# Software Architecture Principles Analysis of Telegram, Android Mobile Messaging Application

Yi-Chen Lin, Dheeraj Mohan Kumar, Neha Pradeep Patil, Ku Kim

MSWE SWE264P Distributed Software Architecture

Team 11 Final

Sam Malek

March 15, 2023

# Table Of Contents

# Introduction

## Overview of Telegram Application

Telegram is a cross-platform, cloud based, messaging application that has 700 million monthly active users. The application allows users to create channels, start chat groups, and send photos, videos, and files.

Telegram offers a mobile application for Android and iOS devices, a desktop application for Windows, MacOS, and Linux, and a web-based application that can be accessed from a web browser on any device. We chose Telegram App for Android as a project subject. (GitLink_to_project)

Analyzing the architecture of Telegram would allow developers to gain a sense of the architectural styles, patterns, and designs used in messaging applications. The analysis of Telegram not only exemplifies the general style behind messaging applications, but also provides an additional look into cross-platform and cloud-based architecture.

## Project Focus

Although Telegram shines  as a cross platform application, the Android version provides a rich architectural cornucopia to analyze for Domain Specific Software Architecture patterns and components. The key components in Telegram's Android Application are: Activities, Fragments, Views, Custom Views, Networking, API Clients, Protocols, SQLite, Security, media handling. The topographical breakdown of Telegram's architecture will focus on these key components.

