1. INTRODUCTION

Chat Application is medium of communication between users that utilizes computer programs that allow for two-way conversations between users in real time environment. Typically, the users of android smartphone using this app will be client and the data is passed and receive through Firebase Database will act as server if we consider client-server model. Once they are friends to each other, they can communicate with one another by typing messages in a network connected environment. The user can also see all of the messages entered by his friends. To be friends with each other one of the user should send friend request to another users and that user to accept it. Users can login, register to their account and they can also update their status.

1.1. Purpose of Study

In this project, we explore the critical role chat apps play in the distribution of digital journalism today and in the future. Mobile traffic due to its portability and simplicity of news apps, chatting apps present a good opportunity for customer development and engagement.

Communication has emerged as the new trend of social on mobile devices, and the sheer size of audiences on the social chat apps is too big to consider. These type of apps also present chance to classify differences mobile traffic sources and to minimize vulnerability should WhatsApp or other platforms decrease traffic for fresh settled tools similar to them. While new comers generally indicated optimist and excitement for their work on communicating apps, nearly all targeted out that as an industry we are still in an early, exploratory phase. Most major chatting apps expended the last few years achieving their user experience, only lately turning their attention to media-owner corporations. By that in mind, we need readers to not only learn from the case trainings obtainable, but also to initiate experiments of their own to find the right plan for any reporting team.

1.2. Problem Statement

- This project is to create the chat application with a server which will enable the users to chat with each other
- To build instant messaging solution to allow users to flawlessly connect with each other.
- To make this project easy to navigate from one activity to another activity.
- To develop timing information like when user was online.
- To ensure the security that the message only delivered to the target person.
- Encryption service for each message.
- Offline storing message for loading faster.
- Compressing the image for performance issue.

2. Hardware and software requirment

Hardware System Configuration:

Processor	- Intel Core i3/i5/i7
Speed	- 1.6 GHz
RAM	- 4GB (min)
Hard Disk	- 100 GB (min)

Software System Configuration:

Operating System	- Windows / Ubuntu/Mac
Programming Language	- Java, XML
Compiler for windows OS	– Android Studio

Client requirement:

Any android mobile device with any version

3. Project Design

3.1. Architecture:

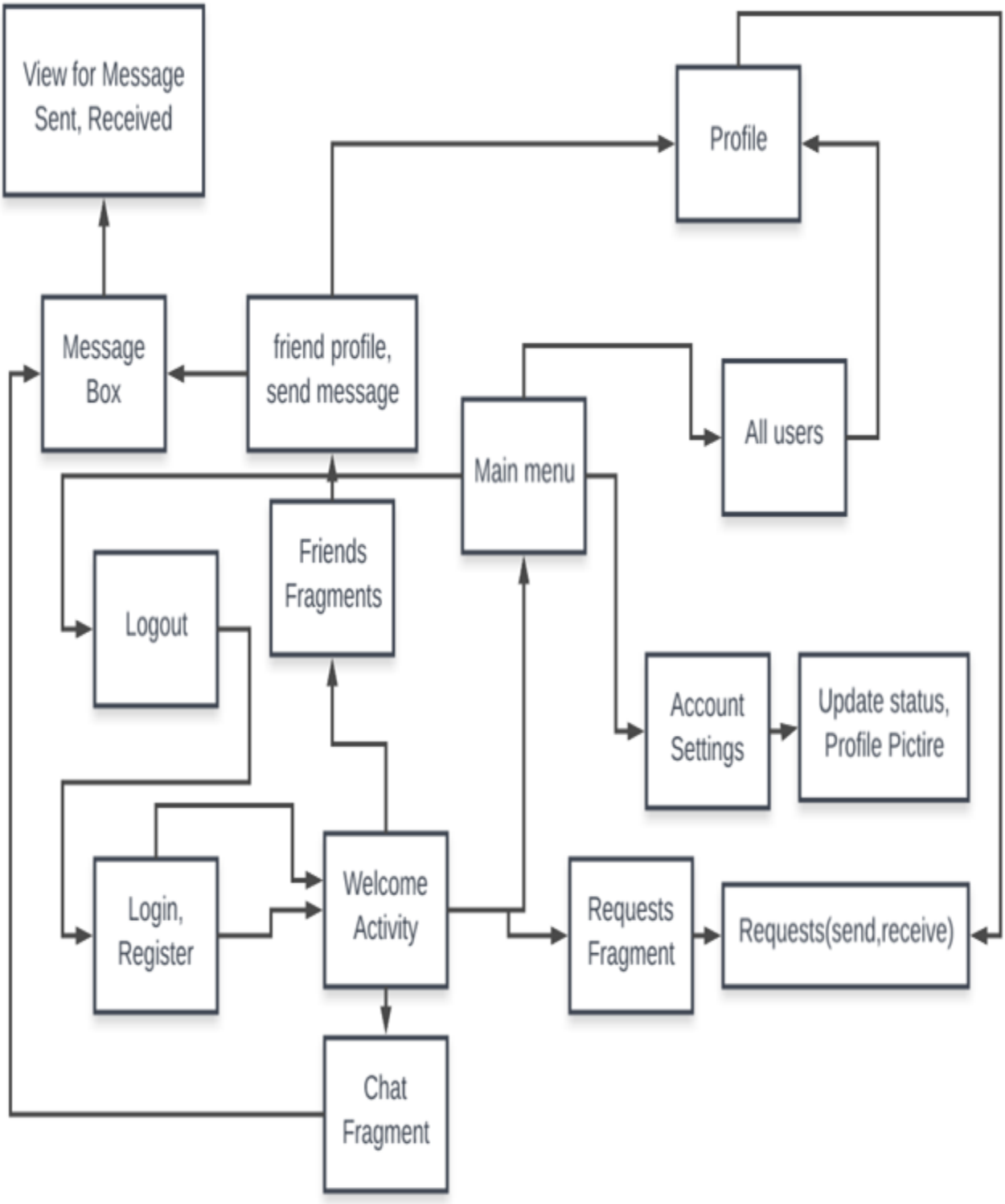


Fig 1 Architecture Diagram

This Messenger application would be able on a Server-Client building inside network. The Client– server design is service which allows task divides between the suppliers of source s, then send request to clients. The Server Side would be a continuously running service listening to the different Clients asking its services. The chat application would be connected on every chatting client. A Database of users would be maintained by the Server. When a client login to the application, the Server authenticates the user of the client android device. Once the user is authenticated the mail of the client is registered to the Server and it sends the list of connected user friends and other applicable information to the Client. When the user wishes to chat to some other user, his message along to name and time will send to receiver. Once are friends to each other, they can communicate with one another by typing messages in a network connected environment. The user can also see all of the messages entered by his friends. To be friends with each other one of the user should send friend request to another users and that user to accept it. Users can login, register to their account and they can also update their status.

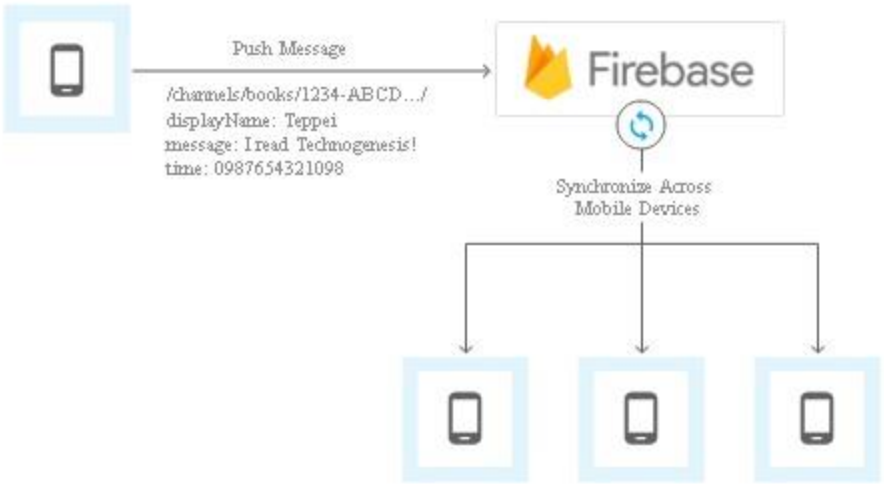


Fig 2. Firebase as a data warehouse and process tool

3.2. Algorithm

Step 1: The app run welcome page and then go to start page.

