॥वसुधैव कुटुम्बकम्॥

Symbiosis Institute of Technology

A Project Report on

SANVAAD

Submitted by:

Safeer Khan	**18070122033**
Priyanshu Meena	**18070122045**
Nachiket Sahare	**18070122055**

Under the Guidance of

Dr. Preeti Mulay

Department of Computer Science

Contents

Problem Statement

Communicating with a person who has hearing, or speech disability can be very cumbersome. It requires both parties to use sign language as a medium. Historically this has worked great but has several flaws, mainly as it assumes the sender and receiver both know the sign language. This makes it necessary for friends of the person to know the language, makes it harder for these individuals to socialize; leaving them completely out of group discussions etc.

Introduction

Purpose of Document

This is a Project Report document for an android application for assisting communication with persons with hearing disability. The app is aimed to allow the user to interact with people and allow other parties to communicate naturally with the user, eliminating the need of sign language. This document describes the scope, objectives and goal of the new system. In addition to describing non-functional requirements, this document models the functional requirements with use cases, interaction diagrams, and class models. This document also includes the architecture of the application, the methodology followed, the software and hardware tools used and the detailed results acquired during the testing phase of the application.

Project Summary

Project Name: **Sanvaad**

Team Members : Safeer Khan (18070122033)

Priyanshu Meena (18070122045)

Nachiket Sahare (18070122055)

Motivation

In 2020, with almost everyone having a smartphone, this problem can be solved easily through an app. The person can use the app to listen to conversations they are part of, thus not be left out and without everyone learning and speaking in sign language. The app would also feature a text to speech to allow persons with speech disability to express themselves.

Besides persons-of-disabilities, this app can also be used for solving other communication barriers such as accents, which can be helpful to international students, tourists, etc.

Project Scope

The scope of this project is to build an android application that makes use of third-party APIs in order to achieve the required functionalities.The application provides an interface to the user to perform functions that are essential for smooth flow of the conversation with the user.

Limitations

The application is limited to the services offered by the third-party Speech APIs. The application does not have any Machine learning services built in. The application is limited to the use of open source libraries for voice functions.

The system doesn't have a dedicated backend services and shall make use of managed database and authentication services.

Functional Requirements

High Priority

- System must convert the speaker's message from voice to text
- The system must play the users message as audio from the device speaker.
- The speaker and user messages must be displayed to the user.

Medium Priority

- System should be able to add, update and delete contacts.
- The system should also be able to save and retrieve chat conversations.
- System should allow users to authenticate themselves by either logging in or registering as a new user.
- System must allow users to assign contacts to chat messages.
- Users should also be able to add contacts in a chat conversation.
- Users should be able to access a list of common messages for effective communication.

Low Priority

- Users should be able to give system feedback.
- Users can change the gender of their voice.
- Users can toggle the speech features.
- Users can edit profile details.

Non-Functional Requirements

App permissions

The application requires the following permissions to access the following resources on the device:

1. Microphone
2. Internet
3. Network state
4. Media player

Performance

- The speech-to-text result must be displayed to the user within 1500ms after the end of sentence.
- The text to speech result must be played within 800ms of user action

Privacy

- User chat data should be stored only on the user's device and not on a remote database.
- Only authenticated users should be able to access the chat data stored on the device.

Ease of Use

- The UI should be simple and easy to interact with so that users can effectively conversate and navigate between chats.
- The process of adding, updating and accessing contacts should be similar to existing experience of users with smartphones.
- Provides efficient speech to text service to the user.

API subscriptions

- A speech to text API service is required.
- A Text-to-Speech API service is required.
- A remote data store and authentication service is also needed.

Interfaces

The system must be able to interface with

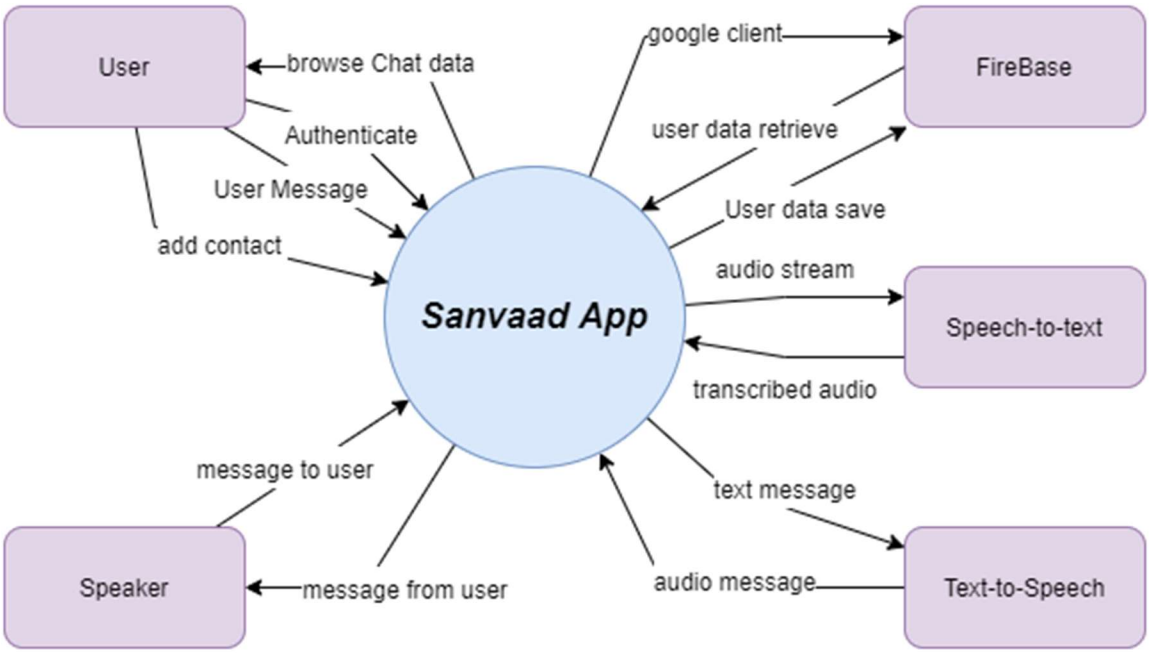
- The Speech to text api
- The text to speech api
- The remote database
- The authentication service

