Real-Time Platform for Monitoring and Tracking Mobile Devices

Report

Siim Plangi
Institute of Computer Science
University of Tartu
Tartu, Estonia
splangi@ut.ee

***Abstract* – Smartphones are ideal tools for data collection and many researchers are using the opportunity for different projects. Unfortunately building reliable data collecting applications requires effort and time. Mob-Collector is a platform that reduces the effort building data gathering applications. Only a part of application needs to be implemented to get an application that is ready to be distributed.**

Keywords - Data, Collection, Mob-Collector

I. INTRODUCTION

Data is the base of many scientific researches. Without it many of the researches could not be done or validated. Thus data must be collected and preprocessed. With the rising of smartphones, an opportunity for scientists has opened up. Smartphones have various kinds of sensors that can be useful for many researches. As a result, there is a need for a flexible software that can allow handling and collecting mobility data without creating different data collection application for each project. This is the problem which is addressed in this paper.

This paper introduces an Android mobile application designed specifically to make data collection easier for anyone wishing to gather it from smartphones. The application is made as modular as possible to plug in different data gatherers, authentication managers and storage managers. Examples of each of these features are also provided in this paper. Only small parts of application must be rewritten to get the necessary data gathering functionality.

II. DATA COLLECTING FROM SMARTPHONES

Smartphones have a lot of sensors which can provide a lot of useful data for various kinds of researches. Most common sensors that most Smartphones have are [1]:

- Camera – possibility to take images and video
- GPS – detect the location of device
- Accelerometer – measure the acceleration of the device
- Gyroscope – get the device angle
- Magnetometer [2] – sensor to detect the direction of magnetic field at a point in space
- Proximity sensor – can be used to detect if an object is close to the sensor
- Light sensor – can be used to detect the lighting conditions of surroundings.

The sensors vary across devices and thus only the most common sensors were listed above. However data does not have to come from sensors at all. Other kind of data from smartphones can be useful as well. For example:

- Connected network info – Connected Wi-Fi devices and Cell Tower info can give a very good indication of user's current location.
- CPU load or Battery level – Combined with time information and possibly other sensor data this can indicate for example the usage hours of smartphone

These again are just a few examples of data that can be collected from smartphones.

When scientists use smartphone sensor or other data for research they usually have to build custom collector application which to distribute to the users. This can be quite troublesome and can take time. Additionally, when there are many voluntary users in research then the UI must be user friendly. This is not an easy task to achieve and will take additional time. Mob-Collector is an application that has the platform for the UI, background services and base classes for gathering various kinds of smartphone data. This should reduce the time for building various kinds of data collector software.

III. MOB-COLLECTOR

Mob-Collector is an Android application that can help gather data from smartphones easier. The UI, background services and other necessary classes are already implemented. All what needs to implemented are a few Java interfaces and plug those objects into the platform. This should make data collection much easier, since most of the application is already built. There are also some ready-built objects for most common use cases like LocationGatherer or FirebaseDatabaseManager. First gathers location data and the latter sends data to a Firebase database. In the following paragraphs there will be deeper explanations regarding what must be implemented and what can be adjusted in the platform to achieve desired data collection functionality. There will be three different sub-paragraphs for Model, Controller and User-Interface.

A. *Models*

Mob-Collector has three main models which the controllers can use. All of these classes are presented as interfaces but there are simple implementations of each in case no special requirements exist for Controller implementations. The three models are listed below with figures [1, 2, 3] showing the structure of those models.

1) User

The user object is a simple class used to hold data about the smartphone user to identify the owner of data.

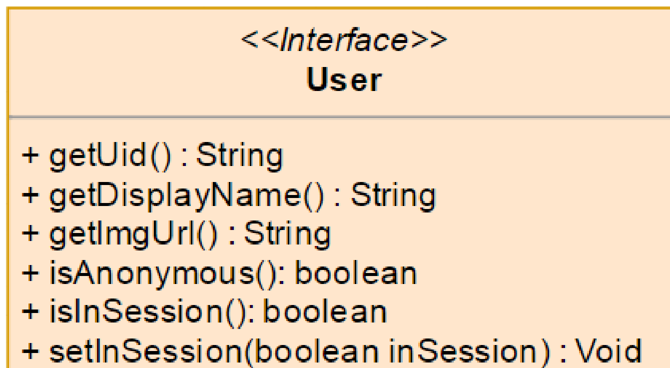

Figure 1 User interface

2) Session

The session object holds data for the current data recording. The session object is created when user starts the data collection and it is made inactive when the user stops the collection. It also holds the session's owners id for various use cases.

