Got It Diabetes Management: Design Document

by a Coursera student

2015-11-18

1 User-Facing Components

In this section of the document you will see the app from a user's perspective, that is the screens and major features the user can use. The technical and implementation details will be described in the next section.

1.1 Authentication

A newly installed *Diabetes Management* app should display the authentication screen to the user. Depending on whether the user has an existing account he/she needs to either log in or sign up (see the figure 1).

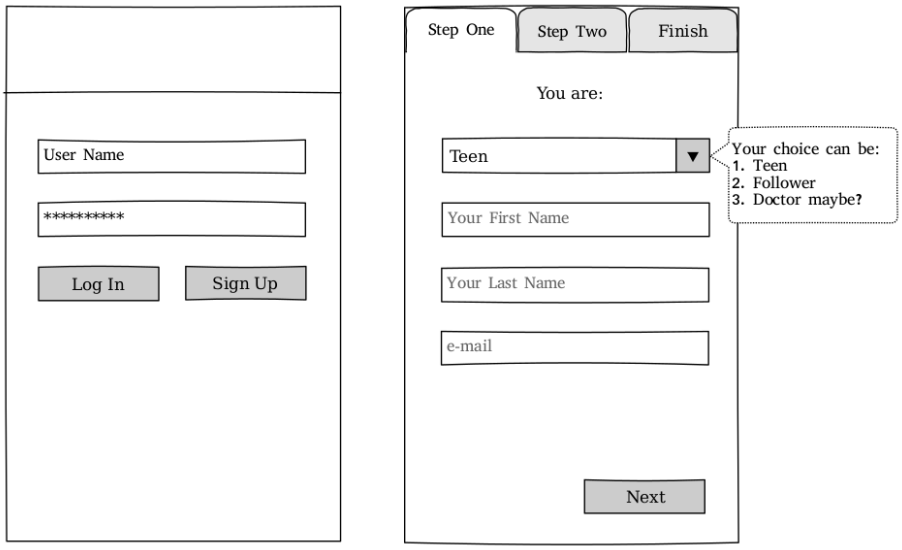

Figure 1: authentication screens

Log In

The user needs to enter their user name and password. After hitting on the *Log In* button the app will display the *Check-In* screen (see the figure 3).

Sign Up

The registration process includes several steps, each of which gathers various user's information. When any information is missing the user is prompted with a toast message showing the error. [1] Both account types require an e-mail address and other info.

Log Out

As you can see in the figure 2, the user can also log out if, for example, another person needs to use the app under their own account. [2]

[1] 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.

[2] Basic Project Requirement #1: App supports multiple users via individual user accounts

1.2 Main Menu

To give the users ability to easily navigate between all the screens, the application uses a navigation drawer, which is called *Main Menu*.

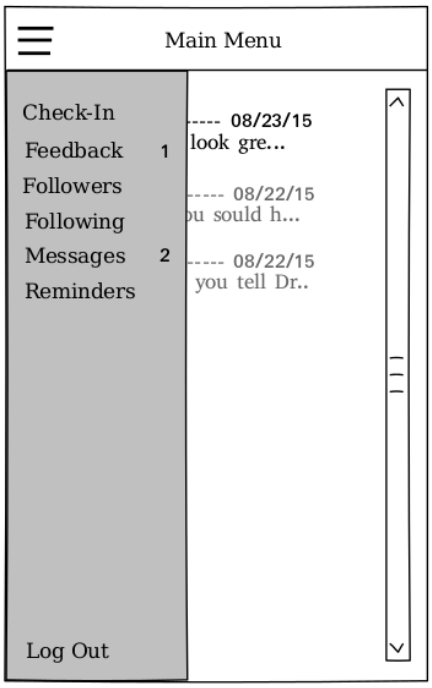

Figure 2: main menu

This menu and the screens, which can be navigated to from the *Main Menu*, are only available to authenticated users. [3]

[3]Basic Project Requirement #2: App contains at least one user facing function available only to authenticated users