- The local database

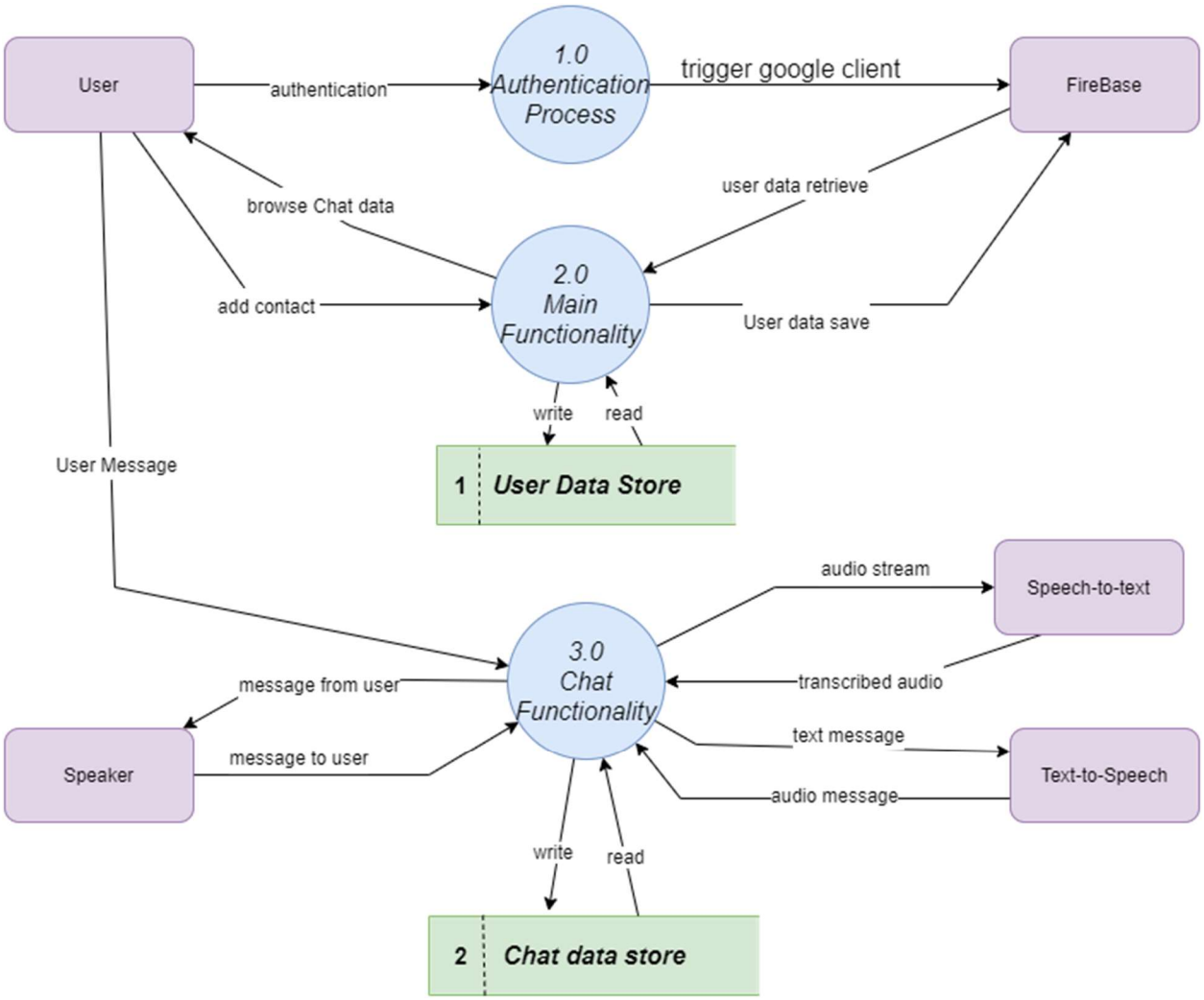
Data Flow Diagrams

We have created Three diagrams to indicate data flows. They are as follows:

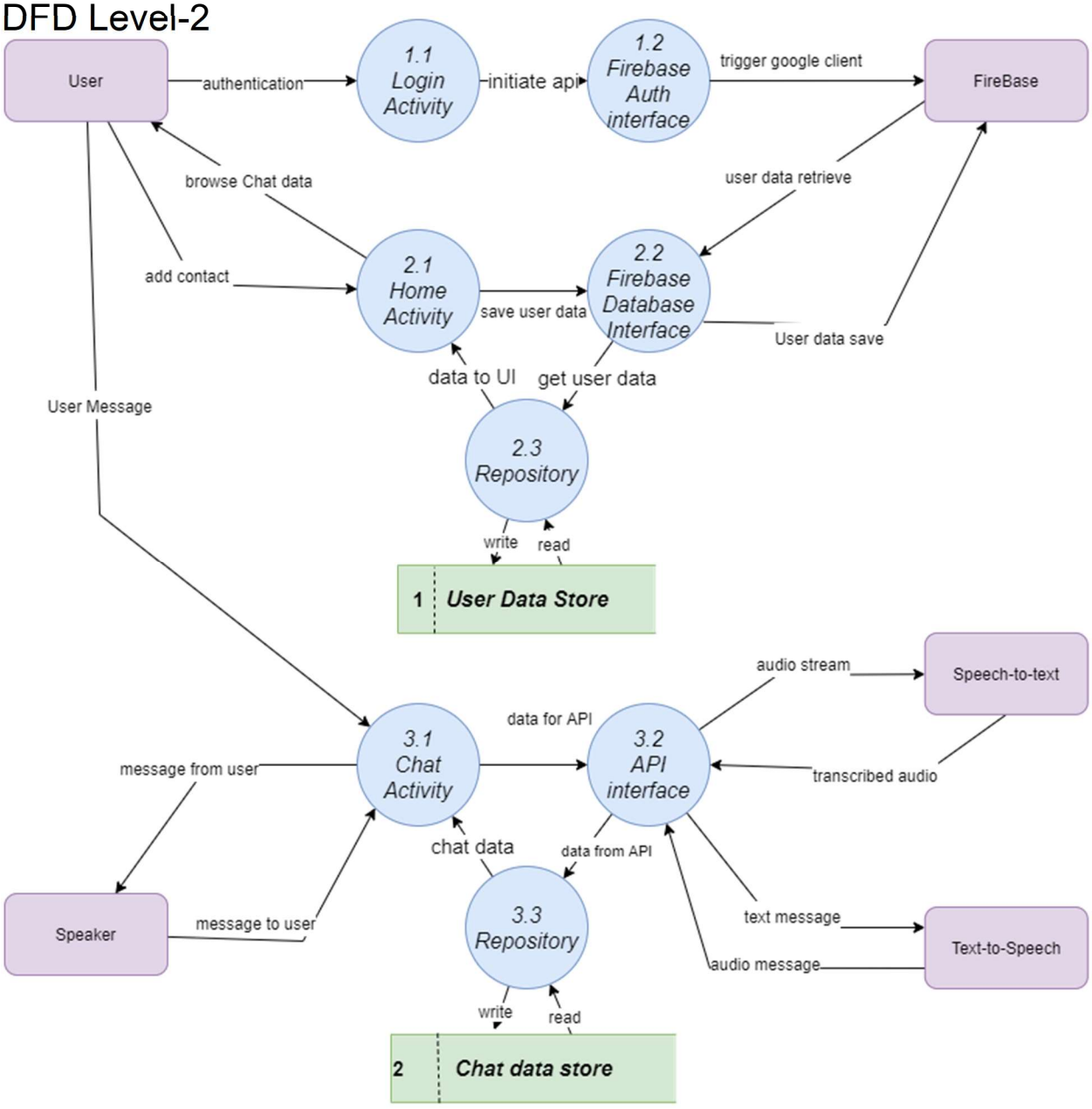
Context Diagram



DFD Level-1


User
authentication
1.0
Authentication
Process
trigger google client
FireBase
browse Chat data
user data retrieve
2.0
Main
Functionality
add contact
User data save
write
read
User Message
1
User Data Store
audio stream
Speech-to-text
3.0
Chat
Functionality
message from user
transcribed audio
text message
Speaker
message to user
Text-to-Speech
audio message
write
read
2
Chat data store

DFD Level-2


User
authentication
1.1 Login Activity
initiate api
1.2 Firebase Auth interface
trigger google client
FireBase
browse Chat data
user data retrieve
add contact
2.1 Home Activity
save user data
2.2 Firebase Database Interface
User data save
data to UI
get user data
User Message
2.3 Repository
write
read
1 User Data Store
audio stream
Speech-to-text
data for API
3.1 Chat Activity
3.2 API interface
message from user
transcribed audio
chat data
data from API
Speaker
message to user
3.3 Repository
text message
Text-to-Speech
audio message
write
read
2 Chat data store

Use Cases

The external entities interacting twitch the application are

- User
- Contact/Contacts
- Firebase
- Speech to text- API
- Text-to-Speech API

Accordingly, we have framed use cases for all interactions of these entities with the application in a tabular format.

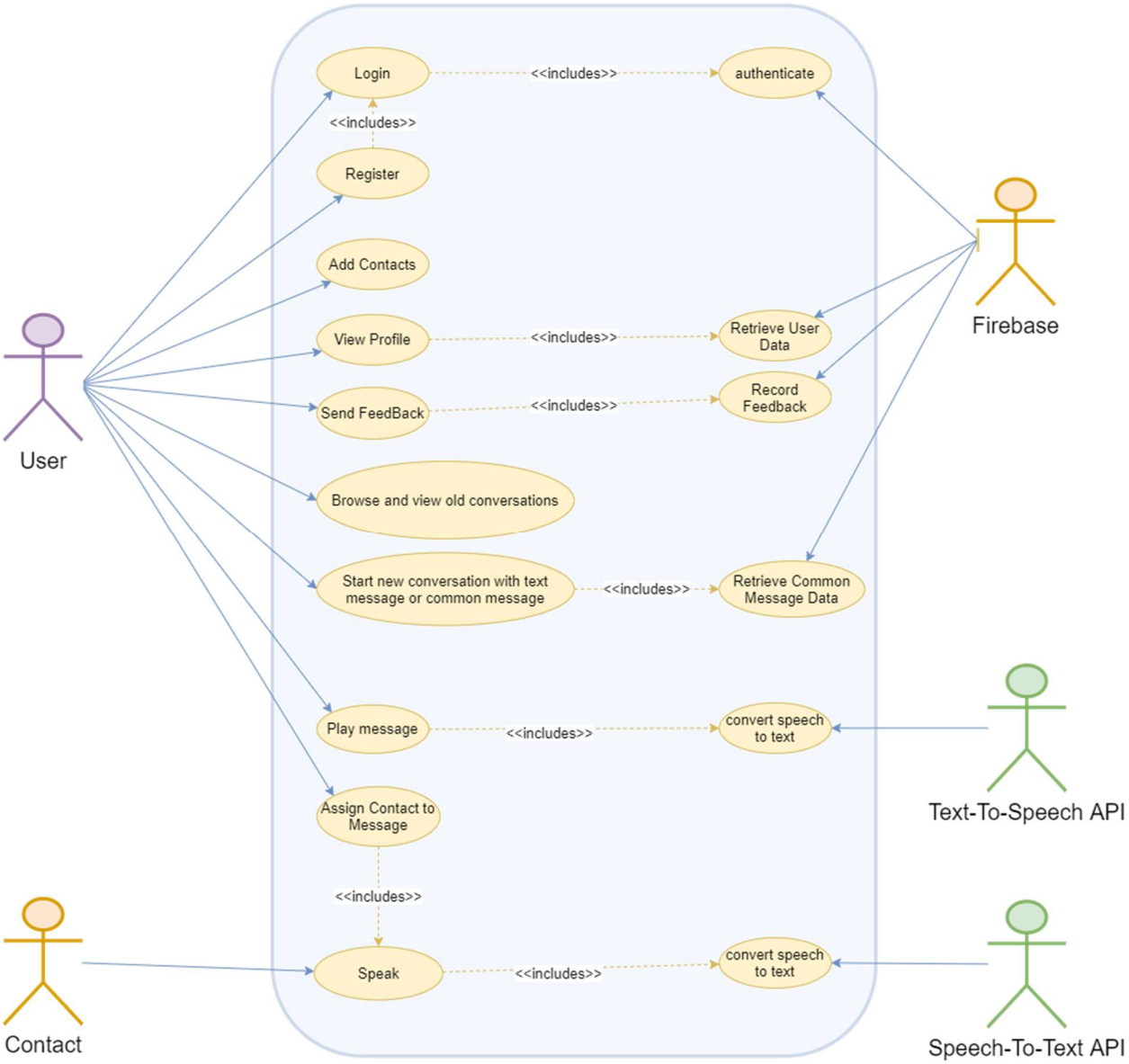
We have the following Use cases:

- Login
- Register
- Authenticate
- BrowseChats and View Chats
- Add Contacts
- View Profiles
- Send Feedback
- Retrieve User Data
- Record Feedback
- Start new chat, Enter Message or common message
- Retrieve common message Data
- Play Message
- Convert Text to Speech
- Assign Contact to Message
- Speak
- Convert Speech to Text

All these Use cases are shown in the diagram below.

The Details of all these use cases are present in APPENDIX 1

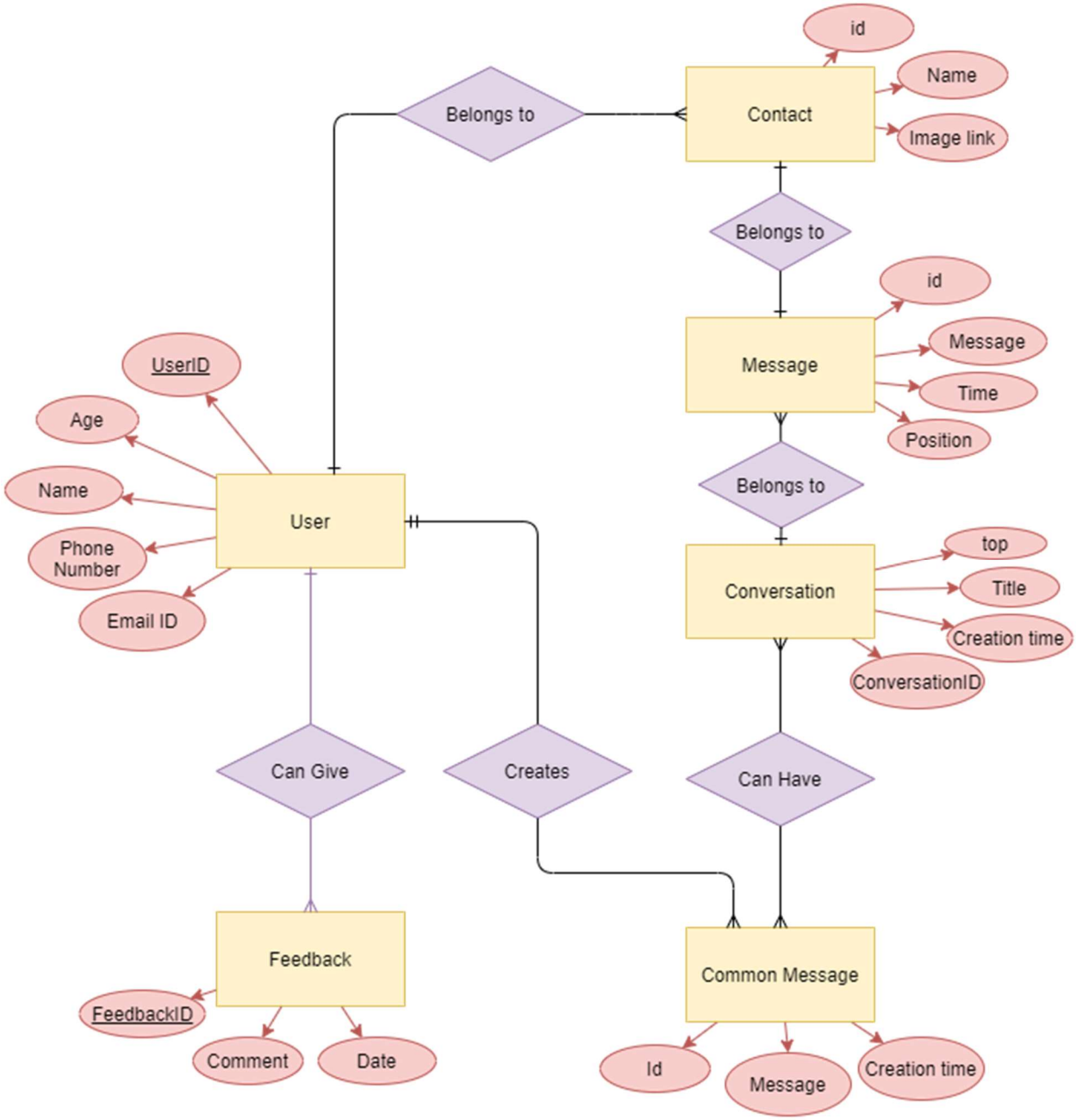
Use case Diagram



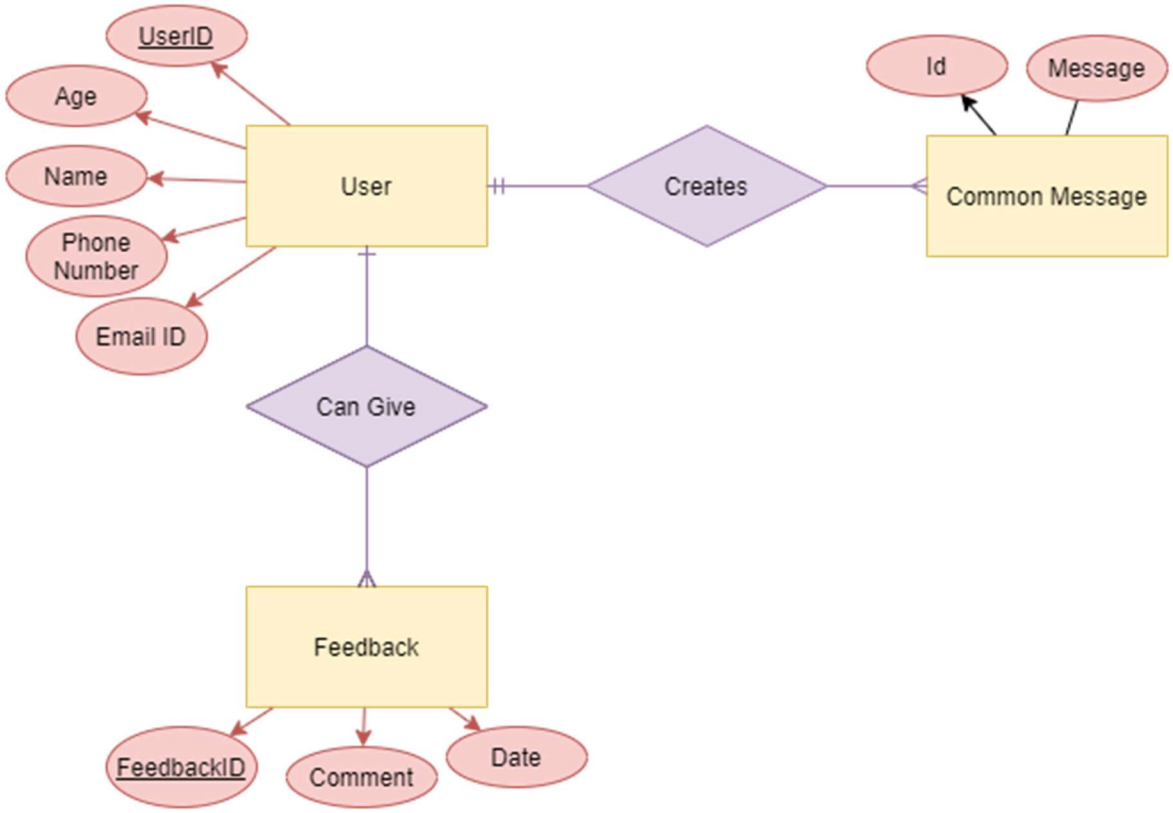
Entity Relationship Diagram

We are maintaining two databases for this application. One local database which will be storing data on the device itself. This is a Relational database called SQLite, which stores data on the android device itself.