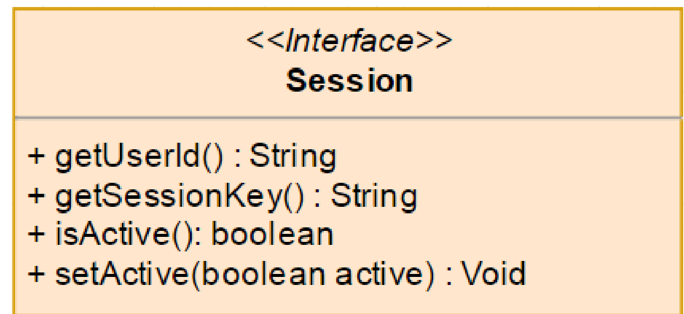

Figure 2 Session interface

3) Datacollection

The DataCollection object holds one type of collected data (e.g. Location data or Gyroscope data) from the session. As each session can have multiple data collections, then this is necessary to separate the two data groups. The Type enum represents the type of data collected, which is also editable.

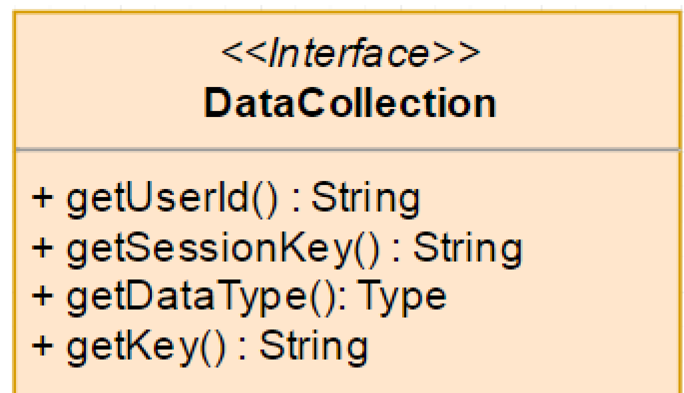

Figure 3 DataCollection inteface

B. *Controllers*

There are a total of 3 controllers which need to be implemented interfaces that needs to be implemented to get the desired functionality

- Gatherer abstract class – Base class for data gathering class implementation. Main method, which must be implemented, returns a Observable object. This Observable object must emit the collectable data objects.
- DatabaseManager interface – When implemented this class will be responsible for storing the collected data. The application will use its public methods to input data

- AuthManager interface – When implemented this class will be responsible to identify the user.

In the following sub-paragraphs each interface will be discussed in length.

1) Gatherer abstract class

The gatherer class is responsible for gathering data. The following methods must be implemented to create a valid gatherer object:

- *String getPreferenceRef()* – the Reference constant string in preferences which is used to determine if the user has enabled or disabled this collection method
- *Set<String> getPermissions()* – returns the set of permissions this gatherer needs to function properly
- *DataCollection.Type getDataCollectionType()* - returns a enum which kind of data is collected. Different data must have different type.
- *Observable<T> getObservable()* - returns an observable object which starts emitting the objects which must be collected. Example would be a *LocationObservable<Location>* object which would emit Location data. This data can be anything and the implementation is up to the user

<<*abstract class*>> **Gatherer**
+ abstract getPreferenceRef(): String + abstract getPermissions(): Set<String> + abstract getDataCollectionType(): Type + abstract getInfoGatheringObservable(): Observable<T extends TypeInfo>

Figure 4 Gatherer abstract class

When a session is initiated, firstly the Gatherer object checks if it is enabled by the user. When it's enabled and has the necessary permissions it starts observing the supplied observable which will start emitting the collectable data. The gatherer object then passes the collected objects to the DatabaseManager, which is responsible for storing the data.