Step 2: in start page it will display login or register buttons.

Step 3: If login selected then displays main page it will display 3 fragments(friend requests, messages, users) and menu and jump to step 5 else go to step 4.

Step 4: If register button is clicked then it will store all information to database and goes to step 3.

Step 5: In menu bar

-if “Logout” is selected it will erase the login information and goes to step 2.
-if “All users” is selected then it will display all users
-if “Account setting” Is selected then it will goes to setting page.

In Fragments

-If “Friends fragment” slides it will display the friends request and sent users
-If “Message fragment” is slides it will display the message
-If “Friends fragments” is selected it will show all online and offline friends.
if any user list is selected it will go to step 6

Step 6: In Friends profile.one user can send request, cancel request, accept request and decline request.

3.3. Data model

The Firebase Realtime Database is a cloud-based data storage and processing database. Information is put in storage as JSON and synchronized in real-time to every associated client. Once you figure cross-platform apps with our Smart Phone, and JavaScript SDKs, all of our clients share Realtime Database instance then spontaneously receive apprises with the latest data.

Data structure of this project

```
users
{
  "Friend_Requests" : {
    "sender_id" : {
      "receiver_id" : {
        "request_type" : "sent"
      }
    },
  },
  "Friends" : {
    "user_id" : {
      "friend_id" : {
        "date" : "22-March-2018"
      },
        },
  "Messages" : {
    "user_id" : {
      "receiver_id" : {
        "-L8InfN6tb4hM_JZeFYa" : {
          "message" : "jgnnqakacoaucplc0  ",
          "seen" : false,
          "time" : 1521824479260,
          "type" : "text"
        },
        "-LAnjEsNNf5DjdzaaSWn" : {
          "message" : "jk\n",
          "seen" : false,
          "time" : 1524507673810,
          "type" : "text"
        }
      },
        "Users" : {
    "user_id" : {
      "online" : true,
      "user_image" : "";
      "user_name" : "Arbind Saraf",
      "user_status" : "Hey, I am using powerful Message App",
      "user_thumb_image" :
  }
}
```

3.4. User Interface Model

ConstraintLayout permits you to generate large and complex layouts with a flat view order (no nested view groups). It's alike to RelativeLayout in that all opinions are laid out according to relations between sibling views in addition the parent layout, but it's more elastic than RelativeLayout and cooler to use with Android Studio's Layout Editor.

All the control of ConstraintLayout is available straight from the Layout Editor's graphic tools, as the layout API and the Layout Editor were specifically built for each other. So you can figure your layout with ConstraintLayout totally by drag-and-dropping in its place of editing the XML.

ConstraintLayout is accessible in an API public library that's well-suited with Android 2.3 (API level 9) and greater. This sheet delivers a monitor to construction a layout with ConstraintLayout in Android Studio 3.0 or higher. If you'd like more info about the Layout Editor himself, see the Android Studio monitor to form a UI with Layout Editor.

To describe a view's location in ConstraintLayout, you necessity add at minimum one horizontal and one vertical constraint on behalf of the view. Each constraint represents a joining or alignment to additional view, the parent layout, or an imperceptible guideline. Each constraint describes the view's position along also the vertical or horizontal axis; so each view must have a minimum of one constraint for each axis, but often more are necessary.

When you drip a view into the Layout Editor, it stays somewhere you leave it even if it has nope constraints. Though, this is simply to create editing cooler; if a view has no restrictions when you track your layout on a device.