The second database is the Remote database. For this, we make use of Firebase Realtime-database. THis is a NoSQL database, which makes use of JSON objects to store the data.
We have created the ER diagram for both the data bases as follows.

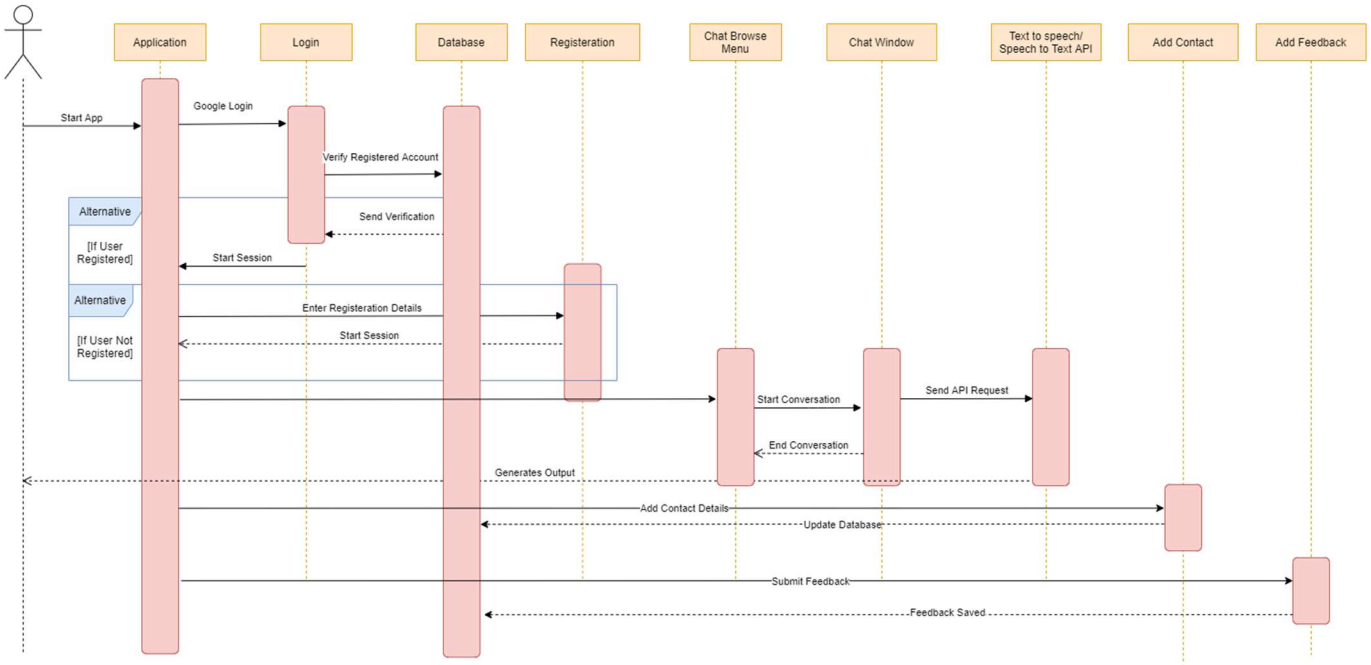
The Local Database (SQLite).



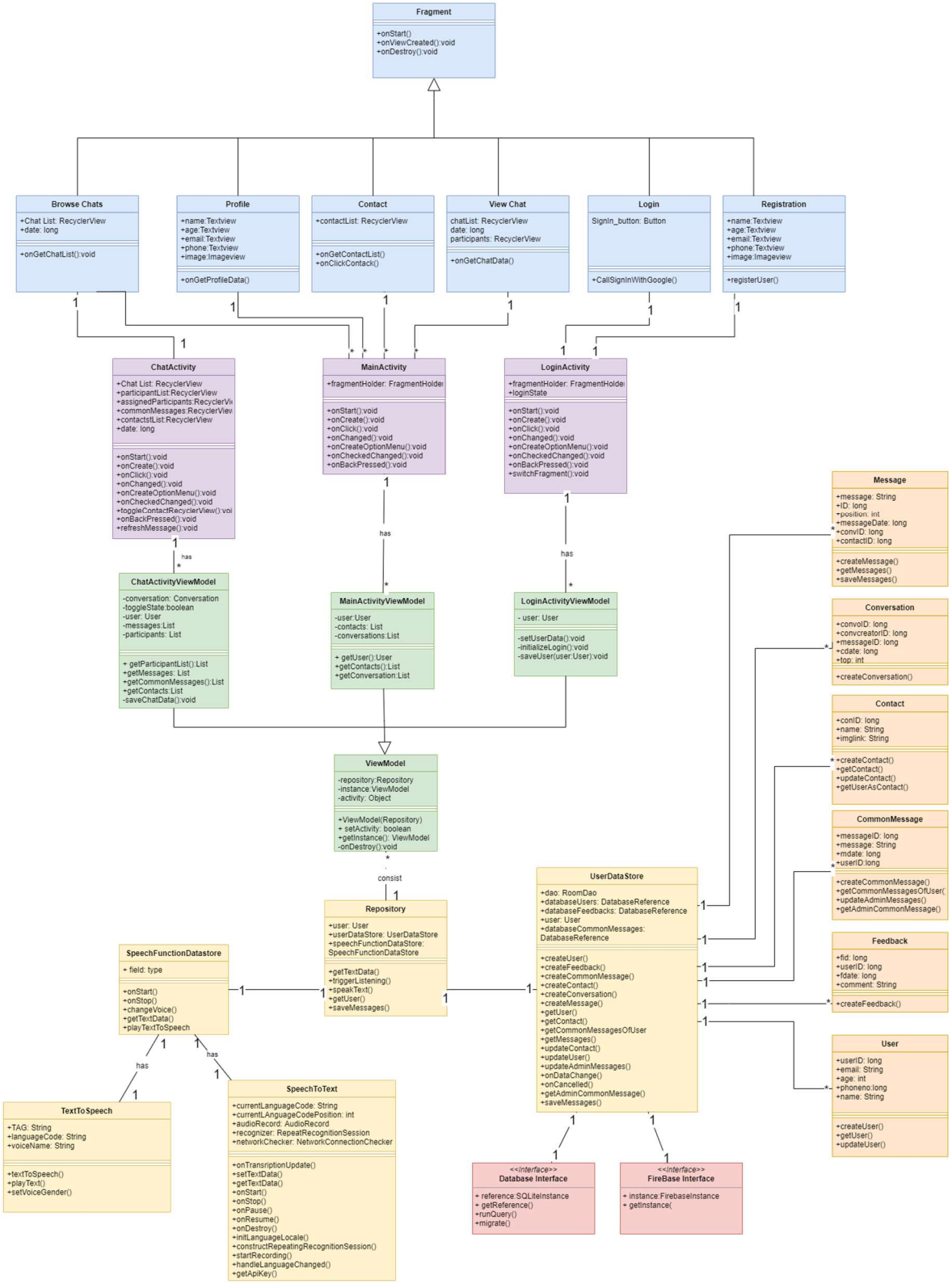
The Remote Database (Firebase NoSQL)