## Setting Up Telegram on Android Studio

1. Import Telegram project to your own GitHub repository

2. Install Android studio

Download Android Studio & App Tools - Android Developers

3. Import Telegram project

**Android Studio>File>Project From Version Control**

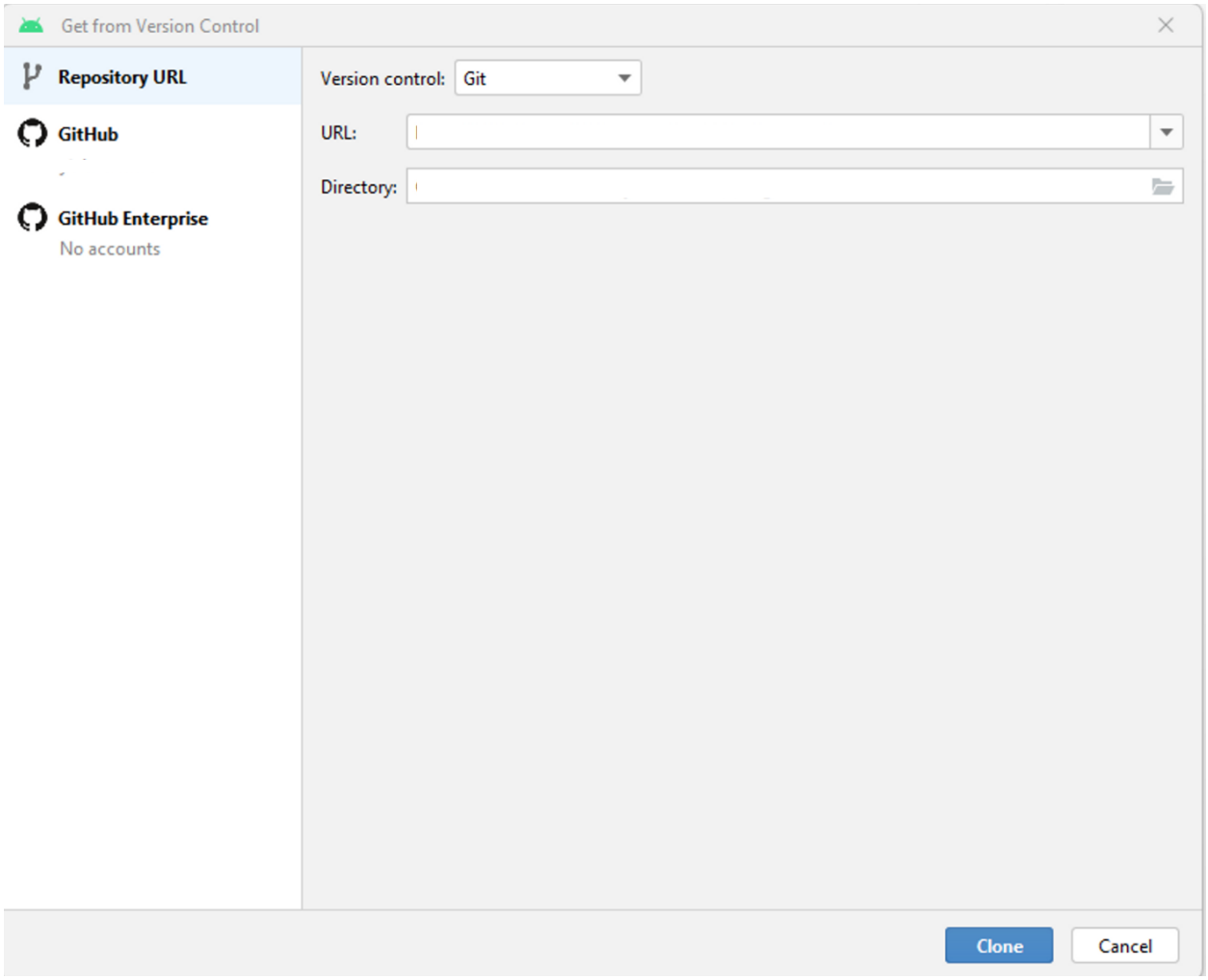



*Figure 1. Importing Telegram Using Version Control*

## 4. Set up an Android emulator

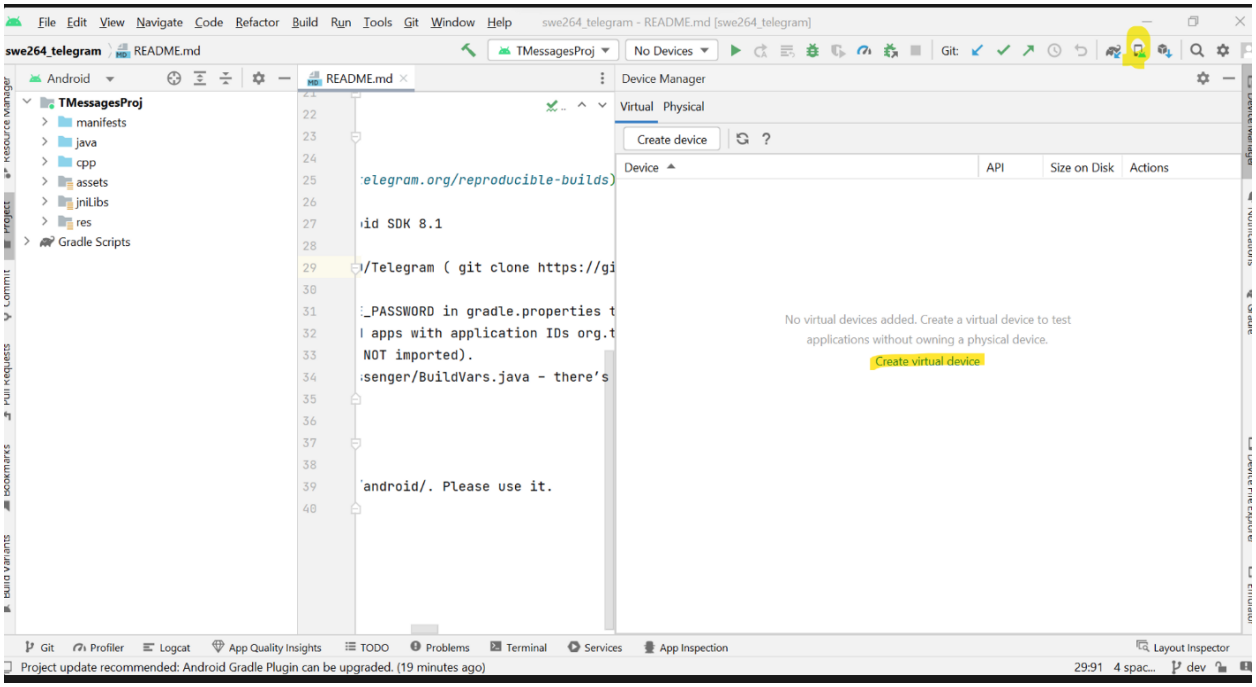



*Figure 2. Opening the device manager*

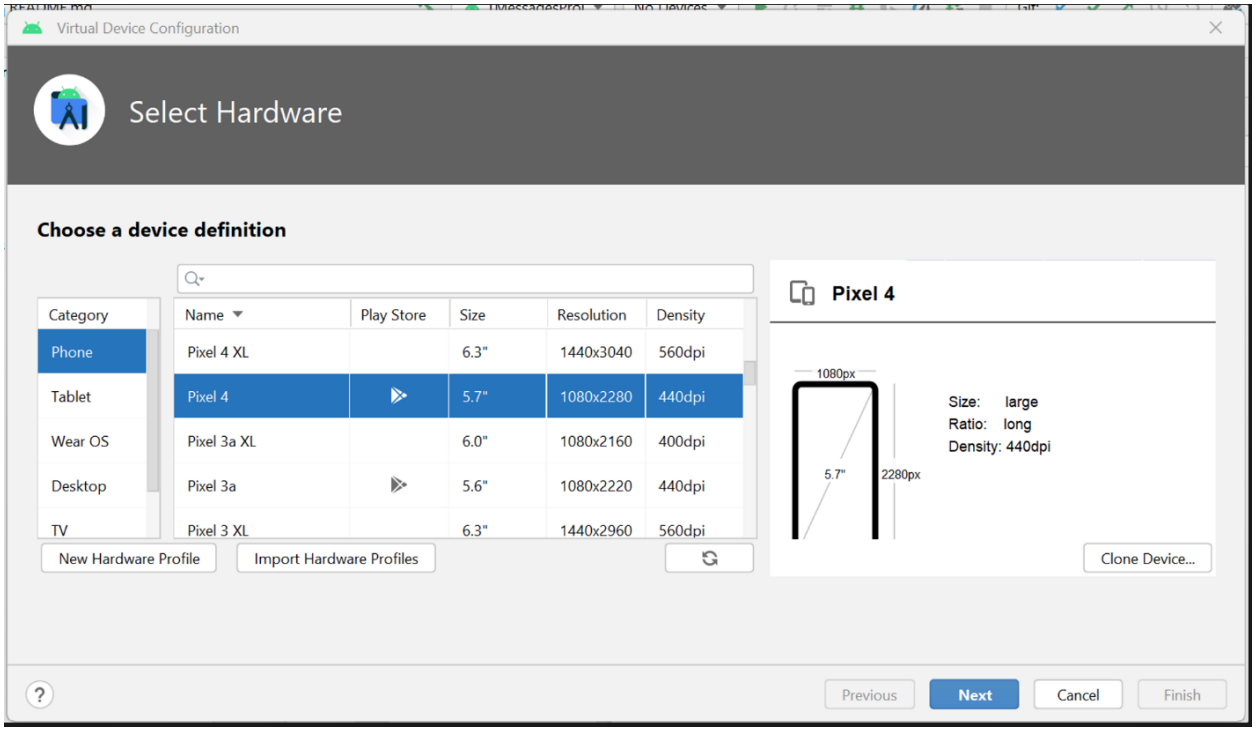



*Figure 3. Creating a device on Emulator*

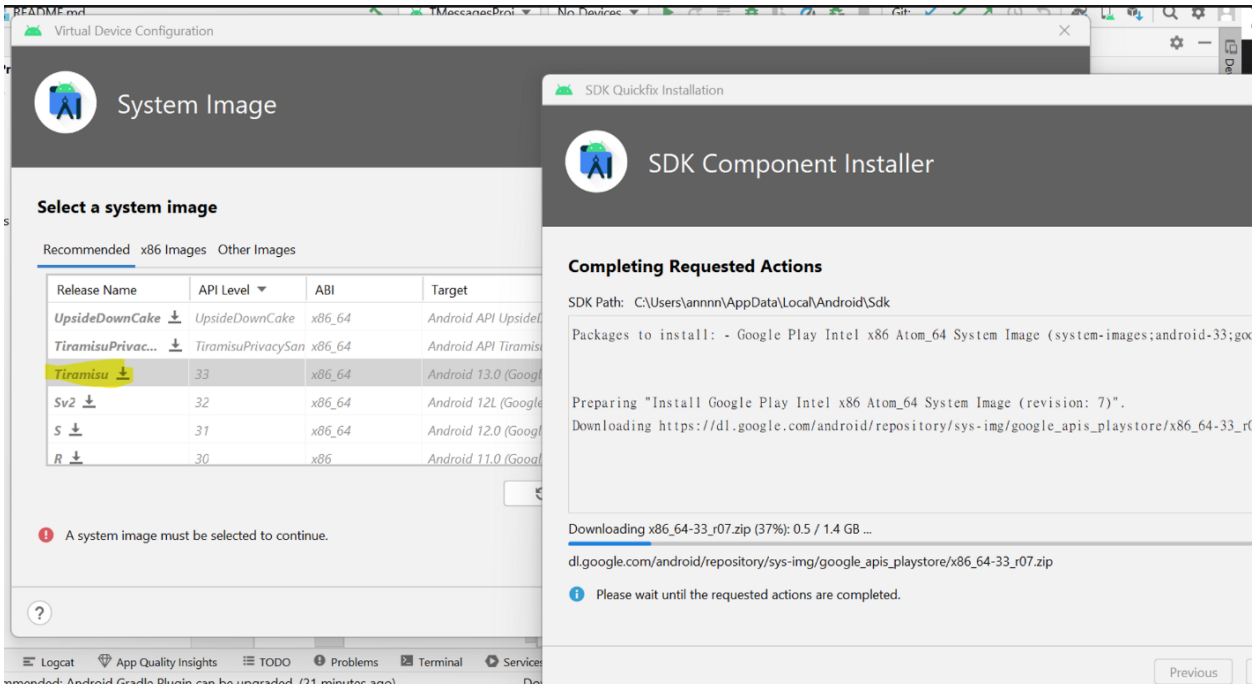



*Figure 4. Downloading a System Image*

## 5. Apply for Telegram API_ID

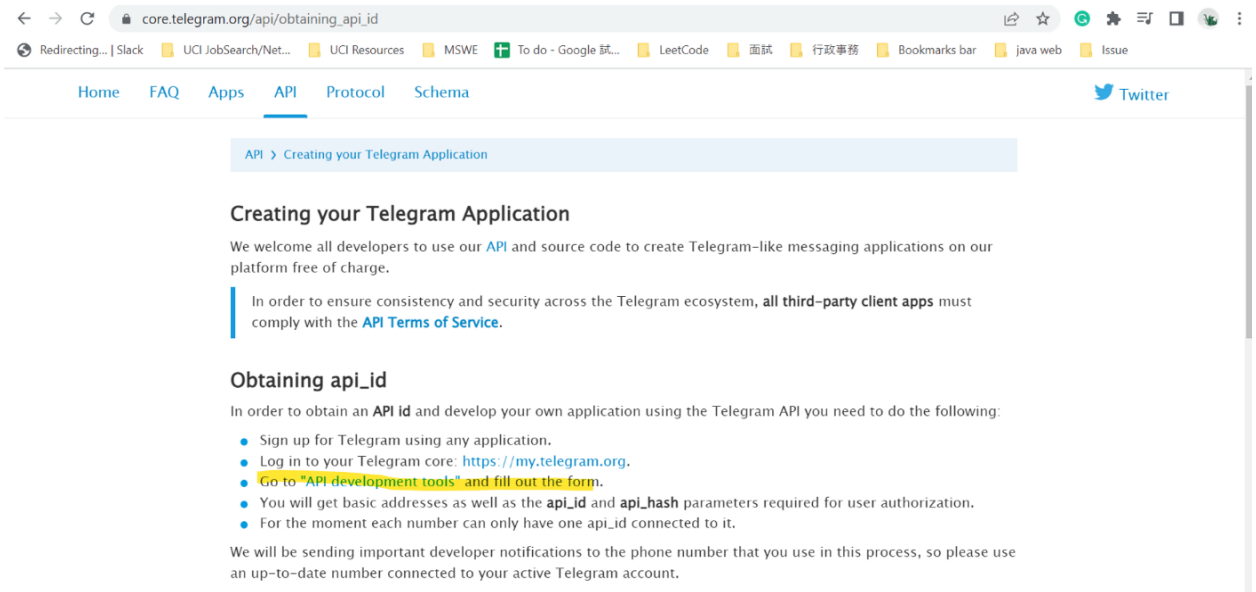



*Figure 5. Applying for the Telegram APP_ID*

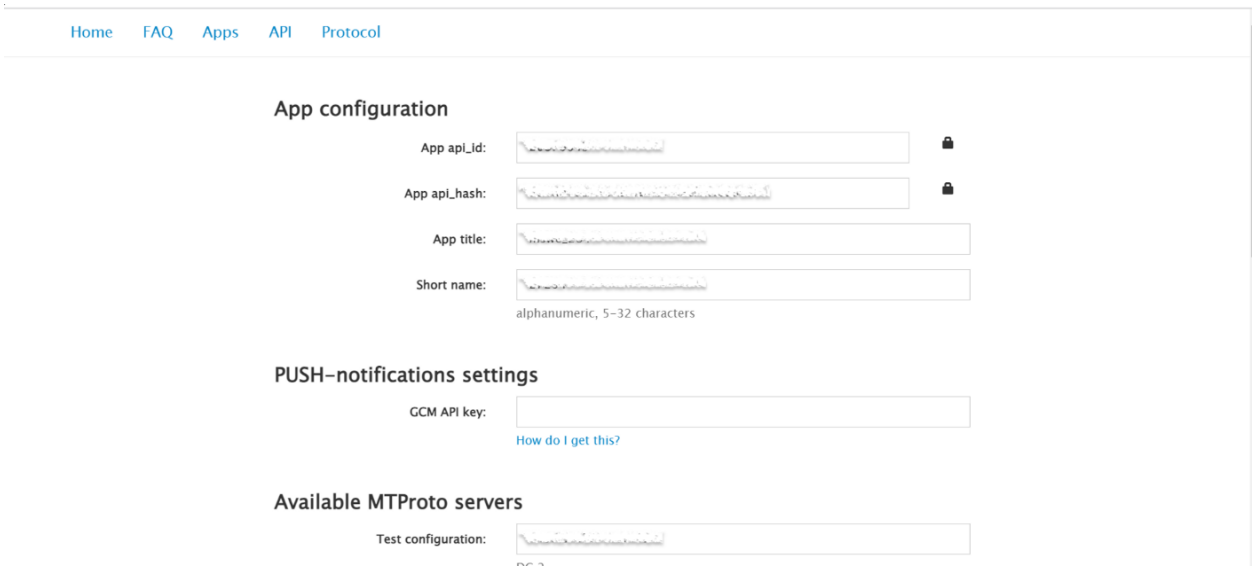



*Figure 6. Input of APP_ID*

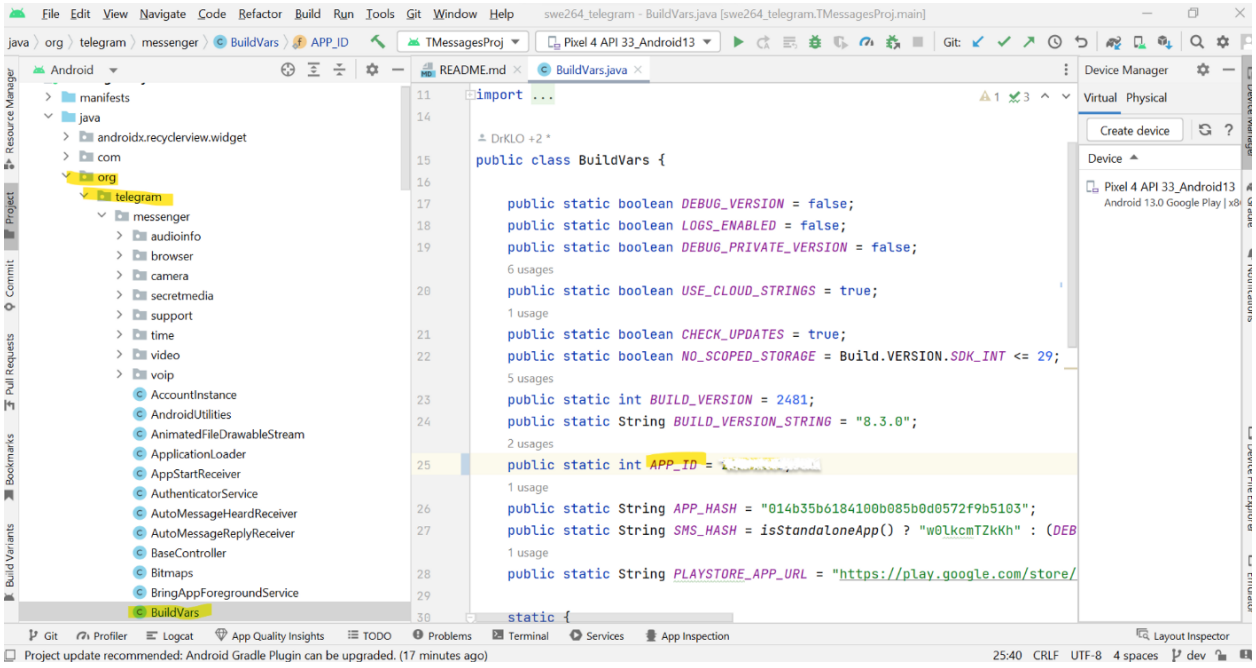



*Figure 7. Building Exoplayer on Android Studio*

## 6. Build project

# Architecture Analysis Approach

In order to get an overview of the functionalities of Telegram, we investigated the existing 144 UI Activities. We listed all of the Activities and drew a web of their interactions. This allowed us to form a solid idea of how the activities are grouped. We prioritized the groups of activities that are most relevant in understanding the messaging portion of the app. Functions such as the settings, or FAQ are not normally used and thus were filtered out. We focused primarily on the group of activities that are relevant to the chat functionalities such as sending and receiving plain text, sending and receiving media, and initiating and receiving calls.

After the activity relationships were established, a few pieces of the puzzle remained: finding relevant connectors, services, and content providers. We delved into the manifest file and listed out all of the intent filters, services, and providers and described the roles that each component plays. Many of the services and intent filters were irrelevant to the main chat functionalities and were put aside. A deeper dive into source code allowed us to find the association between activities, services, and content providers.

# Architecture Styles

While determining the overall architecture of Telegram, many architectural styles were unearthed. Telegram for Android has object-oriented style, layered style, client-server style, publish-subscribe style, message-based explicit invocation style, and message-based implicit invocation style.

### Object-Oriented Style

Telegram for Android uses object-oriented style in its architecture. The application's UI logic is separated from the business logic, and each component is implemented as an object with its own properties and methods. The objects are instantiated before the objects' methods can be called.

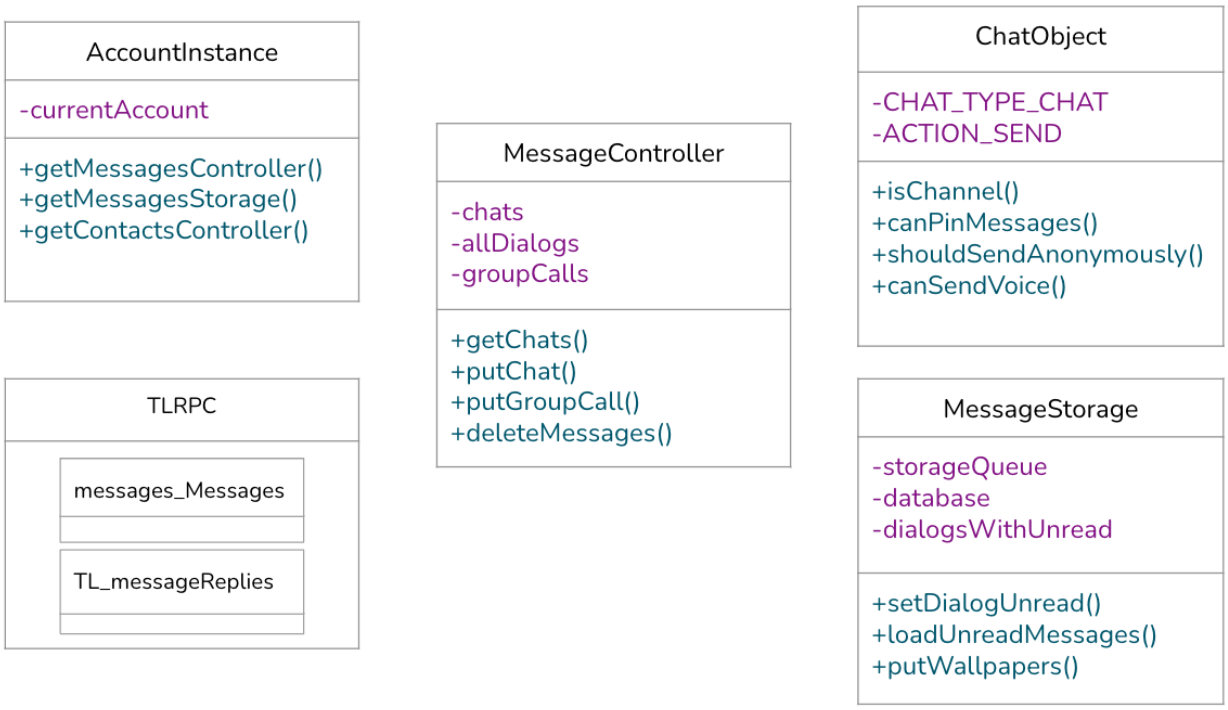



*Figure 8. Object examples in Telegram for Android*

## Layered Style

Telegram for Android follows a layered style. The layered style separates an application into layers, each of which has a specific responsibility and interacts with the layers above and below it. Telegram for Android's layered architecture can be broken down into the following layers:

1. GUI Layer: This layer is responsible for handling the user interface of the application. It includes activities, fragments, and views that interact with the user.
2. Controller Layer: This layer is responsible for the business logic of the application.
3. Data Access Layer: This layer is responsible for handling the storage and retrieval of data. It includes database classes and network classes that interact with remote data sources.

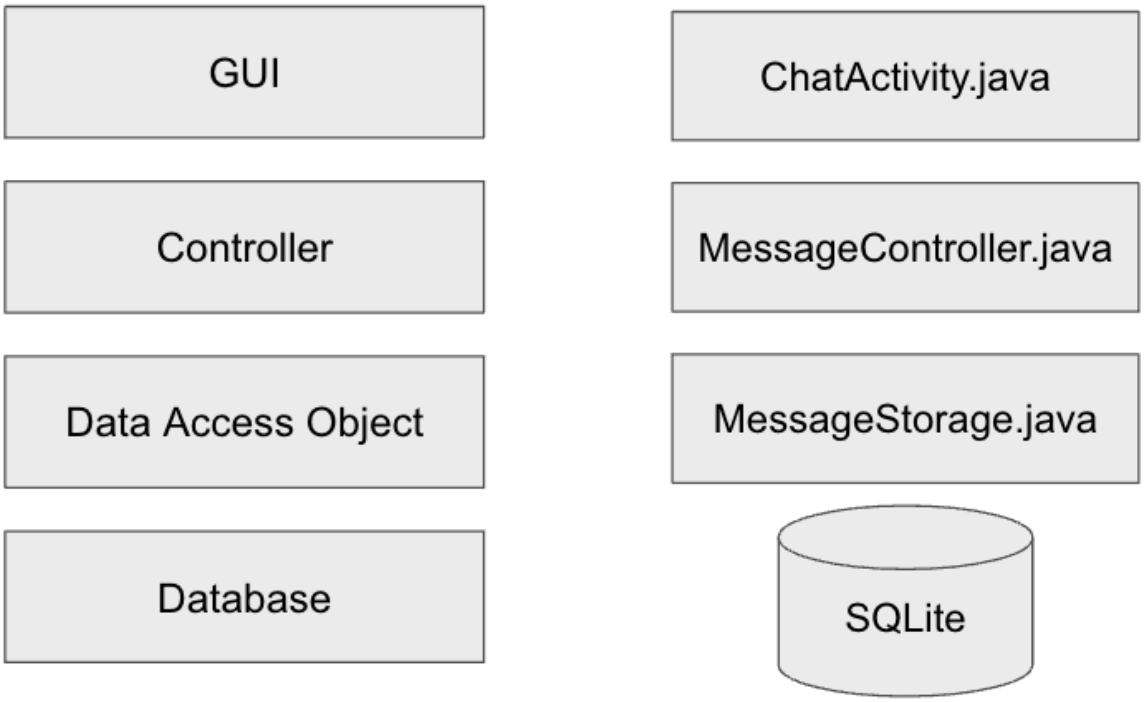



*Figure 9. Layers and corresponding class examples to each layer in Telegram for Android*

## Client-Server Style

Telegram follows a client-server architecture. The client is the Telegram app installed on a user's device, and the server is the Telegram backend that handles all the requests made by the clients.

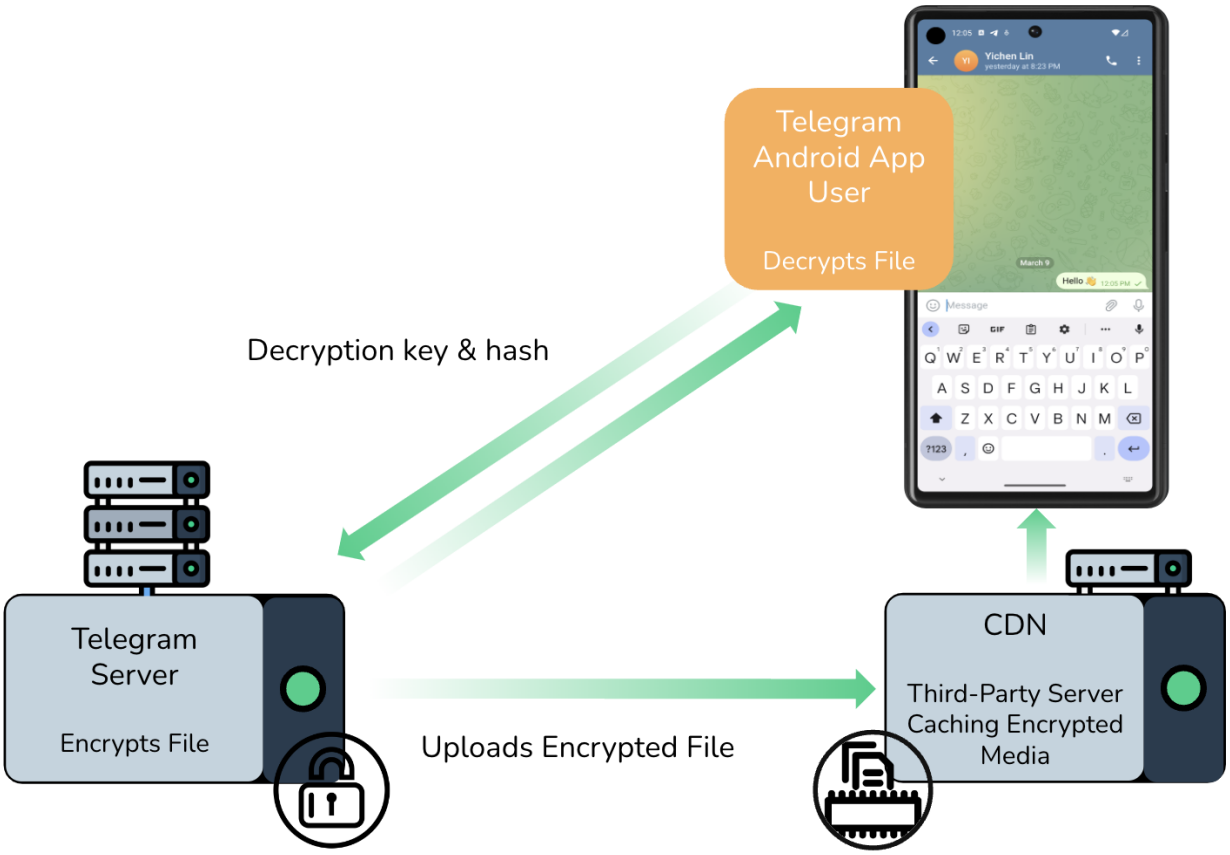



*Figure 10. Telegram Android app user as the client and Telegram server*

## Publish-Subscribe Style

Implementation of BroadcastReceivers shows that Telegram follows the publish-subscribe style. Broadcast receivers are leveraged to dispatch messages to multiple receiving components and receive Intents from other components.

NotificationsService class is a service component of the Telegram for Android app's notification handling system, responsible for initializing the app and handling incoming notifications. The **onDestroy()** method is called when the service is destroyed and sends a broadcast intent to start the app's push notification service if it is enabled in the app's settings.

AndroidManifest.xml

```
459    <receiver android:name=".AppStartReceiver" android:enabled="true" android:exported="false">
460        <intent-filter>
461            <action android:name="org.telegram.start" />
462            <action android:name="android.intent.action.BOOT_COMPLETED" />
463        </intent-filter>
464    </receiver>
```

*Figure 11. Declaration of AppStartReceiver and the actions it subscribes to in AndroidManifest.xml*

NotificationsService.java

```
25    public int onStartCommand(Intent intent, int flags, int startId) { return START_STICKY; }
28
      DrKLO
29    @Override
30    public IBinder onBind(Intent intent) { return null; }
33
      DrKLO +1
34    public void onDestroy() {
35        super.onDestroy();
36        SharedPreferences preferences = MessagesController.getGlobalNotificationsSettings();
37        if (preferences.getBoolean( s: "pushService",  b: true)) {
38            Intent intent = new Intent( action: "org.telegram.start");
39            sendBroadcast(intent);
40        }
41    }
```

*Figure 12. Sending broadcast in NotificationsService*

AppStartReceiver.java

```
1   /.../
8
9   package org.telegram.messenger;
10
11  import android.content.BroadcastReceiver;
12  import android.content.Context;
13  import android.content.Intent;
14
      1 usage   DrKLO +1
15  public class AppStartReceiver extends BroadcastReceiver {
16
          DrKLO +1
17      public void onReceive(Context context, Intent intent) {
18          if (intent != null && Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) {
19              AndroidUtilities.runOnUIThread(() -> {
20                  SharedConfig.loadConfig();
21                  if (SharedConfig.passcodeHash.length() > 0) {
22                      SharedConfig.appLocked = true;
23                      SharedConfig.saveConfig();
24                  }
25                  ApplicationLoader.startPushService();
26              });
27          }
28      }
29  }
```

*Figure 13. AppStarReceiver as a BroadcastReceiver*

The other example that follows the publish-subscribe style is NotificationCenter class. The NotificationCenter class is used as the central hub for managing communications among different components, with other parts of the app subscribing to specific notifications and the NotificationCenter broadcasting those notifications when they occur. *Figure 14* is an example that shows adding ChatActivity as a new observer for the given notification events.