1.3 Check-In

This is the main screen for gathering a teen's diabetes related information. It might be open when the user clicks on a check-in notification or be navigated to from the *Main Menu*. The figure 3 shows what information is required from a *Teen* user. [4]

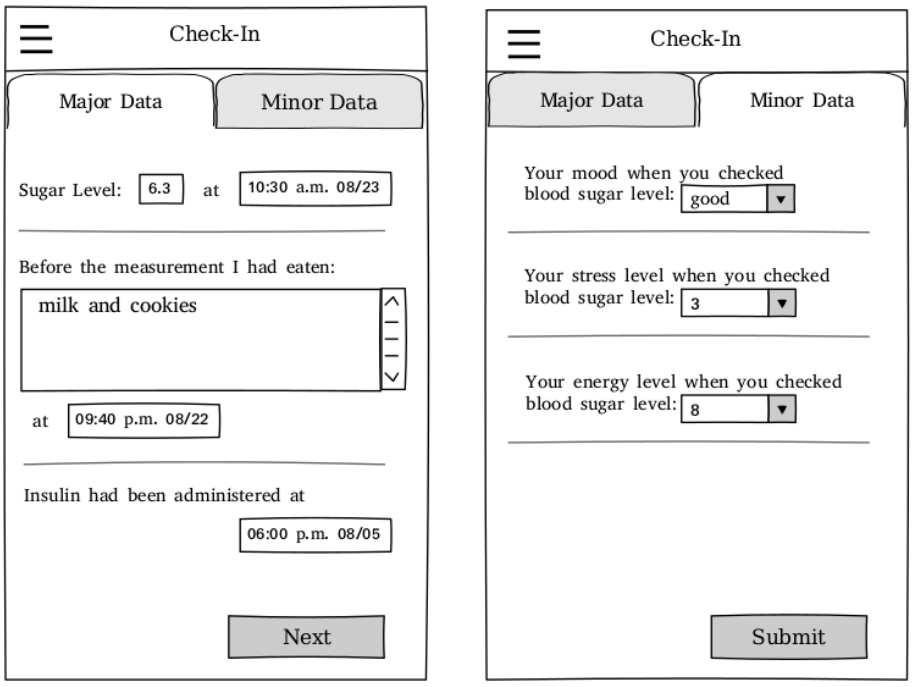

Figure 3: check-in screen

This screen is only available for the *Teen* users. The *Follower* users cannot preform check-in's. [5]

[4]Functional Description and App Requirement #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.

[5]Functional Description and App Requirement #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*.

1.4 Feedback

Both *Teen* and *Follower* accounts can receive feedback from the users they follow. This information is displayed on the *Feedback* screen (see the figure 4).

Either the *Followers* screen (figure 5) or the *Main Menu* (in case of a *Teen* account to see their own feedback data) can be used to navigate to the *Feedback* screen.

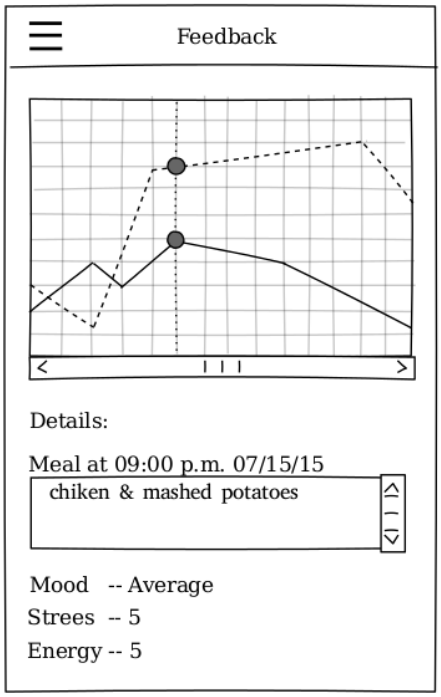

Figure 4: feedback screen

Whenever a *Teen* submits check-in data, his/her followers receives the data immediately in the *Feedback* screen. [6]

The amount of information depends on the permissions the corresponding *Teen* granted to their *Follower(s)* (see the figures 5 and 6). Additionally, touch gestures are used to zoom in and zoom out the feedback plot. [7]

[6] Functional Description and App Requirement #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.