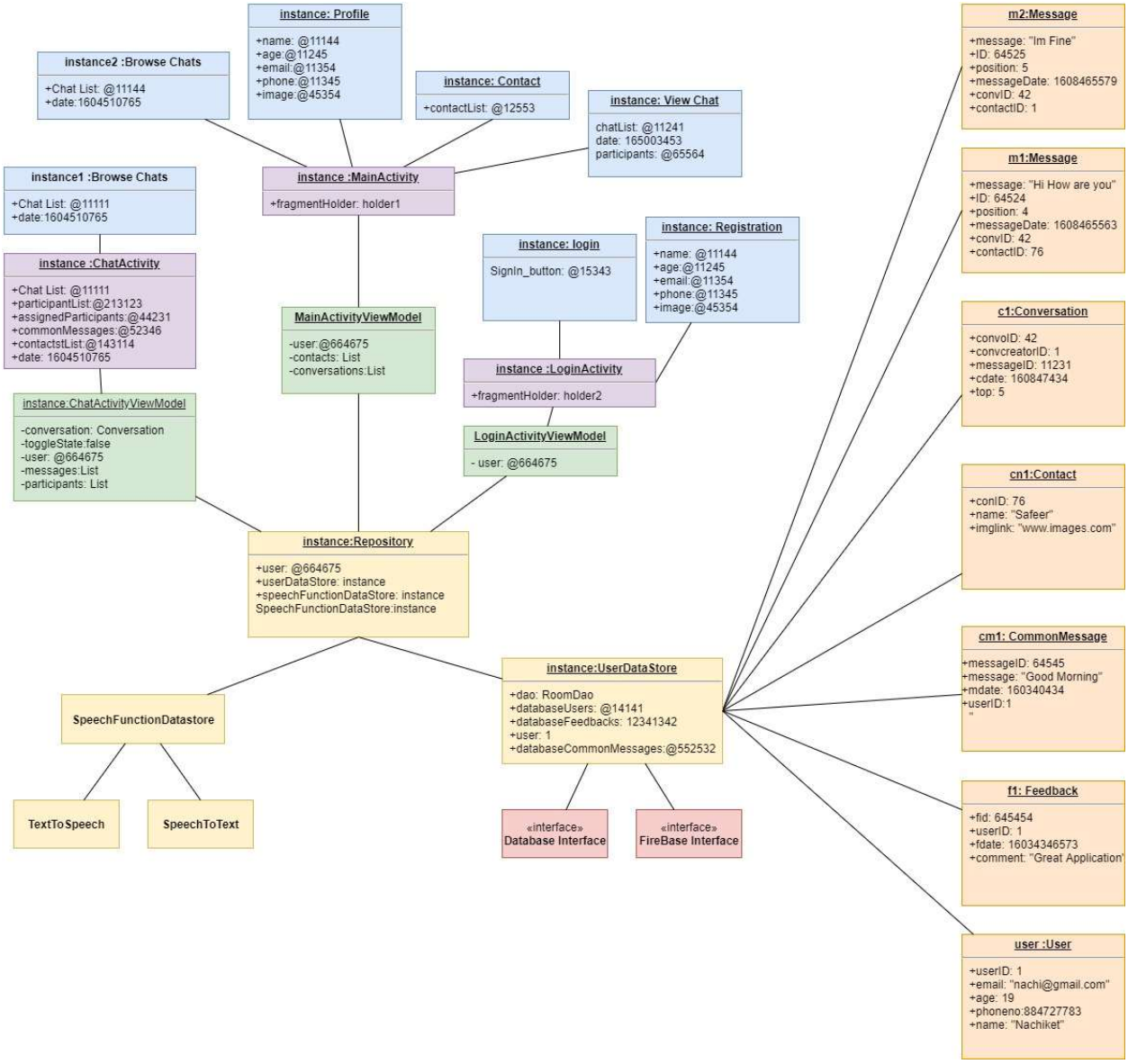
Sequence Diagram



Class Diagrams

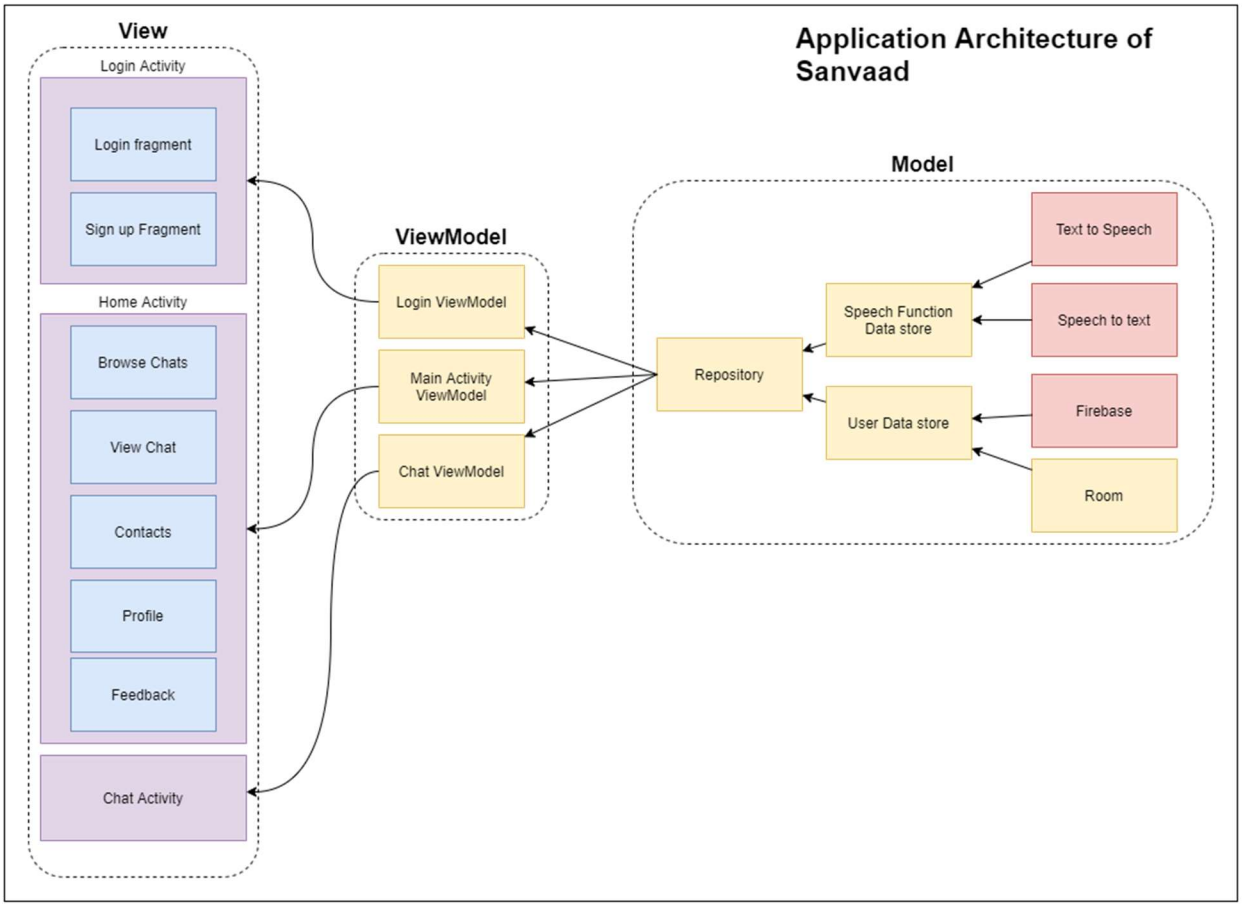


Object Diagram



[Click here to view diagrams in better quality]

Application Architecture



For our application we have followed the Model View ViewModel(MVVM) architecture. The MVVM architecture separates the UI part of the application from the data part, thereby preventing any memory leaks. It makes use of ViewModel to connect the UI with the Data.
In the Above Diagram, we can see how we have implemented the MVVM architecture.

Methodology

As specified in SRS and other previous documents, we had chosen Agile as our SDLC.
Agile software development can be done using two methods:- SCRUM and Kanban. Since we had to meet deadlines for our project, Scrum was an ideal choice.
Scrum done using Sprints. A sprint consists of processes such as discussions,requirement collections, design the solution, implementation of solution, review.
Each sprint lasted for the duration of one week.

Discussion.

The first step in any sprint was a discussion. In this, the agenda mostly included discussing the progress till date, any issues faced in previous sprints, and assignments due this week. This would follow with discussion