Documentation by Judah Sistrunk on 7/29/16

CRM app

about:

This app is pretty much a fancy contacts list. It sounds lame, but what makes it not so lame is its ability to sync to a server, and thus the information can be used by other applications.

Structure:

Most of the functionality of the app is done by fragments that are imbedded into activities. Main activity spawns main fragment, which is where most of everything happens, so don’t go looking for much useful stuff from main activity.

The app pulls things from the server, but doesn’t store all the contact info until the user actually clicks on that contact in the list of contacts. This prevents the app from being overloaded by information it may not need to know.

Getting and storing information is a complex process. First there is the Kardia fetcher. This is the thing that actually connects to the server and gets the stuff the app needs. If has several specific functions that can get the specific things you need. It will return what you want as either an object or a list of objects depending on what you requested. Once the information gets to the app it is stored in the local database. The local database is complicated. First, there is the CRM contract. This class holds all the constants used by the database. Then there is the CRM open helper. This class creates the database when the app is first opened. The thing that does all the actual querying is the Kardia provider. This class is not supposed to be referenced directly. Instead, you have to use context.getContentProvider.(query, insert, update, or delete) in order to access the database.

In order to use the Kardia fetcher, you must create an async task. The fetcher cannot be used in the main thread. It will cause a main thread exception. You must use an async task. There should be a couple of examples of how this works in the code already. If you ever need to access kardia, use what has already been done as an example. It would be nice is someone later down the line finds a way to put the async task into a public class that can be used anywhere. This would be preferred over the way that it is currently being done. Rather than making a private class for everything, you could just use a public class that can handle everything you need. That way it can be more modular. The post and patch json classes are good examples of this.

Information is passed back and forth between fragments and views using intents and args. The database is not accessed much, so you need to pass things around a lot. This does make the code a bit messy and hard to deal with. It also makes adding columns to the database very difficult since you have to make new variables to pass around. If someone could make this better sometime down the line, it would be greatly appreciated. A function that gets or sets partner information from/to the database would be nice rather than setting up a new query every time you need something.

things to do:

profile pics

get the timeline to work without network access

things done:

get account information

get list of partners

timeline

click phone number to call

click email to send email

click address for google map

patch for edits

way to choose protocol and port number

sign out

add contacts

Dumb things the app likes to do:

Sometimes the app will open with no information in it. When this happens, you have to uninstall and reinstall the app. This is a pain. :( As far as we can tell, this has something to do with not having network access.

Sometimes the app will forget contacts.

Sometimes the app will display blank information if there is no network access.

Plead:

Please comment everything you do. There has been a whole lot of wasted time working on this app due to poor documentation and lack of comments. Please be considerate of the people that will work on this after you. Comment whatever you add or change. It will be greatly appreciated. : )

-Judah Sistrunk

**Talk with Greg – 7/24/2017**

Interaction

Calls from intents from app. Record later within interface.

Person Search API still not working properly. (Waiting on Greg)

Trust Data Cache for short period of time. (5-10 minutes) configurable by user.

No SyncAdapter.

EngagementTracks

New engagement.

Modify Label per Engagement.

Comments per step

Move up a step.

Bugs

Future todos

Overview of what the app does

Progress Bars

**CRM App Documentation**

Created by Daniel Garcia on 10/Aug/2017

**Overview:**

The app’s main purpose is to allow a user to keep track of their connections all in one place. The app keeps track of the user’s collaborators, showing their contact info, recent activity, and other pertinent information. The app also allows the user to quickly add new collaborators, record interactions with them, and update any information regarding a particular collaborator.

**Important Server Connection Information:**

There are 6 classes whose use should be noted;

KardiaFetcher: this class uses the Kardia API to get information from the server

KardiaProvider: this class uses SQL commands to get information from the app’s local database

CRMContract: this keeps all the constants used in the local database (which is all kept in SQL tables)

CRMOpenHelper: this simply creates the local database and all its tables

PostJson: this class uses OKHttp to connect to the Kardia server and takes care of uploading any new information

PatchJson: much of this class is similar to PostJson, except it updates or patches existing information rather than posting new things

**Features:**

Profile Display:

This activity displays a collaborator’s profile picture, contact info, and timeline (which includes any recent activity including interactions, prayer requests, etc.).

New Interactions:

After any phone call made or email sent from the app, the user is asked whether or not to record that interaction. An interaction can also be manually added. Interactions keep track of the type of interaction (phone call, in-person, update, etc.), date and time of interaction, subject, and any relevant notes. Optionally, a follow-up can be scheduled (along with a brief note) that will create a notification for the user to remember to follow up on their interaction with the collaborator.

Person Search:

TO\_BE\_ADDED\_BY\_TIM

Engagement Tracks:

TO\_BE\_ADDED\_BY\_TIM

Future To-Dos: