Sweetie: a One to One app for sharing everything

Federico Ghirardelli, *Computer Science Master Degree, federico.ghirardelli@studenti.unipd.it,*
Eduard Bicego, *Computer Science Master Degree, eduard.bicego@studenti.unipd.it,*

***Abstract*—This paper shows the project of an extended One to One chat app designed and built as didactic project for the course Mobile Programming in Master Degree in Computer Science of University of Padua. The app is a basic app that offers to the user the possibility to create a couple, communicate with the partner through a chat and geolocalized gift over the possibility to share multiple galleries managed by the couple and multiple to-do-lists. The main aspects of the project was to give to us a point of view in how to program in Android and what are the main difficulties in programming into a mobile environment. Secondary we want to build a prototype of an idea born and developed as a business project in the course "Gestione di imprese informatiche". The app described in this paper will be used as product in order to show the idea to potential investors.**

***Index Terms*—android, social, chat, didactic project**

I. Introduction

We design and develop Sweetie for create a first prototype in order to show a minimal product born from the idea to potential investors. Once the users are coupled, they could share almost everything, text messages, images, photos, to-do lists send localized message. The main features built are:

- Multiple chats based on the today's standard of messaging apps leader, WhatsApp and Telegram;
- Galleries that are containers of images grouped by both partners;
- Infos of Chat, Gallery and To-Do list, accessible from the menu of each, show the date of creation and, optional, the photo cover. Only for the galleries there is the optional placement, selectable through a Location Picker UI;
- To-Do lists managed by both, for any kind of topic, like grocery list, movies to watch, trips to plan;
- Geogift, are messages like a post-it, pictures or emoticons, geolocalized. The partner won't know about the gift until he/she will come exactly on that physical place e.g., a specific street address or store in the city. This thanks to the geofence, a virtual perimeter set on a real geographic area that should be monitored by a service. The simplest geofence to be created needs of center position, explicated by latitude and longitude, and the radius of circle area, setted in meters. When the boundary of geofence is crossed, the user is alerted by a push notification;
- Maps to see, as a markers, the Geogifts sent or discovered and the galleries that have a photo cover and a location specified;
- Calendar where to easily visualize the own saved message (bookmarked).

For that we don't put so much effort into usability and accessibility of an Android app, for example our app lacks of a initial tutorial or a welcome page where the user is instructed with what he could do with Sweetie furthermore some important feature: image compression and a complete notifications service were not implemented, we preferred to invest the last works time to test our application in order to fix underhand bugs.

The paper is organized as follows: section II describes how we analysis the domain of Sweetie and what are the entities that we manipulated for create the core business logic of the application. Section III expose the app architecture, with merits and defects, and why we have done some choices. Section IV show the list of external library that are used in order to support some feature of Sweetie. In section V we group all the major problematics that requires more effort and we explain the solution or workaround that we implements with some accepted trade-offs. Section VI show some important feature that our app has not yet implemented. Finally the section VII describes our considerations on the development of this project.

II. App domain

In order to understand more the domain and study in deep the requirements we tried to use the Domain Driven Development (DDD) technique described by Eric J. Evans [1]. We start with a domain described with Actions that the user could do in the Home screen from the main fab button () . The mains object extends the class Action, they are Chat, ToDoList, Gallery and Geogift. Chat, To-Do list and Gallery have a relation 1 to N with Message, CheckEntry and Photo respectively. The Chat also have a relation 1 to N with ChatDiary that are an entity that group all saved messages from one chat in a precise date, calendar feature use it.

We use only one type of Message and of Photo because this ensure a more simple interface with firebase database in the GET and POST operation. Despite of the beneficial of use DDD for understand the requirements and for build a vocabulary of the domain the initial model in figure () doesn't reflect the last implementation of the application and the DDD technique was abandoned due to our lack of discipline.

III. App architecture

Sweetie could be a long terms project with new extensions in the future so we concentrate a lot of effort to take and maintain a certain architecture in order to keep maintainability and extendability of the app.

Today in the Android community Model View Presenter architecture pattern is the major suggest architecture pattern although the MVP is not a complete architecture pattern as Uncle Bob said in numerous of his talks [2]. By the way we would like to clarify that the official Android designer Dianne Hackborn [3] explicitly tell that Android components don't

give a precise architecture bounds, Android components by design could be use in very different architecture styles.

In fact the MVP in Android is an element of the Clean Architecture explained very well by his inventor Uncle Bob [4]. You could see this from the many examples grouped by android community in Android MVP blueprint repository [5]. We study this example and starts with a simple triad defining by a contract (an interface) of components that are: View Presenter and Controller An important aspect is that the View is identified by a Fragment class, next the reference to View meaning the architecture View that is a Fragment, while the presenter is a middle class that keeps independent the Fragment from the model layer of our application. In the Controller we keep all the logic of data and database query, in many architecture the name Controller are substituted with Repository or Service. We don't use Service for not have confusing about the Service component of Android and we don't use Repository word because it means that encapsulated some database logic, in our app the database logic is all into Firebase API so "repository" was not valued as a good name choice.

The activity act like a management of the triad of components of our architecture. Its responsibility is of instantiate the triad and managed the life cycle of it, in particular when the life cycle go into Destroy or Stop status it must detach the listeners from Firebase database in order to avoid memory leak. Because the role of activities is similar, in order to reduce duplicate codes, almost all activity extend a BaseActivity that implements checks and other common functions. Only the LoginActivity doe not extend BaseActivity.

In order to achieve the separation of concern we created for every model data class needed to the View a copy class defined as ViewModel class. The Presenter is the responsible of this conversation that was done when new data come from the Controller.

For every feature showed in the introduction with the exception of Geogift the application have a triad of classes plus the activity over other utility classes like Adapter and ViewHolder.

The communication between components of our architecture are always direct calls to method from upper layer (the widget views) and callbacks from lower layer (Controller or android Service) to upper layer except between Views and Presenter.

The communication between Views and Presenter are done through interfaces as the figure 2 show. PresenterImp extends Presenter interface that are the reference with which ViewImp communicate, ViewImp extends View interface that are the reference with which PresenterImp communicate.

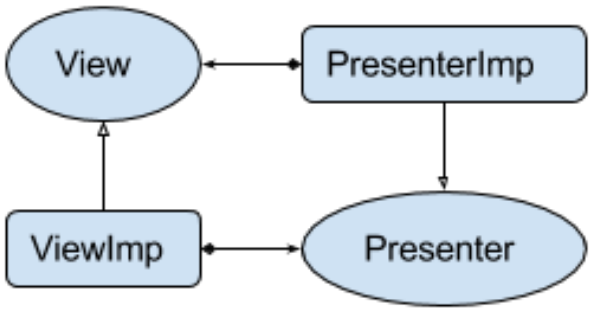

Fig. 2. MVP interface

The main packages identified a feature of the application and they contain the triad and other utility classes of the feature, for example a DialogFragment.

Aside all pregius of an architecture like MVP we met some difficulties that require the implementations of workaround or a complex solution for a simple task.

Our need to isolated firebase from other parts of app forced us to use firebase with some limitation. For example Google I/O 2017 available in a open repository in Github [6] app use a more complex MVP architecture but the components dependent a lot on firebase API.

Other disadvantages caused by architecture choices are the several callback communication between component, a simple click in a ViewHolder must pass into an Adapter then a Fragment then a Presenter and finally the Controller. By the way we believe that this strong division of relationship between components is a good investments for the future extension of the app.