In figure 3, the layout looks decent in the editor, however there's not any vertical restriction on view C. When this layout draws on a android device, view C flat aligns with the left and right ends of view A, but seems at the top of the screen since it has no vertical constraint.

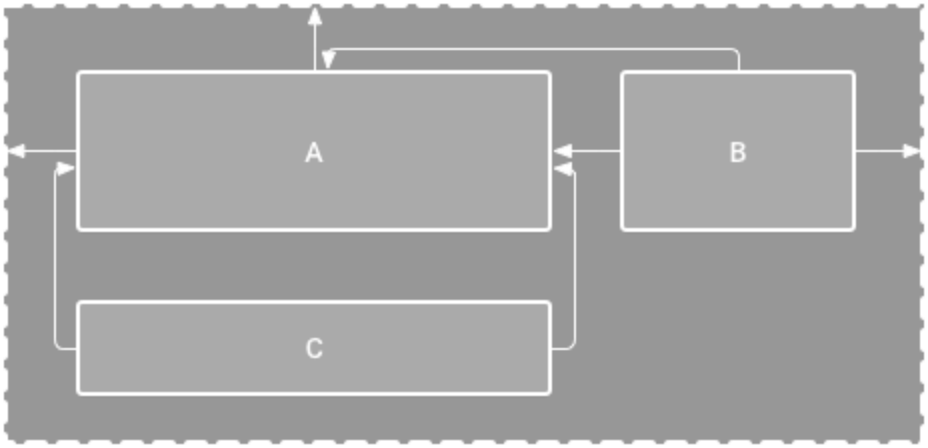


Figure 3. The editor shows view C below A, but it has no vertical constraint

4. Programing concept

4.1. List of APIs used in App

- **Firebase APIs --**
  implementation 'com.google.firebase:firebase-auth:11.0.4'
  implementation 'com.google.firebase:firebase-database:11.0.4'
  implementation 'com.google.firebase:firebase-storage:11.0.4'
  implementation 'com.firebaseui:firebase-ui-database:2.3.0'

- **Cirricular Image View** --'de.hdodenhof:circleimageview:2.1.0'
- **Cropping image** --'com.theartofdev.edmodo:android-image-cropper:2.4.+'
- **Picasso image utilities** --'com.squareup.picasso:picasso:2.5.2'
- **Image Compressor** --'id.zelory:compressor:2.1.0'
- **Offline Features** 'com.squareup.okhttp:okhttp:2.5.0'

4.2. Important Code Snippets:

Firebase

Initialize firebase variables

```
private DatabaseReference mFirebaseDatabaseReference;
private FirebaseRecyclerAdapter<FriendlyMessage, MessageViewHolder>
        mFirebaseAdapter;
```