ChatActivity.java

```
1652  if (threadMessageId == 0) {
1653      getNotificationCenter().addObserver( observer: this, NotificationCenter.screenshotTook);
1654      getNotificationCenter().addObserver( observer: this, NotificationCenter.encryptedChatUpdated);
1655      getNotificationCenter().addObserver( observer: this, NotificationCenter.messagesReadEncrypted);
1656      getNotificationCenter().addObserver( observer: this, NotificationCenter.botKeyboardDidLoad);
1657      getNotificationCenter().addObserver( observer: this, NotificationCenter.updateMentionsCount);
1658      getNotificationCenter().addObserver( observer: this, NotificationCenter.newDraftReceived);
1659      getNotificationCenter().addObserver( observer: this, NotificationCenter.chatOnlineCountDidLoad);
1660      getNotificationCenter().addObserver( observer: this, NotificationCenter.peerSettingsDidLoad);
1661      getNotificationCenter().addObserver( observer: this, NotificationCenter.didLoadPinnedMessages);
1662      getNotificationCenter().addObserver( observer: this, NotificationCenter.commentsRead);
1663      getNotificationCenter().addObserver( observer: this, NotificationCenter.changeRepliesCounter);
1664      getNotificationCenter().addObserver( observer: this, NotificationCenter.messagesRead);
1665      getNotificationCenter().addObserver( observer: this, NotificationCenter.didLoadChatInviter);
1666      getNotificationCenter().addObserver( observer: this, NotificationCenter.groupCallUpdated);
1667  } else {
1668      getNotificationCenter().addObserver( observer: this, NotificationCenter.threadMessagesRead);
1669      if (isTopic) {
1670          getNotificationCenter().addObserver( observer: this, NotificationCenter.updateMentionsCount);
1671          getNotificationCenter().addObserver( observer: this, NotificationCenter.didLoadPinnedMessages);
1672      }
```

*Figure 14. Adding ChatActivity as a new observer for given events*

## Message-based Explicit Invocation Style

Telegram for Android follows the message-based explicit invocation style, where intents explicitly identify the name of the recipient component. Within the NotificationsController class, there is a method called **processNewMessages()**. In this method, an intent explicitly identifies PopupNoticationActivity which is responsible for displaying the notification to the user. Later the intent is sent to the PopupNoticationActivity to notify users a new message is coming.

By using explicit invocation, Telegram ensures that the PopupNotificationActivity is the only receiver that will handle the new message intent. This approach improves the

app's security by preventing other apps or components from intercepting the message intent and potentially accessing sensitive user data.


*Figure 15. Intent with an explicitly identified recipient in NotificationsController.processNewMessages()*

## Message-based Implicit Invocation Style

Telegram for Android follows the message-based implicit invocation style, where intents are not explicitly specified with the intended recipients. Instead, they rely on intent filter mechanisms to determine which components will receive the intent. An example of this can be seen in ChatActivity class. In the ChatActivity, **mediaDataController.installShortcut()** method is called. As shown in *Figure 16*, when the method is called, an Intent object named ‘addIntent’ is instantiated. This intent is not tagged with the intended recipients. Instead, it is broadcasted with the action **"com.android.launcher.action.INSTALL_SHORTCUT"** to tell the launcher to add the shortcut to the home screen.

MediaDataController.java

```
4960    Intent addIntent = new Intent();
4961    if (bitmap != null) {
4962        addIntent.putExtra(Intent.EXTRA_SHORTCUT_ICON, bitmap);
4963    } else {
4964        if (user != null) {
4965            if (user.bot) {
4966                addIntent.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE, Intent.ShortcutIconResource.fromContext(ApplicationLoader.applicationContext,
4967            } else {
4968                addIntent.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE, Intent.ShortcutIconResource.fromContext(ApplicationLoader.applicationContext,
4969            }
4970        } else {
4971            if (ChatObject.isChannel(chat) && !chat.megagroup) {
4972                addIntent.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE, Intent.ShortcutIconResource.fromContext(ApplicationLoader.applicationContext,
4973            } else {
4974                addIntent.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE, Intent.ShortcutIconResource.fromContext(ApplicationLoader.applicationContext,
4975            }
4976        }
4977    }
4978
4979    addIntent.putExtra(Intent.EXTRA_SHORTCUT_INTENT, shortcutIntent);
4980    addIntent.putExtra(Intent.EXTRA_SHORTCUT_NAME, name);
4981    addIntent.putExtra( name: "duplicate",  value: false);
4982
4983    addIntent.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
4984    ApplicationLoader.applicationContext.sendBroadcast(addIntent);
```

*Figure 16. Intent without a tagged recipient in MediaDataController.installShortcut()*

# Activity Components and User Interface

Based on our investigation, we categorized different sub-activities into bigger categories. Below is a list of the categories of activities that are significant for Telegram.

1. **Login**: The login activity is the main activity responsible for logging a user into their telegram account. Users can choose different methods of authentication such as SMS, phone call, and email for logging into their telegram account.
2. **Group**: The feature that every messaging app includes, where users can add close contacts to create a chat room only for those people, and communicate with them together. Users can also invite participants to join the group even if the participant is not in their contact list.  Groups support almost all features a chat activity provides like sending text messages, photos, videos, voice messages, docs, location(current and live), gifs, stickers, emoji, etc.,

3. **Chat**: The main activity, which enables users to communicate with one another in private or in group chats. Users can send text messages, photos, videos, voice messages, and other types of files.
4. **Channel**: Public chat rooms that users can join to discuss specific topics. Channel admins can broadcast messages to all members of the channel.
5. **Contact**: A list of contacts that users can chat with. Users can add, delete, and edit contacts.
6. **Setting**: A range of customizable settings that users can adjust to suit their preferences. These include privacy settings, chat settings, notifications, and more.
7. **Bot**: Automated programs that can perform various tasks, such as providing weather forecasts, news updates, and more. Users can interact with bots via chat.
8. **Sticker**: Animated or static images that can be used to express emotions or convey a message in a chat.
9. **Call**: A feature that allows users to make voice or video calls to other Telegram users.

## Activity Table

| Activity | Description | Part of which larger system |
|---|---|---|
| ActionIntroActivity | Get user information such as QR code login, change phone number, etc., | Account setting |
| ArchivedStickersActivity | Archived sticker | Sticker |
| ArticleViewer | View Article.<br>Mark and draw the article | Media reading |
| AutoDeleteMessage | Setting of messages auto-deleting. | Message setting |

| | | |
|---|---|---|
| sActivity | Auto deletes after 1 day/ 1 week/ 1 month/ custom time | |
| CameraScanActivity | Scan QR code, passport, | Camera scan |
| ChangeBioActivity | Edit profile biology | Edit profile |
| ChangeNameActivity | Edit profile name | Edit profile |
| Change Username Activity | Edit user name | Edit profile / Account setting |
| ChannelAdminLogActivity | This is a list of all service actions taken by the group's members and admins in the last 48 hours. | Service Actions Log |
| ChatActivity | Click chats<br><br>(1)Clear chat history<br>(2)Report<br>(3)Leave group<br><br>Save to music/ Edit/ Copy/ Forward | Chat |
| ChatEditActivity | Edit channel name.<br>Set new photo<br>Setting of if new members can see earlier messages | Chat |
| ChatLinkActivity | Link other groups and channels | Chat |
| ChatReactionsEditActivity | Settings permission of members in a chat eg. Allow participants to react to group messages/ Members ot the group can’t add any reactions to messages. | Chat |
| ChatRightsEditActivity | Channel ownership setting | Chat |
| ChatUsersActivity | Remove users | Chat |

| | | |
|---|---|---|
| ContactAddActivity | Add contact | Contact |
| ContactsActivity | Invite contacts who are not using Telegram.<br>Bot setting | Contact |
| CountrySelectActivity | Used to select a country while setting up an account for the first time. | Login |
| CreateTopicEmptyView | Used to create Empty Topic view for group chat | Chat (group chat) |
| DataAutoDownloadActivity | Activity that handles automatic download for media shared on telegram | Settings |
| DataSettingsActivity | Activity that handles the data and storage settings. | Settings |
| DataUsage2Activity | Activity used to track data usage statistics. | Data Usage |
| DefaultThemesPreviewCell | Theme section in chat settings | Settings |
| FeaturedStickersActivity | Trending stickers activity | Settings |
| FilteredSearchView | Used to filter the chats based on messages | Chat |
| FiltersSetupActivity | Sets up the filter activity, based on the user actions and input.<br>Filters based on contacts, groups, non-contacts, etc., | |
| GLIconSettingsView | Has a renderer that helps in setting the view. | |
| GroupCallActivity | Activity to set and handle group calls. | Calls |
| GroupCallTabletGridAdapter | Used to set data into view.<br>Basically the data like/fetched from | Calls |

| | | |
|---|---|---|
| | Group call activity.<br>Binder displays the participants on the call in the UI grid. | |
| GroupCreateActivity | Handles the actions that are needed for group creation. | Group Creation. For a chat or calls media exchange.<br>Mostly can call it a chat room |
| GroupCreateFinalActivity | This is the final step after creating the group. Loads contact list, image views, emojis, and other tools in the chat room. | -‘’- |
| GroupInviteActivity | This activity is going to handle the invite link to a group chat room.<br>When a user decides to create a new group chat and decides to invite users this activity is going to be invoked. It first checks for notification from the notification center if the chat is loaded. Also handles the expired invite link. | Should be under a group creating system. |
| GroupStickersActivity | Sending Stickers in the group chat room. Checks if the stickers are loaded, the chat room is loaded. Provides search functionality for the user to find his liking. | |
| IdenticonActivity | It loads the emoji’s and views for the chat. Also handles animation sets. | |
| IntroActivity | Provides intro UI.<br>It handles themes, displays intro text, loads the buttons for chatting, changes language, etc., | |
| InviteContactsActivity | This activity creates a view for inviting contacts to use the Telegram app. This | |

| | | |
|---|---|---|
| | will send an invite to selected contacts. | |
| KeepMediaPopup View | Maintains view and probably handles the deletions of chat cache. | Delete actions of chat, caches, media |
| KeyboardHideHelp er | Helps in positioning the keyboard. When the user starts typing it pops up the keyboard and hides it when the work is done. | Chat, texts |
| LanguageSelectAc tivity | Language Selection activity. It also has its own view, and provides a search function to look up languages. | Settings |
| LaunchActivity | Main activity, loads user account and permissions, chats, contacts all the supported activities in the app. | Launching the app |
| LauncherIconContr oller | Main app Icon | Launching the app |
| LightModeSettings Activity | Settings for UI, user preferences, themes wallpaper | Display settings. Settings UI |
| LinkEditActivity | Handles the links,<br>Group invite link. Maintains history of link generation and expiry.<br>Also maintains chat logs | Chat |
| LocationActivity | Location sharing, live location, current location. Is in connection with GPS and Maps. | Chat, Location service |
| LoginActivity | Handles user account setup. Takes care of identifying users via number, email and recovery of the user account if it exists, password reset and others. | Before launch. User Set up. |
| LogoutActivity | Will handle user log out activity. Asks the user if he is sure to logout and confirms actions. | User Set up |

| | | |
|---|---|---|
| LongPressListenerWithMovingGesture | Thinking it handles<br>Gesture typing | Chat |
| ManageLinksActivity | Maintains the main invite links, revoked invite links, expired links.<br>Handle any URLs shared | Group Invites |
| MemberRequestsActivity | Handles the main group channels if a user wants to subscribe/join this group | Group chats |
| MessageEnterTransitionContainer | Handles message transactions and transition | Messaging |
| MessageSeenView | Shows if a message is delivered, read. | Chat |
| MessageStatisticActivity | Handles message stats, who sent this message and stuff like that | Chat and Messages |
| NewContactBottomSheet | Handles the contact addition<br>First name Last name email etc,. | User details,<br>Contact info.<br>Basically a phone book |
| NotificationsCustomSettingsActivity | Handles the notifications, User can mute chats, or select diff sounds way of notification for diff chats | Notifications,<br>Service |
| NotificationsSettingsActivity | Handles calls, chats notification | Notifications,<br>Service |
| NotificationsSoundActivity | Handle sound<br>System sounds | Notifications,<br>Service |
| PasscodeActivity | Handles the password for the user account, what needs to be done if the user misses the authentication | Authentication |
| PassportActivity | Handles the user details, password and other info. | Authentication,<br>Main Launch activity |
| PaymentFormActiv | Handles payments | Payment Service |

| | | |
|---|---|---|
| ity | | |
| PeopleNearbyActivity | Looks for people nearby, mainly shows where the contacts live, like on a map view similar to snap chat | Geo location. Contacts |
| PhotoAlbumPickerActivity | Selecting multiple pictures from the user's photo gallery | Media |
| PhotoCropActivity | After selecting a photo, tapping on the photo allows the user to crop the picture and perform basic photo editing | Media |
| PhotoPickerSearchActivity | Can search to find a pictures on the web to attach and send | Media |
| QrActivity | Creating a QR code to share a chat or user | Media |
| SelectAnimatedEmojiDialog | Selecting an animated emoji to send in a chat | Emoji |
| ShareActivity | Gets Uri data, url, and hash to be shared | Messaging |
| SponsporedMessageInfoView | Shows sponsored message information such as "unlike other apps, telegram never uses your private data" | |
| Statistics Activity | Provides user statistics such as followers graph, top hours graph, interactions graph, growth graph, views by source graph, new followers by source graph. Languages graph. | |
| ThemeActivity | Selecting for a specific theme nightType (Dark Theme), day theme | Setting UI Theme |
| ThemePreviewActivity | | Setting UI Theme |
| ThemeSetUrlActivity | | Setting UI Theme |
| TwoStepVerification | Handles login events such as setting | Login |

| | | |
|---|---|---|
| Activity | password | |
| TwoStepVerification SetupActivity | Setting up Two Step verification during the login process. The two-step verification activity.java also processes “forgot password” events | Login |
| VolPFeedbackActivity | | Video Chatting |
| VolPFragment | Accepting, declining video calls | Video Chatting |
| VolPPermissionActivity | Checking permissions for audio, camera | Video Chatting |

*Table 1. Activity components of Telegram*

# Architecture Graph

The main architectural components in android, Activity, Service, Content Provider, and BroadCast Receiver were color coded.