2) DatabaseManager interface

The DatabaseManager gets a data input from Gatherer class and is responsible of storing it. It is also responsible storing and retrieving the User object which AuthManager provides. Examples implementations of it would be LocalSQLDatabaseManager which would store the data into phones local storage or FirebaseDatabaseManager, which would store the data into Firebase database [3]. Below are the methods that are required to be implemented for a custom DatabaseManager solution.

- *void updateUser(User user)* – update the User object in the database
- *getUser()* – Retrive the user from storage for AuthManager.
- *Session createNewSession(User user)* – create a new Session reference (aka Unique ID in SQL base or Value key in NoSQL database) which will be used to save the collected data
- *DataCollection createNewDataCollection(Session session, Type type)* – create a new DataCollection reference.
- *uploadInfo(DataCollection dataCollection, Info writable)* – Store the info with the details in the dataCollection object. As Gatherers might share one databaseManager then passing the dataCollection reference is necessary to make sure that the data is saved at the correct place with correct details.
- *updateSession(Session session)* - store or update the Session data in the database (e.g. set the inactive flag in the database). This is especially useful for live databases (e.g. Firebase), which can pass the active/inactive sessions straight to anyone who might be interested.

<<*Interface*>> **DatabaseManager**
+ updateUser(User user): void + getUser(): User + createNewSession(User user): Session + createNewDataCollection(Session session, Type type): DataCollection + uploadInfo(DataCollection dataCollection, Info writable): Void + updateSession(Session session): Void

Figure 5 DatabaseManager interface

3) AuthManager interface

The Authmanager is responsible for identifying the user. If the project does not need authentication the implementation of AuthenticationManager can just be an object which always returns an Anonymous user in logged in state. Then the user is not identified and no implementation is needed. Although when the user identification is necessary a bit more complex implementation is needed with a custom login screen implementation. Again FirebaseAuthManager has been provided along with source code. The methods that have to be implemented are below

- *void signOut()* – Sign out with the user and clear current user Data
- *User getCurrentUser()* – Get the current logged in user object or null if there is no logged in user
- *boolean isLoggedIn()* – Test if there is a logged in user or not
- *Intent getLoginScreenIntent(Context context)* - If there is logging in functionality, return the Log-In screen intent. Intent is method of starting activities in Android ecosystem.
- *void signIn(Callback<AuthResult> resultCallback)* – Start User signing in process and return the result of Authentication in an callback manner.
- *void signInAnonymously(Callback<AuthResult> resultCallback)* – Same as previous, but starts an anonymous sign-in process.

<<Interface>>
AuthManager

+ signOut(): void
+ getCurrentUser(): User
+ isLoggedIn(): boolean
+ getLoginScreenIntent(Context context): Intent
+ signIn(Callback<AuthResult> resultCallback): Void
+ signInAnonymously(Callback<AuthResult> resultCallback): Void

Figure 6 Authmanager Interface

C. *User Interface*

The user interface requires the least modification to get a recording Application. There are in total of 4 different UI screen, which are: Main page, App drawer, Settings page and Login page. In the following sub-paragraphs each of these is

1) *Main page*

The Main page is made as simple as possible and no modification should be nessessary. It displays a Map with the user current location and a button for recording. This button is used for starting and stopping of data recording [Figure 6]. Once the button is clicked a check is made if the user is authenticated or not. If the user is not authenticated a dialog is shows where the user has the option the log in anonymously or go to the sign-in page.

Figure 7 Main page

2) *App Drawer*

The app drawer [Figure 7] displays the log-in status of user with the provided name and optional image. There are two actions that can be performed from the App Drawer, one being going to the settings page and another one logs out the user.

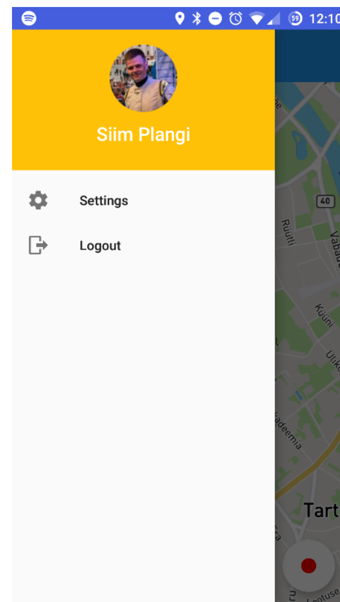

Figure 8 App Drawer

3) *Settings page*

The settings page allows the smartphone user to set up the preferred collection settings. The user can specify which gathering methods are allowed and which are not.