reading from database

```
mFirebaseDatabaseReference = FirebaseDatabase.getInstance().getReference();
SnapshotParser<FriendlyMessage> parser = new SnapshotParser<FriendlyMessage>() {
            @Override
            public FriendlyMessage parseSnapshot(DataSnapshot dataSnapshot) {
                FriendlyMessage friendlyMessage =
dataSnapshot.getValue(FriendlyMessage.class);
                if (friendlyMessage != null) {
                    friendlyMessage.setId(dataSnapshot.getKey());
                }
                return friendlyMessage;
            }
        };

DatabaseReference messagesRef = mFirebaseDatabaseReference.child(MESSAGES_CHILD);
        FirebaseRecyclerOptions<FriendlyMessage> options =
                new FirebaseRecyclerOptions.Builder<FriendlyMessage>()
                        .setQuery(messagesRef, parser)
                        .build();
mFirebaseAdapter = new FirebaseRecyclerAdapter<FriendlyMessage,
MessageViewHolder>(options) {
            @Override
            public MessageViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
```

```
                LayoutInflater inflater = LayoutInflater.from(viewGroup.getContext());
                return new MessageViewHolder(inflater.inflate(R.layout.item_message,
viewGroup, false));
            }

            @Override
            protected void onBindViewHolder(final MessageViewHolder viewHolder,
                                            int position,
                                            FriendlyMessage friendlyMessage) {
        mProgressBar.setVisibility(ProgressBar.INVISIBLE);
if (friendlyMessage.getText() != null) {
    viewHolder.messageTextView.setText(friendlyMessage.getText());
    viewHolder.messageTextView.setVisibility(TextView.VISIBLE);
    viewHolder.messageImageView.setVisibility(ImageView.GONE);
} else {
    String imageUrl = friendlyMessage.getImageUrl();
    if (imageUrl.startsWith("gs://")) {
        StorageReference storageReference = FirebaseStorage.getInstance()
                .getReferenceFromUrl(imageUrl);
        storageReference.getDownloadUrl().addOnCompleteListener(
                new OnCompleteListener<Uri>() {
            @Override
            public void onComplete(@NonNull Task<Uri> task) {
                if (task.isSuccessful()) {
                    String downloadUrl = task.getResult().toString();
                    Glide.with(viewHolder.messageImageView.getContext())
                            .load(downloadUrl)
                            .into(viewHolder.messageImageView);
                } else {
                    Log.w(TAG, "Getting download url was not successful.",
                            task.getException());
                }
            }
        });
    } else {
        Glide.with(viewHolder.messageImageView.getContext())
                .load(friendlyMessage.getImageUrl())
                .into(viewHolder.messageImageView);
    }
    viewHolder.messageImageView.setVisibility(ImageView.VISIBLE);
    viewHolder.messageTextView.setVisibility(TextView.GONE);
}
```

```
viewHolder.messengerTextView.setText(friendlyMessage.getName());
if (friendlyMessage.getPhotoUrl() == null) {

viewHolder.messengerImageView.setImageDrawable(ContextCompat.getDrawable(MainActivity.thi
s,
            R.drawable.ic_account_circle_black_36dp));
} else {
    Glide.with(MainActivity.this)
            .load(friendlyMessage.getPhotoUrl())
            .into(viewHolder.messengerImageView);
}

    }
};

mFirebaseAdapter.registerAdapterDataObserver(new RecyclerView.AdapterDataObserver() {
    @Override
    public void onItemRangeInserted(int positionStart, int itemCount) {
        super.onItemRangeInserted(positionStart, itemCount);
        int friendlyMessageCount = mFirebaseAdapter.getItemCount();
        int lastVisiblePosition =
                mLinearLayoutManager.findLastCompletelyVisibleItemPosition();
        // If the recycler view is initially being loaded or the
        // user is at the bottom of the list, scroll to the bottom
        // of the list to show the newly added message.
        if (lastVisiblePosition == -1 ||
                (positionStart >= (friendlyMessageCount - 1) &&
                        lastVisiblePosition == (positionStart - 1))) {
            mMessageRecyclerView.scrollToPosition(positionStart);
        }
    }
});

mMessageRecyclerView.setAdapter(mFirebaseAdapter);
```

Appropriately start and stop listening for updates from Firebase Realtime Database. Update the *onPause* and *onResume* methods in MainActivity as shown below.

MainActivity.java

```
@Override
public void onPause() {
    mFirebaseAdapter.stopListening();
    super.onPause();
```

```
}

@Override
public void onResume() {
    super.onResume();
    mFirebaseAdapter.startListening();
}
```

Storing into Database

MainActivity.java

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   Log.d(TAG, "onActivityResult: requestCode=" + requestCode + ", resultCode=" +
resultCode);

   if (requestCode == REQUEST_IMAGE) {
       if (resultCode == RESULT_OK) {
           if (data != null) {
               final Uri uri = data.getData();
               Log.d(TAG, "Uri: " + uri.toString());