[7] Basic Project Requirement #7: App uses at least one advanced capability or API from the following list: multimedia capture, multimedia playback, touch gestures, sensors, animation.

1.5 Followers

Each *Teen* account user can have followers. These are users that receive feedback from the *Teen*.

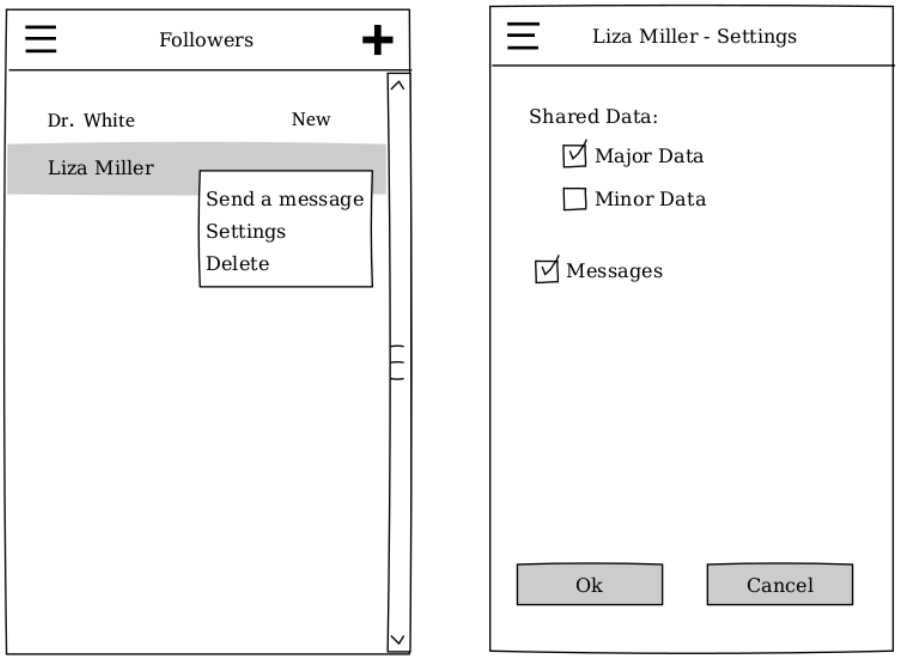

Figure 5: follower and settings screens

As you can see in the figure 5, a *Teen* can also change the amount of data he/she wants to share with a certain *Follower*. [8]

[8] Functional Description and App Requirement #6: The app allows a *Teen* to choose what part(s) of their data to share with one or more *Followers*.

In case if the *Teen* has a new follower request, he/she can click on the corresponding item in the list (figure 5) to open the *New Follow Request* screen. The *Teen* can either accept or reject the request (see the figure 6).

If the *Teen* wants to invite a *Follower*, the + button can be used to open the corresponding screen, where the *Teen* needs to enter the e-mail the person has used during the registration. [9] The person, whom the invite has been sent to, will receive a notification where he/she can accept/reject the invitation.

