submission

This document has been slightly updated with some changes in the road in comparison to the one delivered in the Mid-Point submission.

Basic Requirements

1. *Apps must support multiple users via individual user accounts*

Android Application

There are two different types of user to be logged in Android application: *Teen* and *Follower.* Both of them can be register via Register activity. The way to differentiate one type from another is by checking a box in such activity that opens new fields for the user to provide its medical number and date of birth. Providing these two fields will make this user a *Teen*, while not providing them will make it a *Follower.* The user can be registered via Web Server or Facebook account. For this second case, the user still need to provide its medical number if it wants to be register as a *Teen.*

Any user of the system will be unique identified by the provided e-mail account into Register activity. This information will be used, along with a defined password, to log into Android application. Obviously, if user has been registered via Facebook, its e-mail and password will not be required for the login.

Web Server

It is still possible to log into the Web Server via a HTML page. This access is restricted for the system *Administrator,* which is pre-created in server database and can be considered as a third type of user. There is no way to create another user of *Administrator* type.

This user has privileged access since it is the only one who can dynamically create new *Question*s that is sent to the *Teens.* Moreover, this user can push message notifications to any *Teen* or *Follower* registered in the application*.*

2. *App contains at least one user facing function available only to authenticated users*

Android Application

Only authenticated users can, indeed, use the application, e.g. see list of *Teens,* retrieve list latest *Check-Ins* and so on. This will be done by a control flag persisted in a Shared Preference that will handle the type of user logged in.

Web Server

Basically there are two screens/functions in Web Server that is only available to authenticated user of *Administrator* type: register a new *Question* (or even edit or delete them) and send broadcast messages to a registered device via GCM.

3. *App comprises at least 1 instance of each of at least 2 of the following 4 fundamental Android components*

- **Activity**

Some of activities created along with their basic functionality:

LoginActivity: Default launcher activity of the application. Provides a screen to user authenticate in the application, either via application's server or Facebook.

RegisterActivity**:** Provides an UI for the user to insert necessary data to be registered in application's server. This information could be gotten from Facebook login as well (except from medical number, in case of registering a *Teen*).

MainActivity**:** Provides the main screen of the application after the user is logged in. This activity includes a Navigation Drawer that is used to navigate between screens like: list of *Check-Ins*, list of *Teens* and app's Settings. Such screens are all fragments which are inflated into the MainActivity.

- **BroadcastReceiver**

TeenAlarmReceiver**:** This receiver will manage Intents sent by Alarm Manager Service when it is time for the *Teen* to ask some question about its health.

- **Service**

MyGcmListenerService**:** It is a service that will handle when some push notification via GCM (Google Cloud Messaging) has arrived in the device.

- **ContentProvider**

My system has been designed to store almost all information into server's database. This way, I avoid the user to be able to delete important data via Application Manager in Settings. Moreover, this design will give the possibility to extend the Android application functionality to a web client besides saving space into user's device.

However, the unique things I save into Android database is information about pending questions (as the *Teen* may want to answer them another time) and some information about the user logged in, like its e-mail and type (*Teen* or *Follower*).

4. *App interacts with at least one remotely-hosted Java Spring-based service*

The communication between the Android application and the Spring-based service is made by using Spring for Android library. The data is exchanged via Json with the auxiliary of Jackson library in both ends.

5. *App interacts over the network via HTTP/HTTPS.*

In order to setup a secure SSL connection with my server, the same has been configured to use a certificate issued by a self-signed certificate authority created by myself. As I am just running it on localhost, this should be enough to have secure communication between the application and server.

6. *App allows users to navigate between 3 or more user interface screens at runtime*

As described in requirement #3, there are some activities that the user, once authenticated with the server, is able to navigate through. This includes: screen to show the list of latest *Check-Ins,* screen to show list of *Teens* registered in the system, screen to show a chart of latest *Teen*'s *Feedback*, screen to set preferences in the application and so on.

7. *App uses at least one advanced capability or API from the following list (covered in the MoCCA Specialization): multimedia capture, multimedia playback, touch gestures, sensors, animation. Learners are welcome to use ADDITIONAL other advanced capabilities (e.g., BlueTooth, Wifi-Direct networking, push notifications, search), but must also use at least one from the MoCCA list.*

I have added a feature where the *Teen* can add some picture to complement its *Check-In* data. This photo is sent over to the server and retrieved when some *Follower* of this *Teen* requests its detailed *Check-In* data.

For additional advanced capabilities, I've been using push notifications with GCM to send follow requests to a specific *Teen*. So, whenever a user wants to start following a *Teen*, a message is sent over the network via GCM and it will reach such *Teen* that will confirm whether it authorizes or not to be followed by this user.