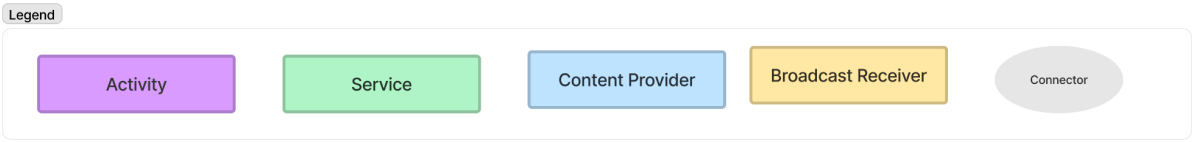



*Figure 17. Colors represented for  Android components*

## Launch

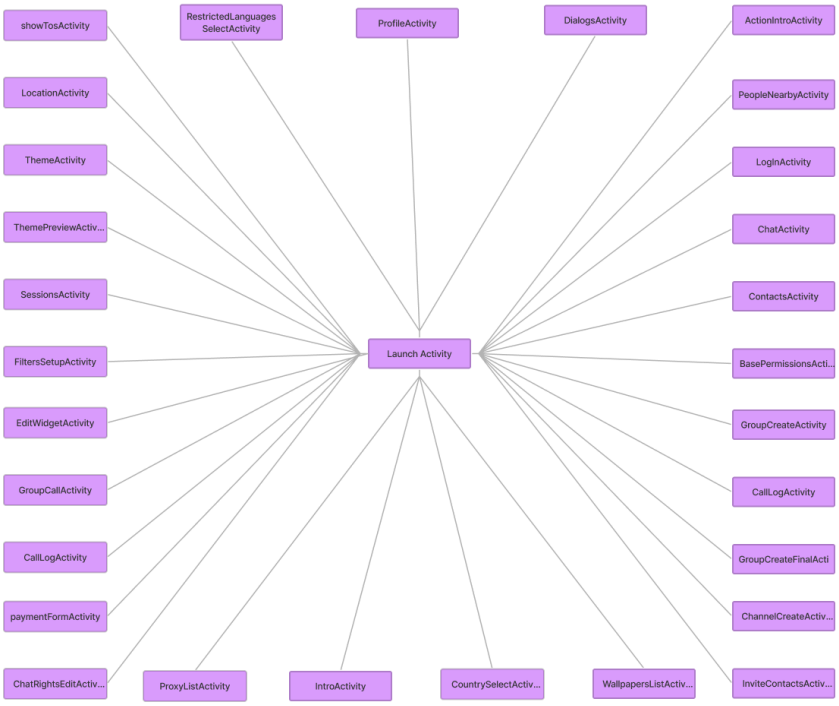



*Figure 18. Activity components connected to LaunchActivity*

The launch activity is solely responsible for loading data from a previous session. The launch activity connects to a blanket of activities. The main activities that are connected with the launch activity are Contact fetching, UI Loading, Theme Fetching, Loading previous messages, group chats, call logs. The launch activity ensures that the general flow of the application is consistent from starting the app.

Here we are focusing on the main functionalities the application provides such as follows.

## Login

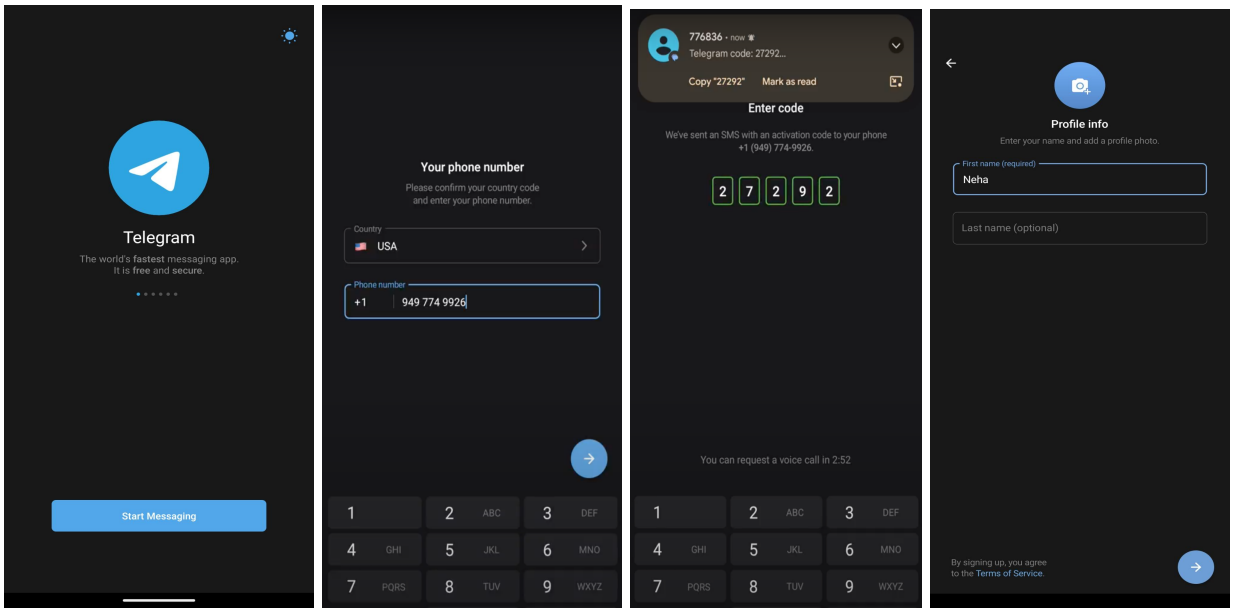



*Figure 19. Log in and set up*

The login activity is the main activity responsible for logging a user into their telegram account. The login activity consists of multiple views as stated in the above graph. Depending on the status of the user in the journey, different views are displayed. For instance, if a user is registering on telegram for the first time, the LoginActivity displays the LoginActivityRegisterView. Telegram provides multiple ways of authentication such as SMS, email; the LoginActivity is responsible for managing the user authentication.

Below is the representation of the activities involved in login inside the Telegram App.

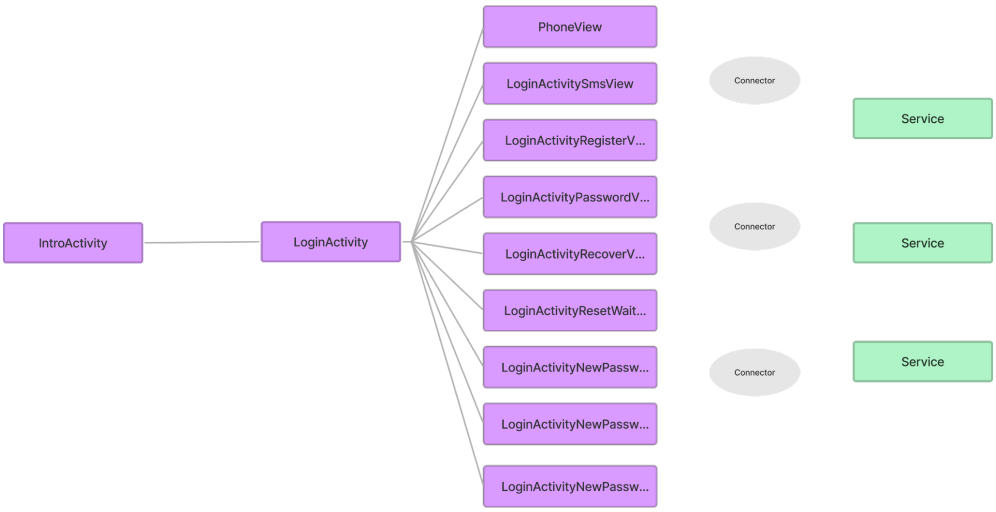



*Figure 20. Representation of the Telegram activities involved in login*

## Chat

Chat screen as seen on the app.

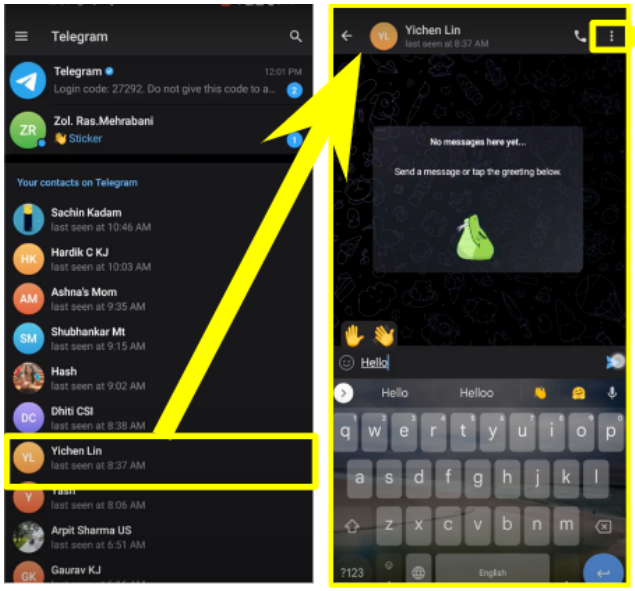



*Figure 21. Launch ChatActivity*

ChatActivity is the main activity responsible for displaying and managing the chat interface of Telegram. It represents a screen of a chat room. When a user selects a chat conversation, the ChatActivity will be launched. It initiates to get the message based on the user id, so this component interacts with the message service. After getting the messages, it displays the messages on the screen. It also invokes other activities such as ChatEditActivity and ChatLinkActivity. ChatLinkActivity is an activity to manage the process of creating and sharing chat links, which are links that allow users to join a Telegram group or channel. As for ChatEditActivity, it manages the process of editing or deleting messages in a Telegram chat room when a user long-presses on a message.

Below is the representation of the activities involved in chat inside the Telegram App.