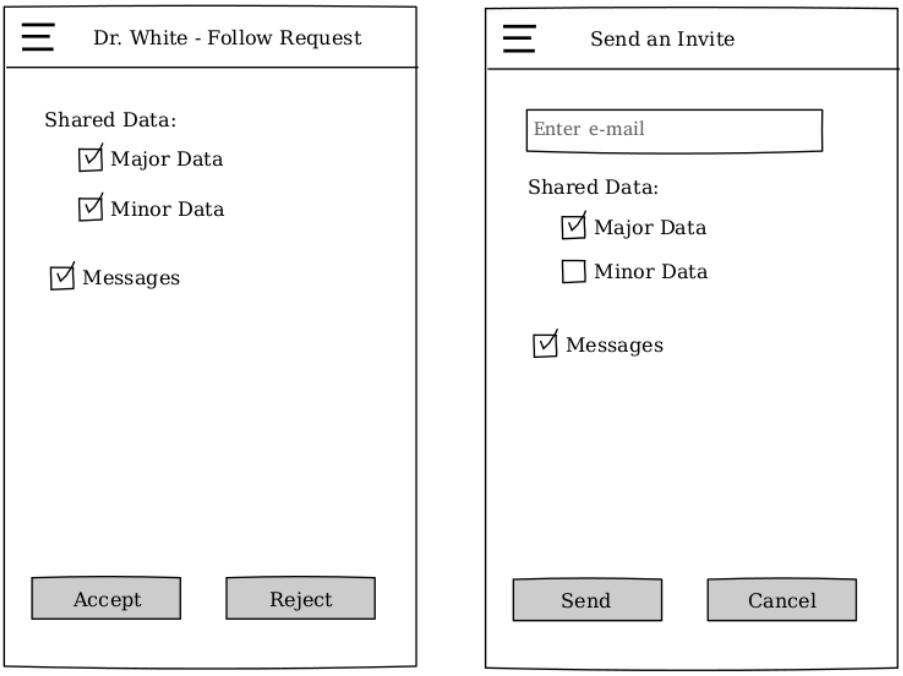

Figure 6: follow request and invitation screens

[9] Functional Description and App Requirement #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 HTTPS part will be described in the implementation details)

1.6 Following

To see the list of the users you currently follow the *Following* screen can be used. The user can access feedback of the users he/she follows or start to follow another one by clicking on the + button (see the figure 7).

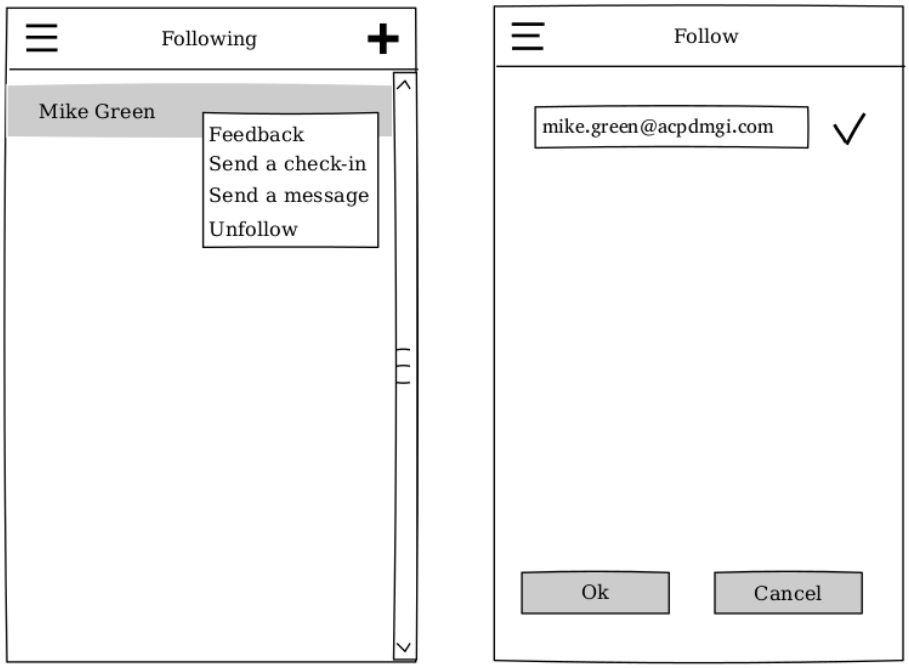

Figure 7: following screen

1.7 Reminders

Each *Teen* receives check-in reminders in the form of alarms at least three times a day. Once a reminder has been received a *Teen* can tap on the notification and the *Check-In* screen is open to perform the check-in process. Reminder can be added, edited or tuned on/off using the *Reminders* screen (see the figure 8). At least three reminders must always be on. [10]