to any new features to be added to the application. Accordingly the discussion would end when all issues were resolved, and a rough list of features to be implemented in the current sprint were made.

Requirements Analysis

We then use the rough list of features to be worked upon in this sprint, and start brainstorming. These sessions would be detailed and a finalized list of requirements was created. We would also decide upon the learning resources during this phase, where we would discuss if we require to learn any skill we don't already have

Design of solution

During this stage, we finalized for each requirement, how exactly we would proceed. For tasks involving diagrams, we brainstormed proposed designs using diagrams. ER diagrams, class diagrams were created during this phase. A general flow for the implementation phase and specified tasks for each team member for this sprint was decided upon during this phase.

Implementation of solution.

During this phase, the actual implementation was done by the team members. This included writing code and graphic design. This was the longest phase in each sprint, since we had already decided upon what's needed to be done.

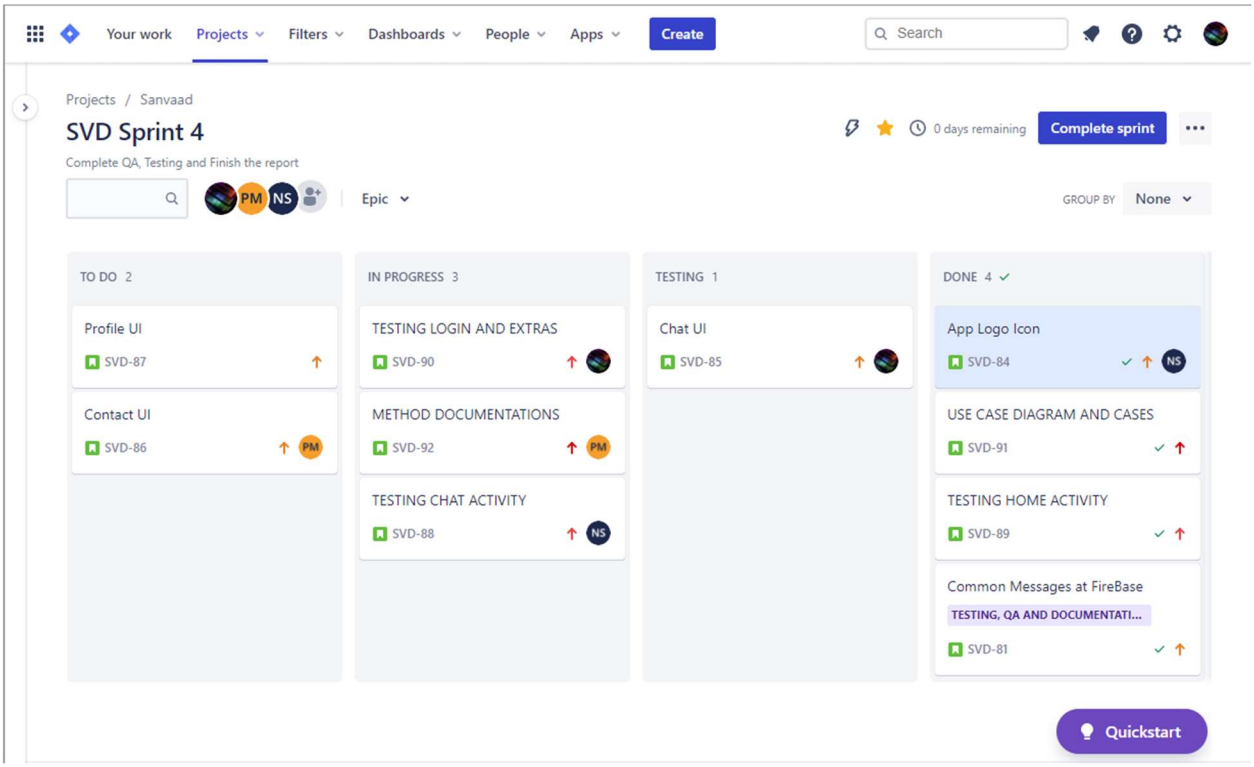
Review.