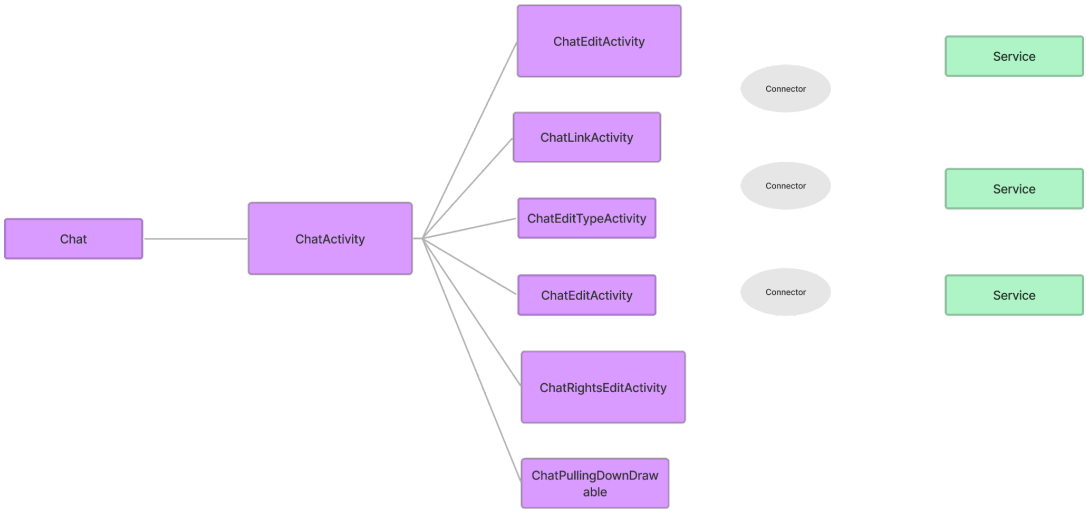
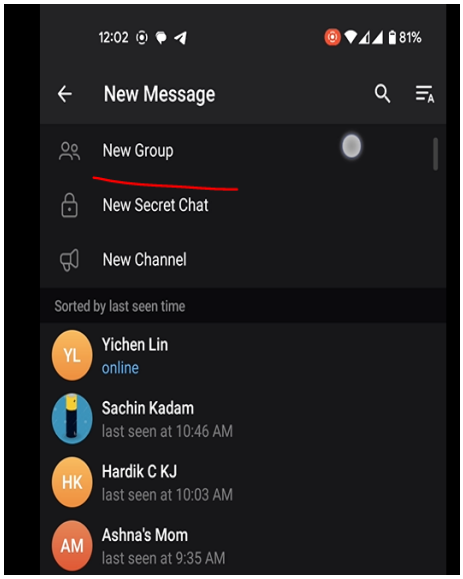


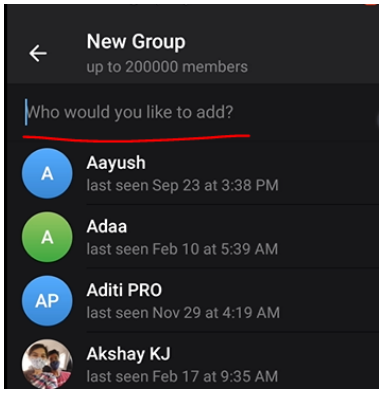

*Figure 22. Representation of the Telegram activities involved in chat*

## Group

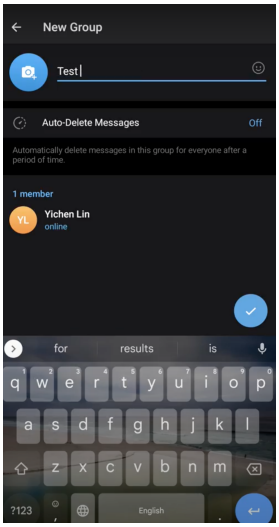
Creation of a group on the app.



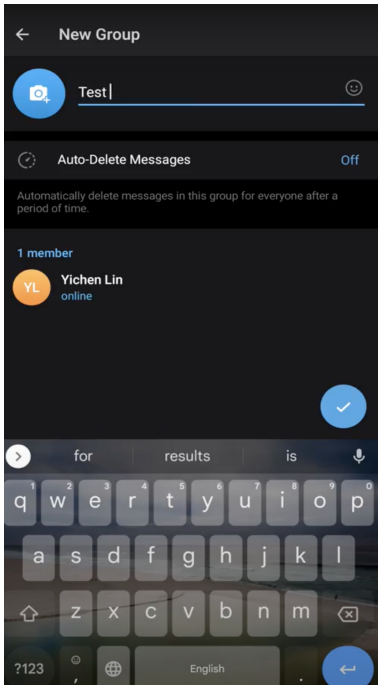

(1)



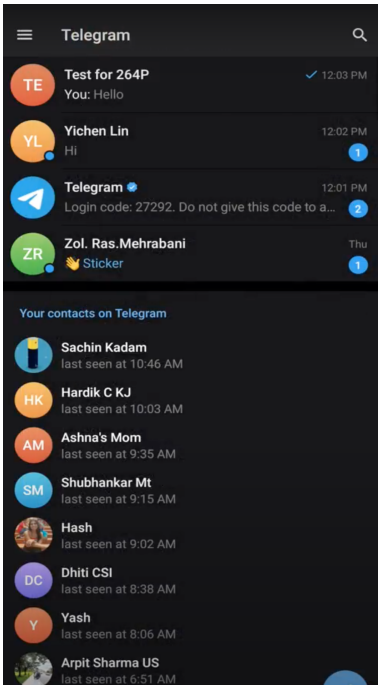

(2)




(3)




(4)




(5)

*Figure 23. Group-related functionalities*

GroupCreationActivity is the main activity that is held responsible for creating a group chat room. This activity invokes some other activities like GroupInviteActivity, IntroActivity, InviteContactsActivity etc. Group also works eventually as a chat component, invoking the previous messages in the chat, loading the users in the chat, MessageEnterTransitionContainer, MessageSeenView these components are responsible for this loading of messages in the chat room. MessageStatisticActivity Handles message stats, who sent this message and status of message after being sent.

Below is the representation of the activities involved in setting up a group chat inside the Telegram App.

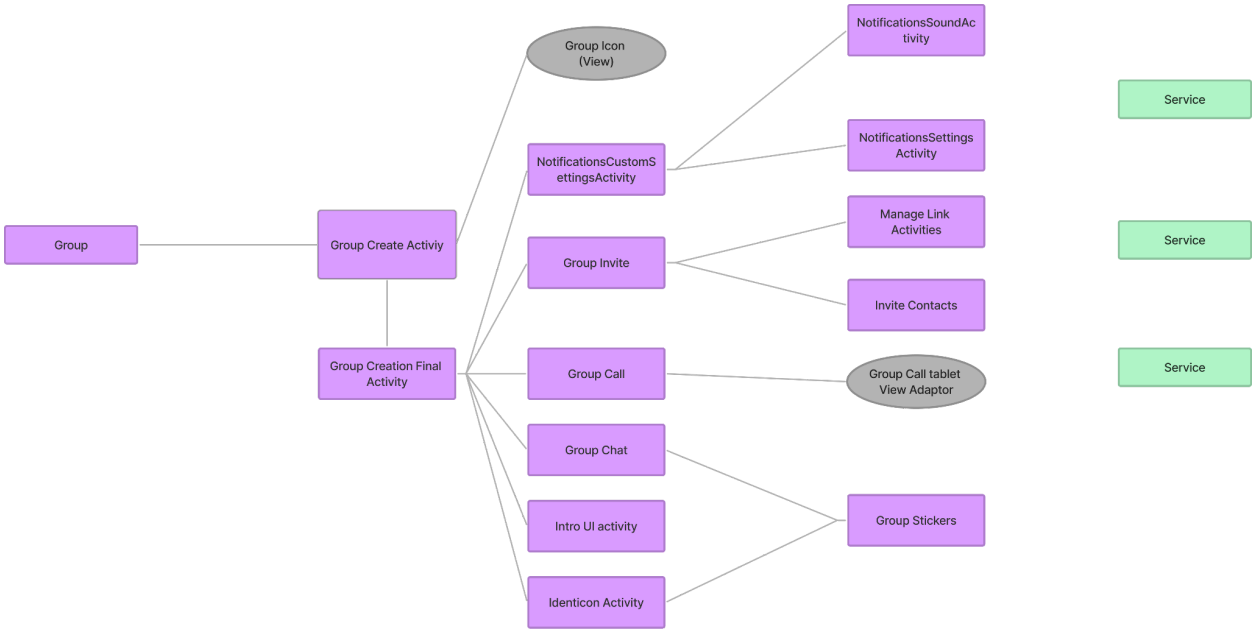



*Figure 24. Representation of the whole activities invoked during the creation of group*

# Components of Telegram CHAT Feature

For the scope of this project, we have decided to focus on the **CHATS** section in the telegram application. We have deep-dived into the 5 major functionalities presented by the chat feature. The functionalities are as follows:

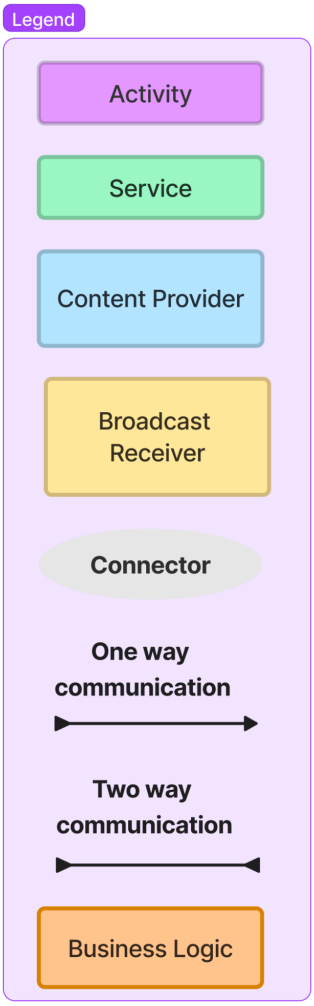



*Figure 25. Legend of the components*

## 1. Retrieving a chat

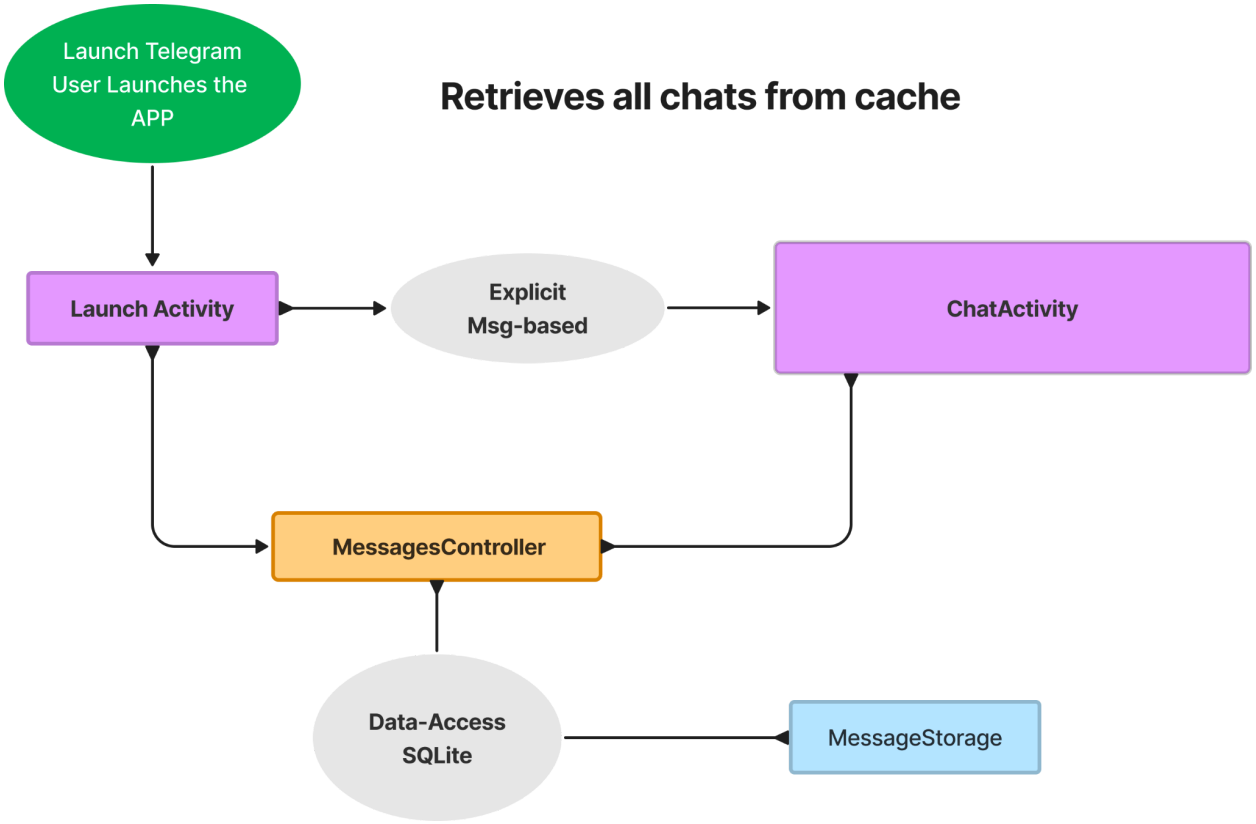



*Figure 26. Components of retrieving a chat*

When the user launches the telegram application, we can see that the app loads all the previous chats, groups and channels in the UI. The telegram app uses SQLite database to store all of the data related to chats  like the messages, media, stickers, gif (animated emojis) etc., these are inside MessageStorage. The Launch activity sends an explicit message to invoke chat activity to load the UI. Launch activity also tells MessagesController to fetch the particular chat, the controller basically has 3 important methods: **putChat**, **putChats**, **getChat**. First we call fetch users, and then call the putChat/s cause we need the right chat for the right users. Then the LaunchActivity calls getChat to load them on the UI from the MessagesController.