               FriendlyMessage tempMessage = new FriendlyMessage(null, mUsername,
mPhotoUrl,
                       LOADING_IMAGE_URL);
               mFirebaseDatabaseReference.child(MESSAGES_CHILD).push()
                       .setValue(tempMessage, new DatabaseReference.CompletionListener()
{
                           @Override
                           public void onComplete(DatabaseError databaseError,
                                                  DatabaseReference databaseReference) {
                               if (databaseError == null) {
                                   String key = databaseReference.getKey();
                                   StorageReference storageReference =
                                           FirebaseStorage.getInstance()
                                           .getReference(mFirebaseUser.getUid())
                                           .child(key)
                                           .child(uri.getLastPathSegment());

                                   putImageInStorage(storageReference, uri, key);
                               } else {
                                   Log.w(TAG, "Unable to write message to database.",
```

```
                                databaseError.toException());
                        }
                    }
                });
            }
        }
    }
}
```

Cropping image

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
  if (requestCode == CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE) {
    CropImage.ActivityResult result = CropImage.getActivityResult(data);
    if (resultCode == RESULT_OK) {
      Uri resultUri = result.getUri();
    } else if (resultCode == CropImage.CROP_IMAGE_ACTIVITY_RESULT_ERROR_CODE) {
      Exception error = result.getError();
    }
  }
}
```

Compressing image

Compressor is a lightweight and dominant android photo compression package. Compressing tools or API helps in compressing big-sized photos into smaller sized photos with very less or small damage in the meaning of the photo.

```
compressedImage = new Compressor(this)
            .setMaxWidth(640)
            .setMaxHeight(480)
            .setQuality(75)
            .setCompressFormat(Bitmap.CompressFormat.WEBP)

.setDestinationDirectoryPath(Environment.getExternalStoragePublicDirectory(
              Environment.DIRECTORY_PICTURES).getAbsolutePath())
            .compressToFile(actualImage);
```

OkHttp API

HTTP is the way recent applications system. It's how we interchange information & media. HTTP powerfully marks your stuff load quicker and protects bandwidth.

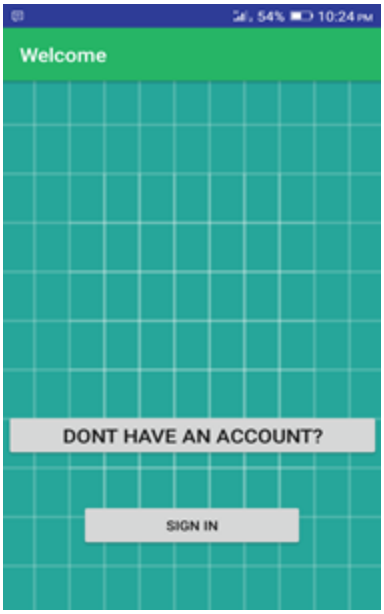
OkHttp is an HTTP client that's efficient by default:

- HTTP/2 support permits all requests to the same host to share a socket.
- Connection pooling decreases request latency (if HTTP/2 isn't available).
- Transparent GZIP shrinks transfer sizes.

OkHttp persists after the network is troublesome: it will noiselessly improve from common connection glitches. If your provision has many IP addresses OkHttp will attempt alternate addresses if the first connect flops. This is essential for IPv4+IPv6 and for services hosted in redundant information centers. OkHttp initiates new influences with up-to-date TLS structures (SNI, ALPN), and falls back to TLS 1.0 if the handshake fails.