In the background the app runs four android Service that act like listener to the status of some data into database. The UserMonitorService listen when the status of user change, in particular it monitors the relationship status. The GeogiftMonitorService manages the life cycle of Geogifts, it downloads them, registers them into LocationService (a Google API Service used to monitor user's location) and monitors the status of them, if one is destroyed it unregisters it from the LocationServices. When LocationService triggers a Geogift the GeogiftTransitionService acts, it is an IntentService and it create a notification to the user when he discover a Geogift. The last Service is the MessagesMonitorService, it monitors the status of all chats and create notifications when new messages are received, it implements a caching system for

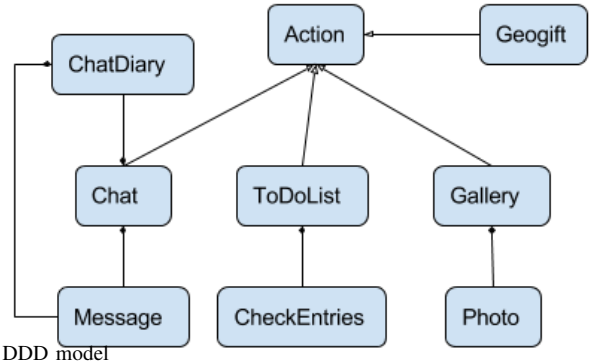

Fig. 1. DDD model

the notification. The ChatActivity is binded to it in order to reset this notification counter when user enter into a chat.

IV. External API

A. Glide

Glide[1] is a fast and efficient open source media management and image loading framework for Android, we use it for download images and display to the user.

B. Firebase

Firebase[2] is a powerful platforms that give us a free server with noSQL database, remote storage and other tools with a front-end API which we use it in order to have asynchronous network connection for CRUD operation of our data;

C. MaterialCalendar

A CalendarView Material[3] look likes. It is used for the calendar feature where user could see his saved messages.

D. ImagePicker

ImagePicker[4] is a library to select single or multiple images from the gallery and camera, used in our Galleries

E. Image Cropper

Image Cropper[5] is a library that offers activities for get a photo for images into the device and crop them. It is used for obtain specific ratio of an image, to set user's avatar, couple cover, chat and gallery cover.

F. Google Mobile Services API

Google Mobile Services[6] are a set of library for In particular we use the `auth` package for Google authentication in app, `location` and `places` packages are used for its services of localization and implements our geogift feature and the `maps` package for the Map section from the main screen of Sweetie.

V. Main issues

A. ViewPager with fragment

The implementation of a ViewPager with a FragmentPagerAdapter for the tab layout in the dashboard of app lead to many problems for managing the lifecycle of the three fragment Diary, Home and Map with the main lifecycle of Dashboard activity. In some cases we were obligated to use workaround with dirty flag [**?**] and break the division set by the architecture.

[1]https://github.com/bumptech/glide
[2]https://firebase.google.com/
[3]https://github.com/prolificinteractive/material-calendarview
[4]https://github.com/esafirm/android-image-picker
[5]https://github.com/ArthurHub/Android-Image-Cropper
[6]https://developers.google.com/android/reference/packages

B. RecyclerView

Managed the hierarchies of ViewHolder was a pain, the RecyclerView impose to developer to avoid polymorphism in the creation of ViewHolder and encourage the use of a switch statement with a check on the type represented by an int value, see onCreateViewHolder on android doc [8]. Fortunately this limitation is only in the creation phase of a ViewHolder, in the binding phase we use the power of polymorphism. Despite of this fact, the RecyclerView, unlike the ListView, give us a more simple flow and an easy managing of the items into the list.

C. Images

Use image in an Android app and at the same time maintain the usability of it is an very hard task. The Glide library helps us a lot in the download but not in the upload. Firebase API for the upload of files are minimal. The main problems here are:

- Upload image and maintain the status progress after Activity is destroyed: we resolve it by change the status of the object into remote server so it persist over the life of the activity, so the both partners see the upload progress;
- Reduce the images download time: we don't implement a compression library into the application but it is necessary for have good usability like WhatsApp, at this moment the first opening of an image it is damn slow.