To add a gathering option to the Setting list, the menu.xml in the source code must be altered with the correct inputs and references.

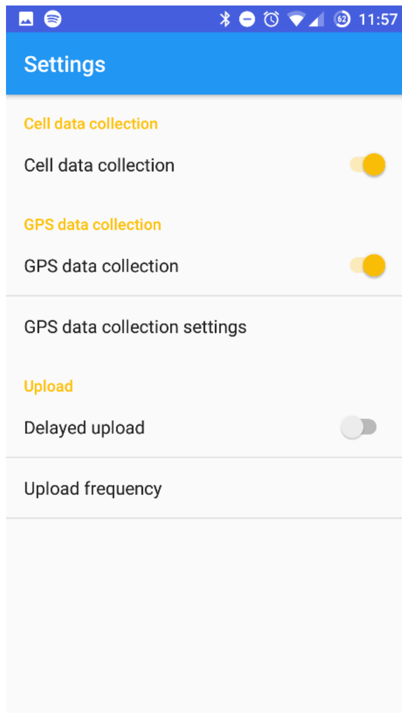

Figure 9 Settings page

4) Sign-In page

The sign in page is an optional page and can be avoided if authentication is not necessary. Although if authentication is needed then the Authentication page must be implemented for the AuthManager used. Below is an Authentication page using the Firebase Authentication [Figure 9]. This page is part of Firebase Android UI [4] and not part of Mob-Collector.

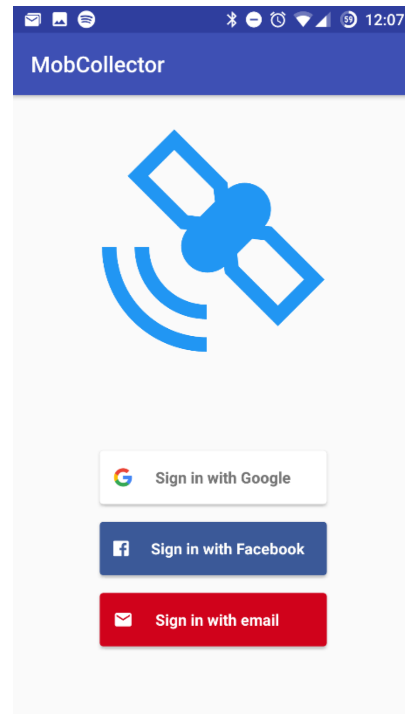

Figure 10 Firebase Log in page

IV. LIMITATIONS AND FUTURE WORK

There are some limitations to the platform, which should be addressed. Firstly, there is the need to modify the source code when a custom solution is built. This is problematic, because when the source code of Mob-Collector is updated there updates are hard to merge into custom products. This problem could be solved with a separate wiring class where all the components are wired up. Secondly, the Main UI is map based which is not ideal in many cases as not all data gatherers are location based. A solution to this would be that each gatherer object implements a View which displays relevant info or feedback to the smartphone user and those views are shown in the Main UI. Furthermore, more base implementations for most common use cases should be created which would speed up the custom data gatherer app development.

V. CONCLUSIONS

Mob-Collector is an Android platform that will reduce the effort needed to build a data collecting application. It does so by having a ready built UI with background services and other base classes which are necessary for a data-collecting application. Only a part of application must be implemented to get an application that can be distributed within smartphone users.

VI. REFERENCES

[1] Phonearena, "Did you know how many different kinds of sensors go inside a smartphone," 06 July 2014. [Online]. Available: http://www.phonearena.com/news/Did-you-know-how-many-different-kinds-of-sensors-go-inside-a-smartphone_id57885. [Accessed 2016 November 13].

[2] Wikipedia, "Magnetometer," [Online]. Available: https://en.wikipedia.org/wiki/Magnetometer. [Accessed 13 11 2016].

[3] Google, "Firebase," [Online]. Available: http://firebase.google.com/. [Accessed 14 11 2014].

[4] Google, "Firebase-UI," [Online]. Available: https://github.com/firebase/FirebaseUI-Android. [Accessed 14 October 2016].