5. Output ScreenShot

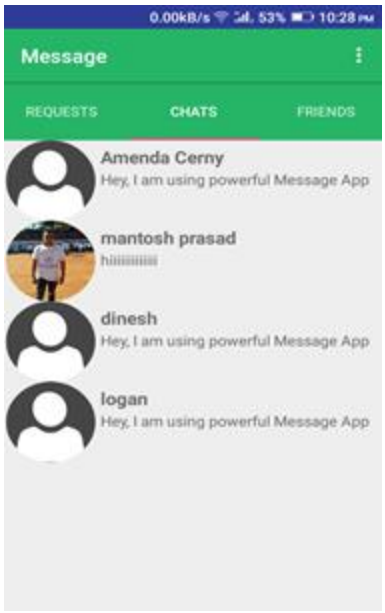
Welcome Activity



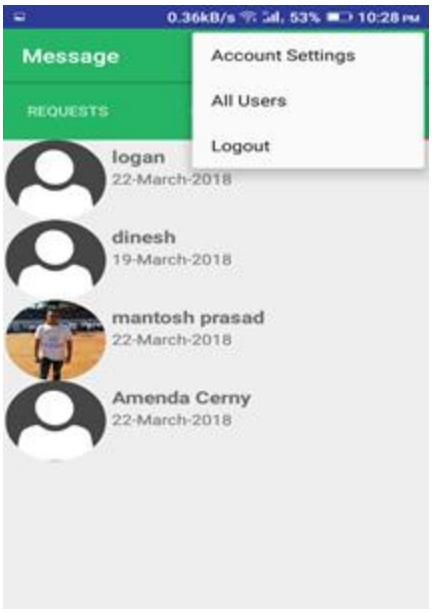
Startup Activity

Login Activity

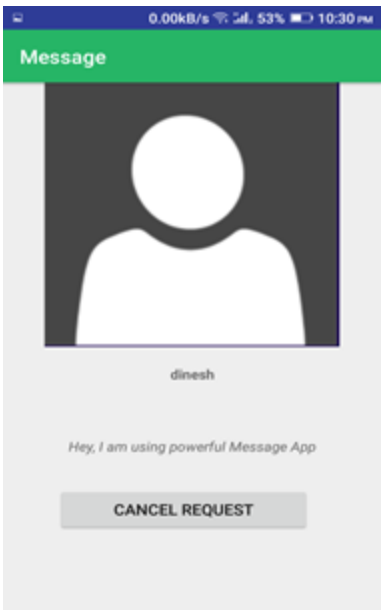
Register Activity

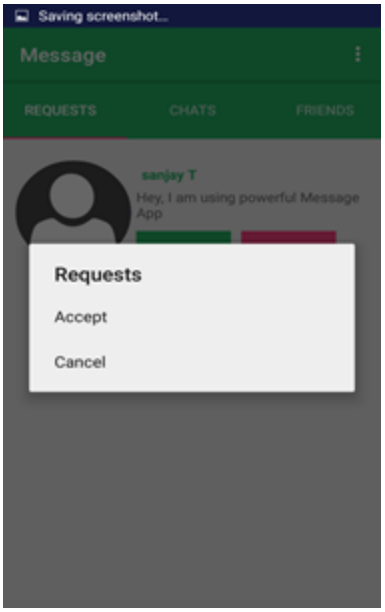


Chats Activity



Infaltor Menu





Profile Activity *Request Fragment* *Options for Request*

6. Conclusion

Chat App is a very common used application among the consumers. Common consumers use the immediate messaging services to communicate with other individual users. In our project we have provided with many enhanced features for a chat application. The features like painting along with chat would be a fun to use and interactive feature for a general user. For professional users it would be very useful for communicating important flowcharts, diagrammatic representation of some problem, making important symbols, etc. It opens up a wide variety of uses for individuals. The predictive texting feature would help a user to chat easily. Various figures of various formats could be opened and sent to other user. It would also give freedom of using any tool for drawing. The chat application is so aimed that the people could have a better experience of chatting. It has the potential to attract more and more users to interact and connect. It also ensures that the message only sent to target user in encrypted manner and provide user with experience like easy navigation, performance and secure.

Reference

- The following books have helped me in my project:-

1. Effective Java (2nd edition) by Joshua Bloch

- The following websites links have been referred to for the project:-

1. https://developers.google.com/training/android/
2. https://www.tutorialspoint.com/android/index.htm
3. https://en.wikipedia.org/wiki/Android_software_development
4. https://www.mindinventory.com/android-application-development.php
5. https://www.springboard.com/learning-paths/android/
6. https://www.udemy.com/complete-android-developer-course/