8. *App supports at least one operation that is performed off the UI Thread in one or more background Threads or Thread pool.*

An AsyncTask is fired whenever the Android application needs to fetch/post some data from/to the Web Server. While this data is being received/sent, a progress dialog is shown to inform the user that something is going on. For example, when the user is sending login information to the server authenticate it in the application, or when user is receiving any kind of data from the server, e.g. list of *Teens,* list of *Check-Ins* and so on.

Another logic that is done off the UI Thread is when the user receives a push notification via GCM or when the alarm to display a *Question* to a *Teen* is fired - in this case the Android application will, in a background Thread, request the Web Server for a new *Question* that will be displayed to the *Teen* (via notifications) when such *Question* is received.

Functional Description and App Requirement

1. *The Teen is the primary user of the mobile app and is represented in the app by a unit of data containing the core set of identifying information about a diabetic adolescent, including (but not necessarily limited to) a first name, a last name, a date of birth, and a medical record number.*

The *Teen* will be register via the Android application in the Register activity. Information such as e-mail (unique key for a *User* in the system), first name, last name and date of birth can be filled into the form *or* fetched from Facebook if user decides so. The medical record number should always be provided for registering a *Teen*.

2. *The Teen receives a Reminder in the form of alarms or notifications at patient-adjustable times at least three times per day.*

An alarm will be set by using Android's Alarm Manager Service. The frequency to be triggered is adjustable in application's settings screen and it can vary between 3, 4 or 6 times a day. Obviously this alarm only runs when the logged user is a *Teen* as we don't want to disturb a logged user which only monitors its *Teens*. When such alarm is triggered, the app reaches the server to get the latest list of questions. Once the list is received, the *Teen* is notified via notification that a new set of question has come and is waiting to be answered.

3. *Once the Teen acknowledges a Reminder, the app opens for a Check-In. A Check-In includes data associated with that Teen, a date, a time, and the user's responses to a set of Questions at that date and time.*

As explained in previous item, a notification comes when it is time for another *Check-In*. The *Teen* has the option to postpone it or to answer it at that moment. If it is decided to postpone, the *Teen* will be able to revisit this action later in a screen of the application which shows a list of pending *Check-Ins*. Otherwise, it will be prompt a wizard for the *Teen* to answer a set of *Questions* retrieved from server. Once this is done, the *Feedback* data is sent to the server and stored in database to be accessible by other *Teens* and/or *Followers*.

4. *A Teen is able to monitor their Feedback data that is updated at some appropriate interval (e.g., when a Check-In is completed, daily, weekly, or when requested by Followers). The Feedback data can be viewed graphically on the mobile device.*

The first page of the app when the user is logged in shows the list of latest *Teen's Check-In.* Important to note that a user only sees information about *Teens* who has previously authorized to be followed by such user. Such information will firstly be summarized displayed in a list. In order to get more detail of that *Check-In,* the *User* needs to click on the correspondent item of the list to open another activity displaying the data in a detailed way. Depending on the type of *Question,* it is possible to see it in a graphical way by clicking on the *TextView* related to that *Question.*

5. *<u>The app includes a Follower role that is a different type of user (e.g., a parent, clinician, friend, etc.) who does not the ability to perform Check-Ins, but who can receive Check-In data shared from one or more Teens. Also, the app allows a Teen to be a Follower for other Teens.</u>*

The *Follower* shall be created using the same Register activity used to create a *Teen.* In order to do that, the user should un-check the “I’m a Teen!” checkbox. Both *Teen* and *Follower* are derived from a base entity called *User* which contains shared information between those two types, such as e-mail, first name, password, Facebook id and so on. The *User* entity can be a *Teen (*then it has a one-to-one relation with *Teen* entity) or a *Follower (*then it has a one-to-one relation with *Follower* entity). In case of a *Teen* being also a *Follower*, the *User* representing it will also have the one-to-one relation with *Follower.* So this implies that this *User* is a *Teen* but also a *Follower* of other *Teens.*

6. *<u>The app allows a Teen to choose what part(s) of their data to share with one or more Followers.</u>*

The *Teen* can configure in application’s settings screen, which part of their *Check-In* data they would like to share. Once they are set, the information is sent to the server that updates the *Teen* table accordingly with that *Teen* preference. Such table will be queried every time a *Check-In* data from a *Teen* is sent to an Android application.