D. Chat keyboard

As Hackborn said [9] the Android SDK doesn't expected that a developer needs to know when soft keyboard is open, unfortunately this is not the case. In order to create a custom emoticons keyboard interchangeable with soft keyboard we need this information. So we built a workaround and implement a listener that it is triggered when the GlobalLayout object change height, from that we calculate the height of the soft keyboard. In the layout we put a FrameLayout placeholder that use this calculated height for push up the chat items and give space to the custom emoticons keyboard and soft keyboard, in this way the switch between keyboards is pleasant. We have not be able to resolve a bug in the first opening of keyboard, that is the moment where app registers the height of soft keyboard, in this case the window resize is not executed correctly: the toolbar disappears as long as the keyboard remains open.

VI. Future Works

Despite of the great amount of operation that user could do with Sweetie, there are others that we were not able to implement in this version, they are:

- Support to other language, in this version the app is in english but the library string get the language set in the system;
- The social sharing buttons for make Sweetie less close with user contents;
- Build a more robustness notification service, in this version when user kills the application also the service

is killed. For do that we will need to run the service into another process;
- A local cache system and compress system for images, this is a serious lack because decrease a lot of our app usability, fortunately Glide reduce this lack but it cache the images only if user previously open them the result is a long wait for the user on the first opening of an image. Over it we need also a compression image library in order to increase the usability and to decrease the use of network connection;
- Finally we need to mentions security that will become necessary if the app in the future it will be commercialize through the Google Play Store. In this version the app does not use any type of cryptography for the messages and file uploaded into server neither Firebase database do all the checks that a back-end should do.

Working on the project new ideas were born, we describe here the most interesting:

- Manage the Geogift discovery with a more smarter way in order to reduce the use of battery and the useless use of localization sensors. For example, the monitoring service should be activated only in proximity of geofence. Improve the experience allowing the sending of Geogifts only in the current physical location and not on remote. Same for the discovering, limited by an expiration time and a distinction between walk and car movements;
- Import the messages from WhatsApp in order to reduce the barrier of usage from new users;
- A machine learning engine will learn user actions, in particular which items are bookmarked and saved to diary. Thanks to profiling and user-user matching, the learning could be suggest what elements may be of interest, especially to early adopters on their imported messages;
- The diary feature that, for us, is still immature and future extensions could bring something more useful for Sweetie's users. In the future we could implement that every features contributes to filling the diary. User can save the most important (emotionally tied) objects present in the home's actions like individual messages and photos. The result will be a kind of story that grows in time;
- Handle the breaking of couples is crucial for maintain active users on the app. One idea is to change main theme of app (different palette colors for example) for an emotional detachment from the relationship just lost, and encourage users to use some dedicated features awaiting a new love story.

VII. Conclusion

Certainly we can say that Sweetie is a great didactic project. We have needed to learn a lot of android SDK content: Fragment, RecyclerView, ConstraintLayout, Bitmap, ToolBar are only some of classes that we need to learn to manage and use correctly. We also learn to use some powerful Google library that are essential for developing in Android.

We learn also some of the best practices in architecture design for Android. Model View Presenter is a great pattern although we didn't use all the advantages: the testing advantage in particular.

Nevertheless the issues leave open we are satisfied of this release and we are confident that this is the first step for a complete application that can reach the Play Store. Anyway this is only the beginning we should have to learn new components and library of Android for extending our knowledge.

References

[1] Eric J. Evans, *Domain Driven Development*, Addison-Wesley Professional, 2004.

[2] Unlce Bob (Robert Martin) talk on software architecture https://www.youtube.com/watch?v=Nsjsiz2A9mg

[3] Dianne Hackborn's post about android architecture https://plus.google.com/+DianneHackborn/posts/FXCCYxepsDU

[4] Uncle Bob (Robert Martin) post on Clean Architecture https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

[5] Android MVP blueprints repository https://github.com/googlesamples/android-architecture

[6] Google I/O 2017 Android Application, open source repository https://github.com/google/iosched/tree/master/lib

[7] DirtyFlag pattern, from Bob Nystrom, *Game Programming Patterns* http://gameprogrammingpatterns.com/dirty-flag.html

[8] RecyclerView.Adapter documentation page https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html

[9] Dianne Hackborn's replies on Soft Keyboard https://groups.google.com/forum/#!topic/android-platform/FyjybyM0wGA