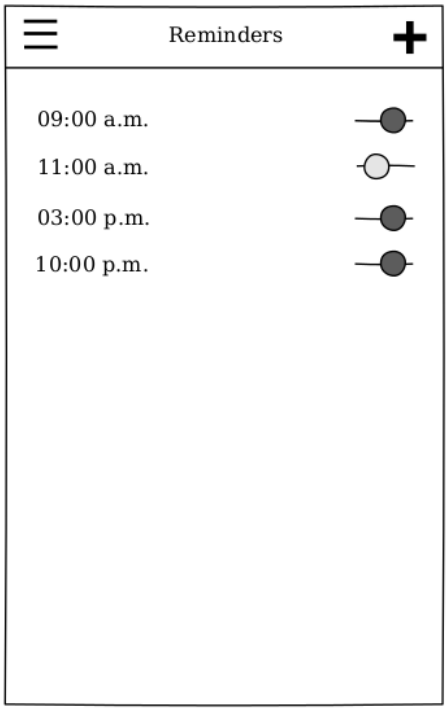

Figure 8: reminders screen

[10] Functional Description and App Requirement #2: The *Teen* receives a Reminder in the form of alarms or notifications at patient-adjustable times at least three times per day.

1.8 Optional Features (haven't been implemented)

The following features will be implemented only if the major capstone project requirements are met and there is development time left.

- Messages

  Teen account will be able to communicate with their *Followers* and vice versa by using the *Messages* screen. A *Teen* or their *Follower(s)* will be able to communicate in the form of a dialog (see the figure 9).

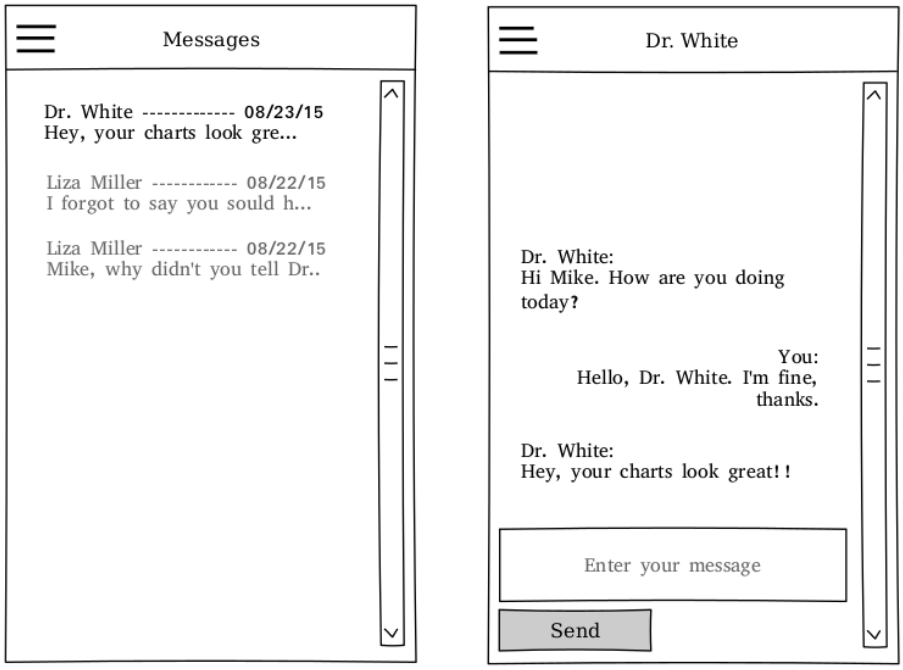

Figure 9: messages and dialog screens

- User Settings (the screen is not present)

This screen will allow to alter different user information and settings. For example, a user could change the password or turn some notifications off.

2 Implementation Details

This section describes the implementation details of the application's components at a very high level. There will be provided an example of use of some of those components to give you a picture of how they are going to interact (there will an outdated picture, the app has a sync adapter now).

2.1 Client and Server Side Implementation

Client Side

The client side is represented by an android-based application.

Server Side

The server side of the project is represented by a remotely-hosted Java Spring-based service. [11] For the sake of simplicity and due to the lack of time, no scaling capabilities is going to be used.

2.2 Security

Client-Server Communication

All network communication between the client and server side is done via HTTPS. [12] For test purposes the project uses a self-signed SSL certificate, which should be changed in production. The client side is distributed with the public-key certificate, which is stored inside a JKS keystore in resources/raw.