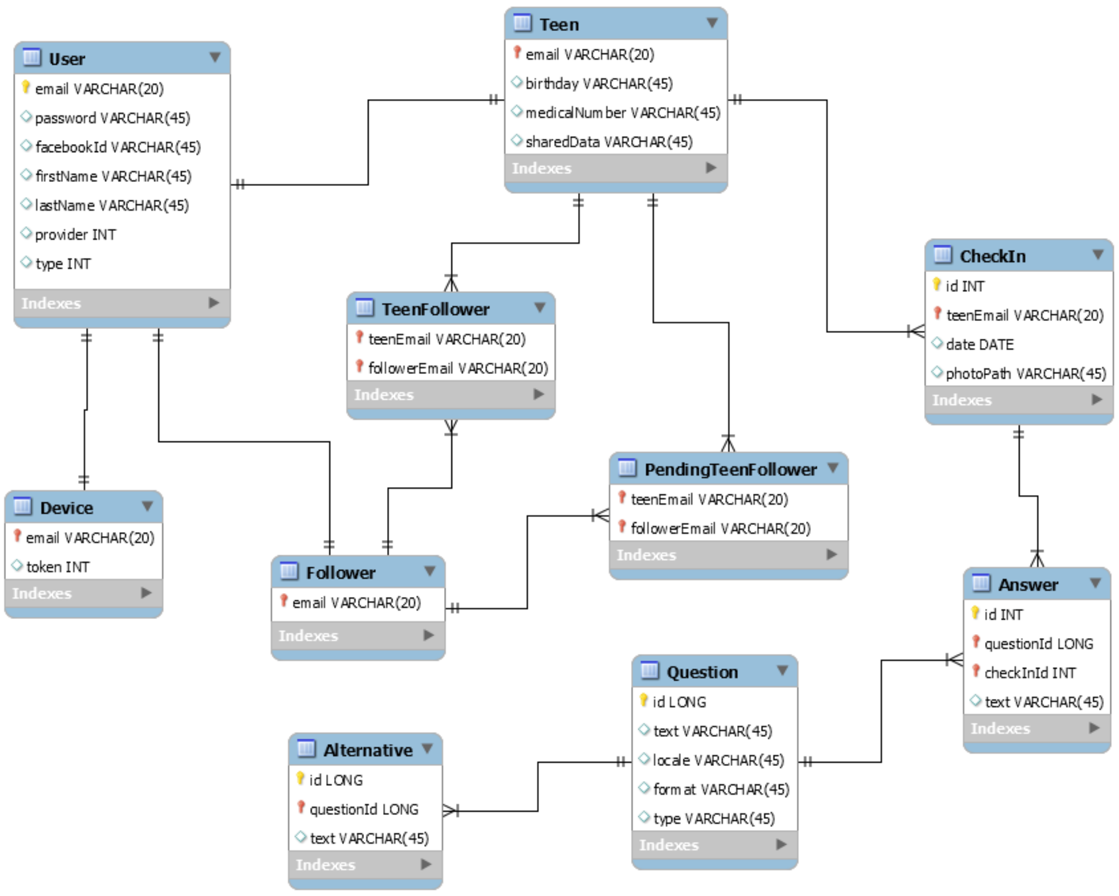
7. *<u>The app only allows Teen data to be disseminated to authorized/authenticated Followers and accessed over HTTPS to enhance privacy and security of the data.</u>*

For a *Follower* effectively start to receive latest *Teen Check-Ins*, the *Teen* must agree to be followed by such *Follower*. So it will not be any *Follower* or *Teen* who can view other’s *Teen Check-Ins* without the consensus of this *Teen*. In order to do that, a push notification will be sent to the *Teen* via GCM when some *User* wants to start following this *Teen.* This request will be on a pending requests table until the *Teen* accepts or rejects it. If it accepts, another requests table will be fulfilled with information of both *Teen’s* and *Follower’s* email.

As explained on item #5 of Basic Requirements, it has been set a SSL server with a certificate issued by a self-signed certificate authority created by myself, so all the traffic exchanged with the server will be encrypted and verified.

EER (Extended Entity–Relationship)

Here we can see the diagram representing the current relationship between the entities in database:



Teen
email VARCHAR(20)
birthday VARCHAR(45)
medicalNumber VARCHAR(45)
sharedData VARCHAR(45)
Indexes
User
email VARCHAR(20)
password VARCHAR(45)
facebookId VARCHAR(45)
firstName VARCHAR(45)
lastName VARCHAR(45)
provider INT
type INT
Indexes
CheckIn
id INT
teenEmail VARCHAR(20)
date DATE
photoPath VARCHAR(45)
Indexes
TeenFollower
teenEmail VARCHAR(20)
followerEmail VARCHAR(20)
Indexes
PendingTeenFollower
teenEmail VARCHAR(20)
followerEmail VARCHAR(20)
Indexes
Device
email VARCHAR(20)
token INT
Indexes
Follower
email VARCHAR(20)
Indexes
Answer
id INT
questionId LONG
checkInId INT
text VARCHAR(45)
Indexes
Question
id LONG
text VARCHAR(45)
locale VARCHAR(45)
format VARCHAR(45)
type VARCHAR(45)
Indexes
Alternative
id LONG
questionId LONG
text VARCHAR(45)
Indexes