## 2. Sending a message

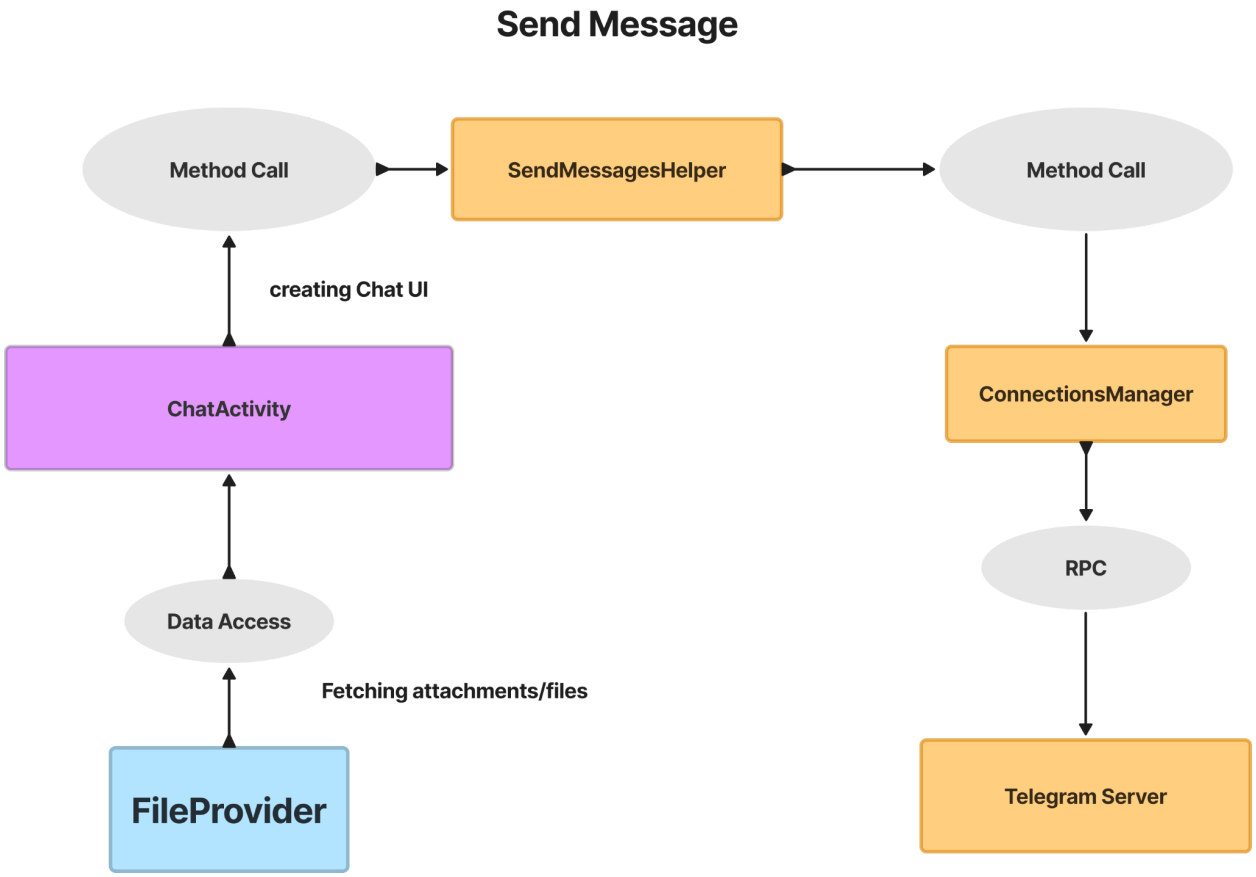



*Figure 27. Components of sending a message*

Once the application is launched, the respective chats of users are loaded from cache. Now user wants to send a message, the chatActivity makes a method call to the SendMessagesHelper like **getSendMessagesHelper().sendMessage(bunch of parameters)** this method call invokes the helper the helper also supports sending stickers, vote, games, media etc. Now SendMessagesHelper makes a method call to Connections manager which helps in relaying messages to the telegram server which sends the message on to google firebase. We have two main intents namely **MESSAGE_SEND_STATE_SENT and MESSAGE_SEND_STATE_SEND_ERROR** these intents are set in the message helper that lets the chatActivity to know if message was sent or not.

## 3. Receiving a message

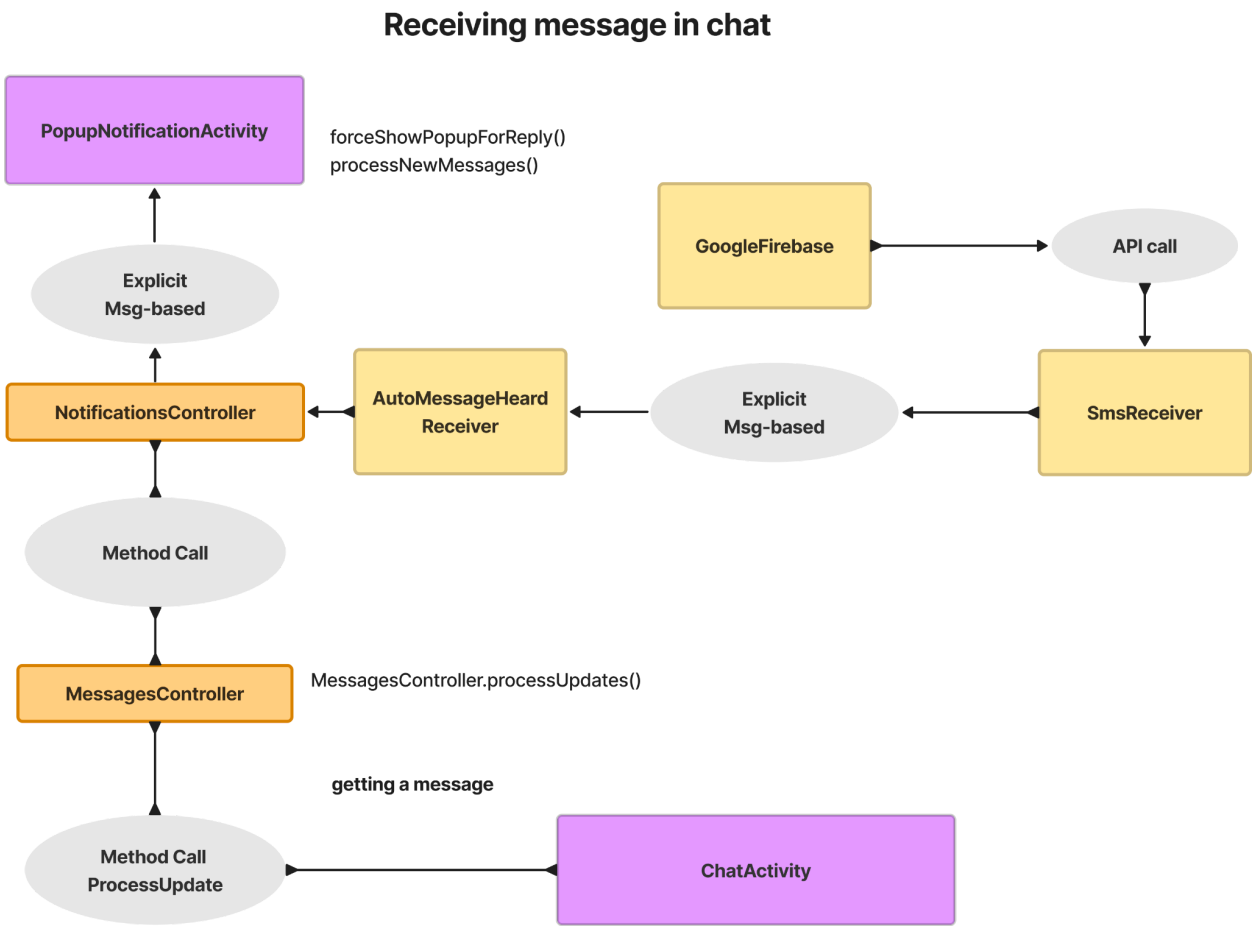



*Figure 28. Components of receiving messages*

The messages are received from the Google Firebase via an API call, the telegram has a broadcast receiver named **SMSReceiver,** when an SMS message is received and retrieved by the Google Play services, the system sends a broadcast intent with the action **com.google.android.gms.auth.api.phone.SMS_RETRIEVED**. The Receiver now invokes AutoMessageHeardReceiver that listens for the **org.telegram.messenger.ACTION_MESSAGE_HEARD** intent. When this intent is broadcasted, the "**onReceive**" method of the "**AutoMessageHeardReceiver**" class will be called to handle the intent. This sends a notification to notification controller this is the main controller of notifications with in the telegram app which makes a method call **MessagesController.processUpdates()** of MessagesController that in turn loads the new chats on the UI via ChatActivity. Now if the user is outside of the APP the telegram app shows a pop up on screen to notify the user of the new message

received. The notification controller pushes the message to **PopupNotificationActivity** which in turn displays the messages as a pop-up on the user screen.

## 4. Initiating a call

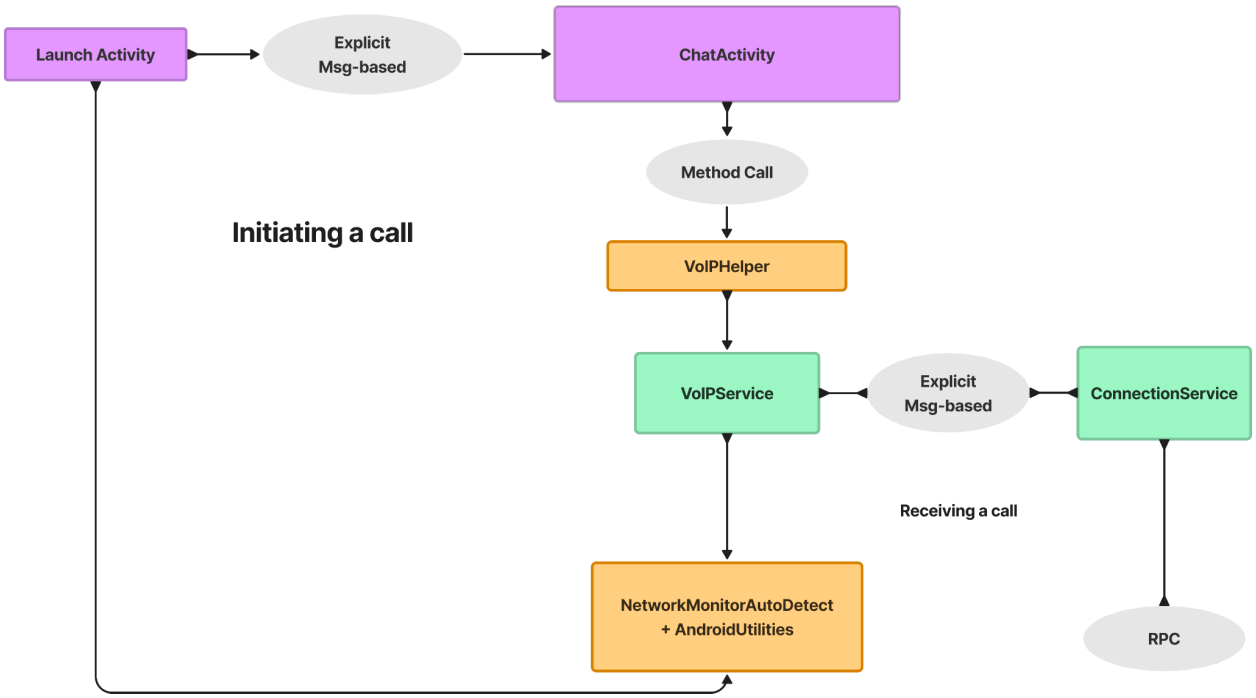



*Figure 29. Components of initiating a call*

When a user wants to make a call, he goes inside the particular chat and hits the call button. Now ChatActivity invokes VoIPHelper via a method call **VoIPHelper.startCall(bunch of parameters)** this calls the start call method inside VoIP helper which in turn calls **initiateCall and doInitiateCall** these methods interact with VoIP service, which is a foreground service that handles voice-over-IP (VoIP) functionality and requires permissions to access the media projection, camera, microphone, and media playback features this invokes connection service which help mock the telephony inside the telegram app.