User Security

OAuth2 is used for the user authorization. All interactions with the client (except for the *Authentication* screen) and the server can be done only if an access token has been granted to the user. Once an access token has been received, it is saved into shared preferences on the client side, so that it can be used from any place in the application and the user could use the app when the network is off (for example, to work with the cached feedback).

User Credentials Security

The users passwords are not stored on the server side. Instead of the plain text format, hash value and salt is used to verify the users credentials.

2.3 Data Management

Client Side

The activities and fragments are stateless as possible and hold almost no data in order to facilitate configuration changes. A content provider backed by an SQL database is used to store cached server data, so that this data could be used even if the user is off-line.

Server Side

The server uses a separate MySQL database to store all the users' data.

[11] Basic Project Requirement #4: App interacts with at least one remotely-hosted Java Spring-based service

[12] Basic Project Requirement #5: App interacts over the network via HTTP/HTTPS

2.4 Network Operations

Client Side

All client side outgoing network operations are performed in a background thread via a single intent service. [13] The incoming ones are handled like this: once the GCM listener has received data from the GCM server, it hands it over to the sync adapter, the sync adapter handles it and downloads data from the server if needed (usually a GCM represents a table to be updated), after the data has been received, it is stored in the content provider. The app fragments and activities use cursor loaders, when the SQLite DB is updated, the UI is automatically updated by cursor loaders and cursor adapters. This separates logic and data makes it easy to handle configurations changes. [14]

Server Side

The server outgoing network operation are performed via a google cloud messaging server. The GCM server delivers any updates to the corresponding client by using push notifications. Depending on the type of the info from the server, a push notification contains either entire data or "command" (usually a table name) for the client side (for example, to fetch new feedback from the server).

2.5 Client GUI

As you could see in the "User-Facing Components" section, the client side GUI is represented by at least 10 screens. [15] Those screen are implemented using about 2-3 activities and 7-8 fragments. The use of fragments makes the GUI smoother and makes it easier to re-use pieces of the user interface.

[13] Basic Project Requirement #8: App supports at least one operation that is performed off the UI Thread in one or more background Threads or Thread pool.

[14] Basic Project Requirement #3: App comprises at least 1 instance of each of at least 2 of the following 4 fundamental Android components: Activity, BroadcastReceiver, Service, ContentProvider

[15] Basic Project Requirement #6: App allows users to navigate between 3 or more user interface screens at runtime

2.6 Example Components' Usage

This is an example of how the certain components are used when one user sends a follow request to another user. Just follow the arrows to see in what order the parts are being used during request.

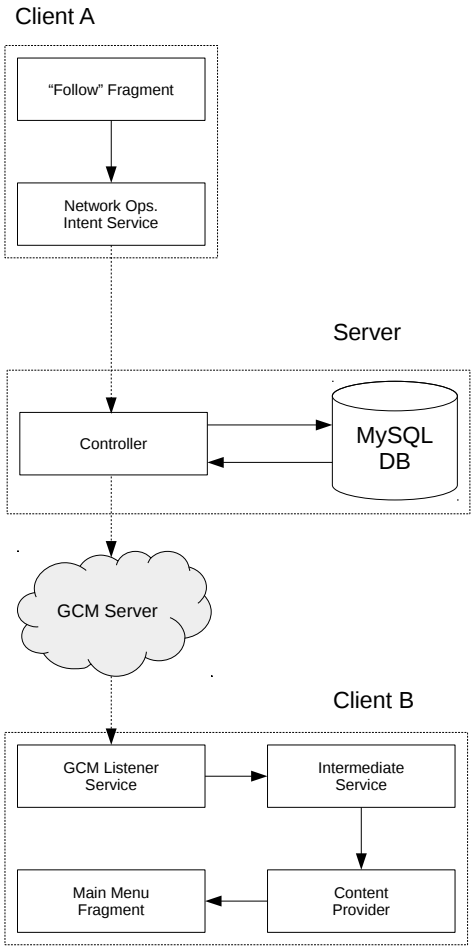

Figure 10: follow request workflow

Thank You!