Got It Diabetes Management: Requirements Mapping

This document contains information about the classes/lines of code where the requirements are met.

Basic Requirements

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

The login information is sent from Android application through an *AsyncTask* called PerformLoginTask. The server, in turn, validates the received data and sends back a response informing whether the login has succeeded or not. This logic from server side is found here.

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

The main activity of Android application is only available for authenticated users. The user (*Teen* or *Follower*) is redirected to such activity when their login has been successfully completed, as shown in this piece of code.

The LoginActivity is the default launcher activity of the application. So it is mandatory the user to be logged in before proceeding to the main activity.

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

- **Activity**

All my activities have been created in com.android.application.activity package and declared in AndroidManifest.xml.

- **BroadcastReceiver**

It has been created two *BroadcastReceiver* to manage the *Check-In* alarm.

TeenAlarmReceiver - It will receive an Intent every time the alarm is fired.

BootReceiver - It will receive an Intent when device is booted, so the *Check-In* alarm can be restarted.

- **Service**

It has been created two Services to manage GCM registration and messages receiving.

MyGcmListenerService - Receives push notification from GCM server and show them in the UI via notification.

RegistrationIntentService - Sends GCM token to server when user is first logged in the application.

- **ContentProvider**

PendingCheckInProvider - This provider simply saves information about the *Check-Ins* skipped by the *Teen*. All its contents will be shown in a dedicated screen of the application so the *Teen* never gets rid of not-answered *Check-Ins*.

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

My Spring-based service is located at:
https://github.com/femosso/CapstoneProject/tree/master/WebServer

Here is an example how I'm sending data (in this case, login information) from Android application to the Spring-based server and here is my Spring-based server receiving data from the Android application.

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

For this requirement, I configured my Apache Tomcat to always connect to port 8443 and provide a certificate, signed by a root CA that I created by myself.
In order to configure Apache Tomcat in this way, I added the following lines in *server.xml* file:

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"
    keystoreFile="${user.home}/keystore/keystore.jks" keystorePass="" />
```

This will use the server certificate and private key stored in *keystore.jks* to make HTTPS connection with the Android application and web page (for *Administrator* access). For this, I also had to add the self-signed root CA I created by myself to the trusted CA list of the Android device and Web browser, as per demonstrated in the screencast video.

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

It has been created several Android *Activities* and *Fragments*. All of them can be found on com.capstone.application.activity and com.capstone.application.fragment packages.

7. *App uses at least one advanced capability or API from the following list (covered in the MoCCA Specialization): multimedia captur.e, 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.*

The *Check-In* photo is taken by clicking on negative button (chosen as negative button because its position in the dialog) of a confirmation dialog that is shown after the *Check-In* Wizard is finished. The file is saved locally in the device before being sent to the server.
https://github.com/femosso/CapstoneProject/blob/master/MyApplication/app/src/main/java/com/capstone/application/activity/CheckInWizardActivity.java#L208

After sending it to the server by the SendCheckInTask, the file is deleted:
https://github.com/femosso/CapstoneProject/blob/master/MyApplication/app/src/main/java/com/capstone/application/activity/CheckInWizardActivity.java#L551

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

There are several *AsyncTasks* and *Services* in the application. Some *AsyncTasks* can be found in the following paths to send/retrieve some kind of data to/from server.

- **AsyncTask**

RetrievePendingFollowRequestTask - Reaches the server to get the list of pending follow request for the logged *Teen*.

RegisterAccountTask - Sends account registration data of a new *Teen* or *Follower* to server.

- **Service**

MyGcmListenerService - Receives push notification from GCM server and show them in the UI via notification.

RegistrationIntentService - Sends GCM token to server when user is first logged in the application.

- **BroadcastReceiver**

TeenAlarmReceiver - Receives an Intent when alarm for a new *Check-In* is fired.

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.*

All the information of a *Teen* should be filled in a form when the user is first register in the application. Here is the xml representing all these fields to identify 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.*

The *Teen* can choose the frequency of the reminder in a PreferenceScreen. The possible values are 3, 4 or 6 times a day, as defined here.

This preference will be read when the *Check-In* alarm is set for the first time, as we can see in this method.

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.*

When *Teen* clicks on *Check-In* notification, a wizard is opened and the *Teen* should answer all *Questions* related to this *Check-In*. This can be found here.

The *Check-In* object carries the required information (date, user's response to a set of *Question* and so on) as you can see in this link.

The *Questions* can be defined dynamically in the server once the *Administrator* is logged in. Here is the JSP page to register a new *Question* and here is the Controller in charge of saving the new created *Question* to database.

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 *Check-In* information of a *Teen* is shown in a separated activity called CheckInDetailsActivity. There are some types of *Question* that can be viewed graphically on the device by clicking on the *TextView* of such *Questions* in CheckInDetailsActivity. This will call RetrieveHistoricTask that will query the server to have the latest status of the determined *Check-In* information.

There are two types of charts:

- LineChartActivity - It shows information about blood glucose levels.
- PieChartActivity - It shows information about the state of a *Teen* and how they were feeling at the moment of the *Check-In*.

5. *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.*

There are several checks in the application to restrict access when the logged user is not a *Teen*. For instance, in MainActivity we are checking if the logged user is a *Teen* before setting up the *Check-In* alarm. If the logged user is not a *Teen*, there is no reason to set it up.

When creating a new *Teen,* we also set the *Follower* field since a *Teen* could be a *Follower* of other *Teens.*

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

The *Teen* can choose on PreferencesFragment what kind of data they would like to share. Once the preference is changed, the server is also updated.

The server considers this information when returning *Check-In* information to Android application as seen here.

7. *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.*

The logic of sending follow requests to a *Teen* is defined here. This information is sent to server which will notify the *Teen* that it has a new *Follower* wanting to follow him/her, as we can see here. The *Teen*, in turn, can accept or deny this follow request, as defined in FollowRequestListAdapter.

As explained in 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.