## 5. Receiving a call

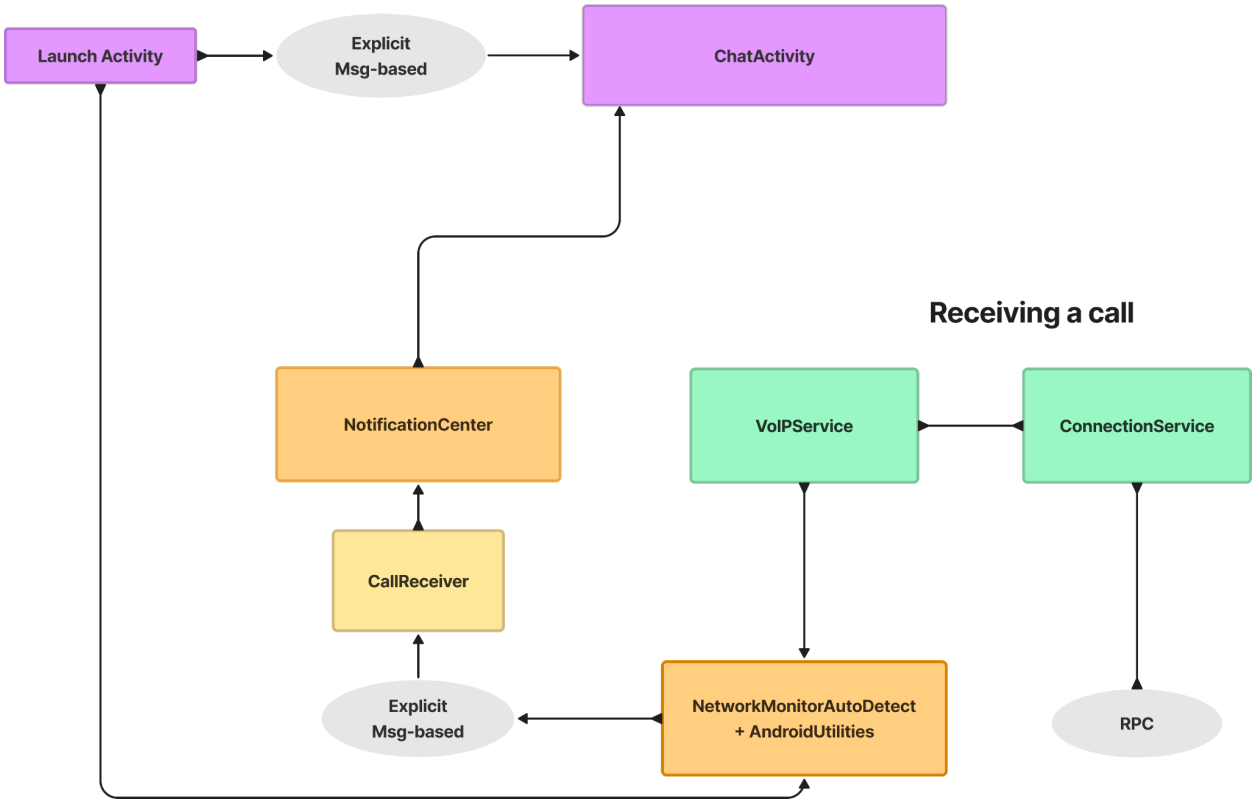



*Figure 30. Components of receiving a call*

Receiving a call is very similar to Initiating a call, the main thing here is telegram's connection service, the **ConnectionService** class provides the necessary APIs for an app to implement its own telephony service that can interact with the native phone app. In this case, the service is being used to handle VoIP functionality within the Telegram app. This service class sends the notification to VOIP service, On telegram's launch activity the NetworkMonitorAutoDetect and AndroidUtilities java classes are set; these classes are responsible for the network operations and registering/ unregistering the receivers respectively the AndroidUtilities has a **setWaitingForCall** method that is called in the **CallReciever** receiver class via an explicit intent **ACTION_PHONE_STATE_CHANGED.** The receiver then sends a notification to the notification center which helps notify the UI ChatActivity.

# Consolidated Architecture Graph

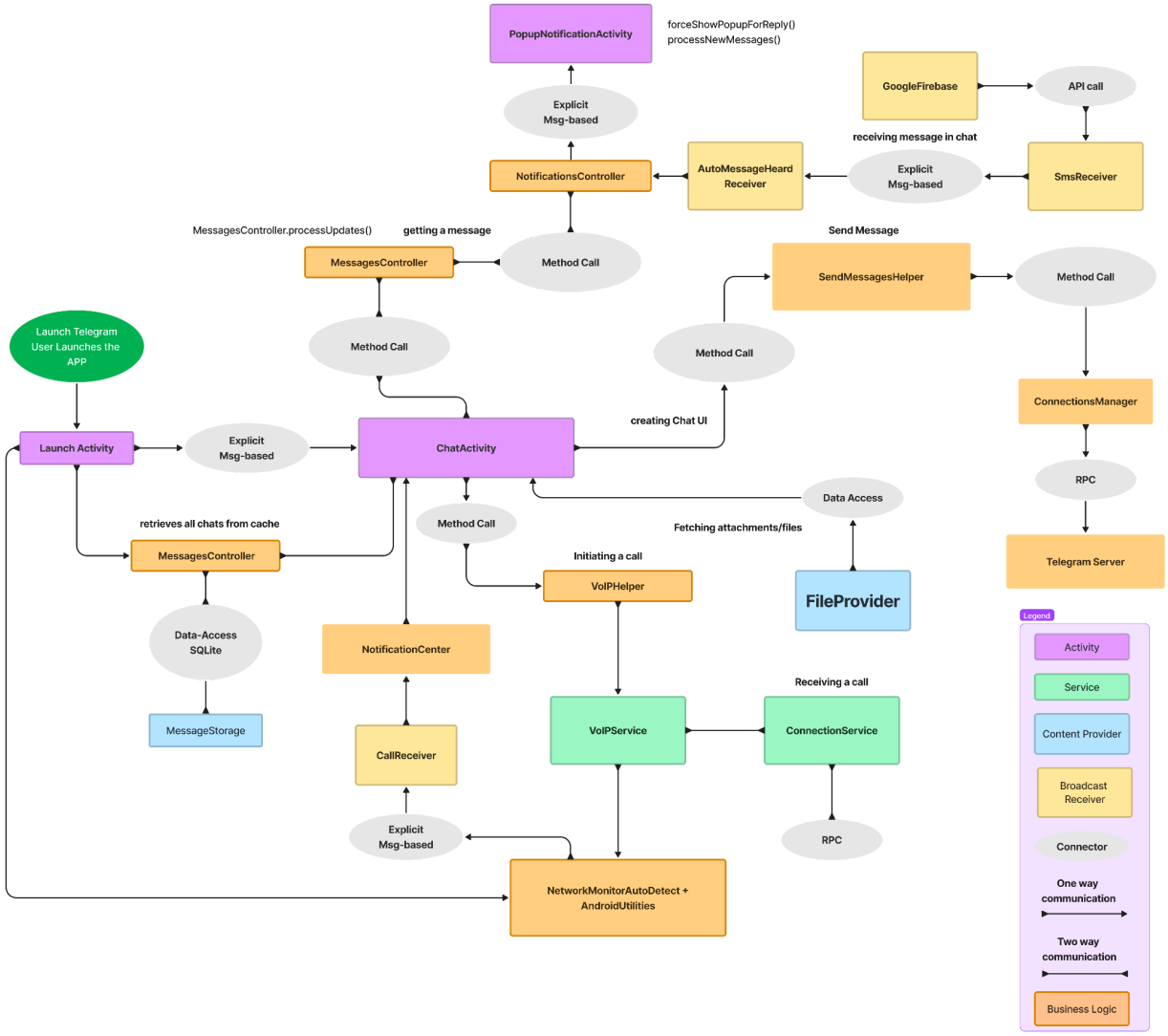



*Figure 31. Consolidated architecture graph of Chat feature*

Android components explored as part of chat feature:

**Content Provider:**

SQLite Database is the primary data storage for the chats functionality which is accessed through the the medium of a MessagesStorage which acts as a ContentProvider.

**Broadcast Receivers:**

There are 20 receivers specified in the AndroidManifest file.
The ones in focus for the chat section are:

- **AutoMessageHeardReceiver**:

This code defines a BroadcastReceiver in an Android app that listens for the "org.telegram.messenger.ACTION_MESSAGE_HEARD" intent. When this intent is broadcasted, the "onReceive" method of the ".AutoMessageHeardReceiver" class will be called to handle the intent.
In other words, this code sets up a listener for a specific intent sent by the Telegram Messenger app, which will trigger some action defined in the "AutoMessageHeardReceiver" class when the intent is received. The "android:exported" attribute is set to "false", which means that this receiver is only available to the app that defines it and cannot be accessed by other apps on the device.

- **SmsReceiver**

When an SMS message is received and retrieved by the Google Play services, the system sends a broadcast intent with the action com.google.android.gms.auth.api.phone.SMS_RETRIEVED. The BroadcastReceiver with the name .SmsReceiver declared in this code block will receive this broadcast intent, and it can perform some actions accordingly. Based on the name of the BroadcastReceiver, it is likely that it is responsible for handling incoming SMS messages in some way. The ability to receive SMS messages is a sensitive permission, and hence it is important to use it judiciously and ensure that the app is not misused.

- **CallReceiver**

When the phone's call state changes (for example, when a call is received, dialed or ended), the system sends a broadcast intent with the action android.intent.action.PHONE_STATE. The BroadcastReceiver with the name .CallReceiver declared in this code block will receive this broadcast intent, and it can perform some actions accordingly. Based on the name of the BroadcastReceiver, it is likely that it is responsible for handling incoming or outgoing phone calls in some way. This may involve performing some action when a call is received, such as starting or stopping a recording, or displaying some information on the screen. However, without

further context or information about the code within the CallReceiver class, it is difficult to say for sure what this code block does.

**Activities:**

These were focused on in the midterm presentation. The main Activity for the chat feature is the ChatActivity.

**Services:**

We invoke two services for the calling feature of telegram.

- ConnectionService: The ConnectionService class provides the necessary APIs for an app to implement its own telephony service that can interact with the native phone app. In this case, the service is being used to handle VoIP functionality within the Telegram app.
- VoIPService: This is a foreground service that handles voice-over-IP (VoIP) functionality and requires permissions to access the media projection, camera, microphone, and media playback features.

**Connectors:**

The following connectors work together to facilitate the the different functionalities for Telegram Chat:

- Explicit messaged-based
- Remote Procedure Call (RPC)
- API Call
- Method Call

# Reference

Telegram. (n.d.) *Telegram Applications*. Retrieved February 6, 2023, from https://telegram.org/apps

Telegram. (n.d.) *Encrypted CDNs for Speed and Security*. Retrieved March 12, 2023, from https://core.telegram.org/cdn

Android for Developers. (n.d.) *Application Fundamentals*. Retrieved February 20, 2023, from https://developer.android.com/guide/components/fundamentals developing_app_like_telegram