The sprint would end with a review, this included testing all functionality developed in that week. We would also discuss the backlog and then prepare for the next sprint.

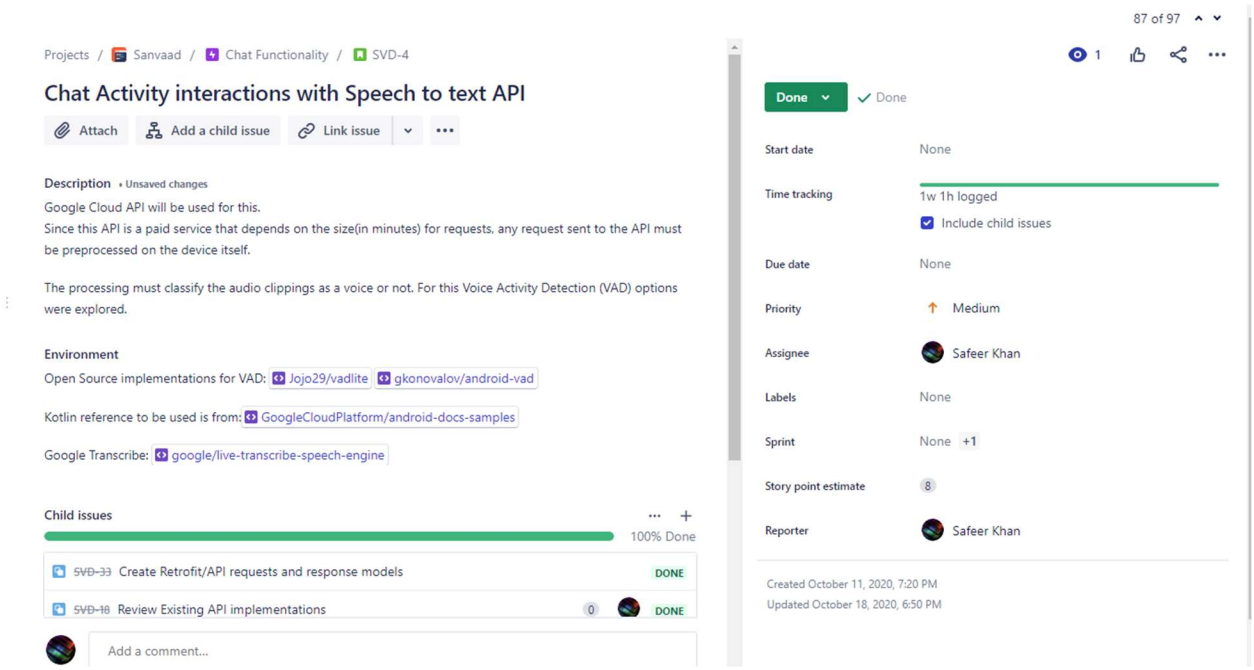
For every sprint, we regularly hosted meetings on Discord at 7 pm. We had three meetings for each sprint. All remaining discussions were done through the Whatsapp chat platform.

Project Management

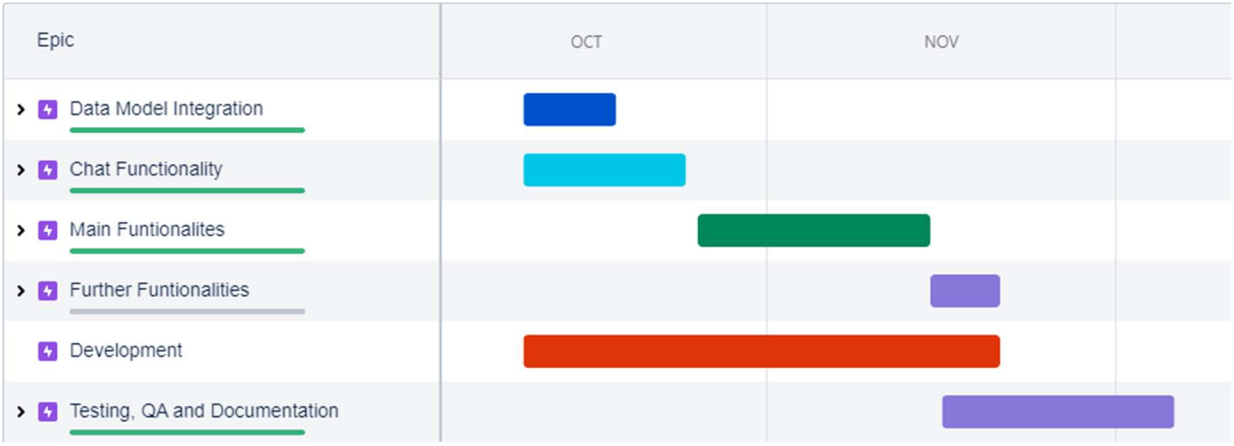
For project management, we made use of an online tool called JIRA. It is a tool that allows managing agile software projects, and allows us to use the Scrum framework.
Scrum is mainly done on a board which consists of small cards, describing the task, called story. Each card can be assigned to a team member, and has all relevant information about the task.
The board is split into 4 sections: Todo, in progress, testing, done. All tasks for the current sprint are initially in Todo, these tasks are added during the design phase. During the implementation phase the team member is working on any task, they shall move it to the ‘in progress’ column on the board. Once the task is done, it is put in the ‘Testing’ column, which is then tested during the review and then considered done when put in the done column.



For each card, the activity, checklist, attachments and reports along with time tracking is done in Jira



Besides Scrum, Jira allowed us to track our progress with the help of Gantt Chart of the project timeline, and to divide stories into modules called epics.
It also links all your documents in one place.

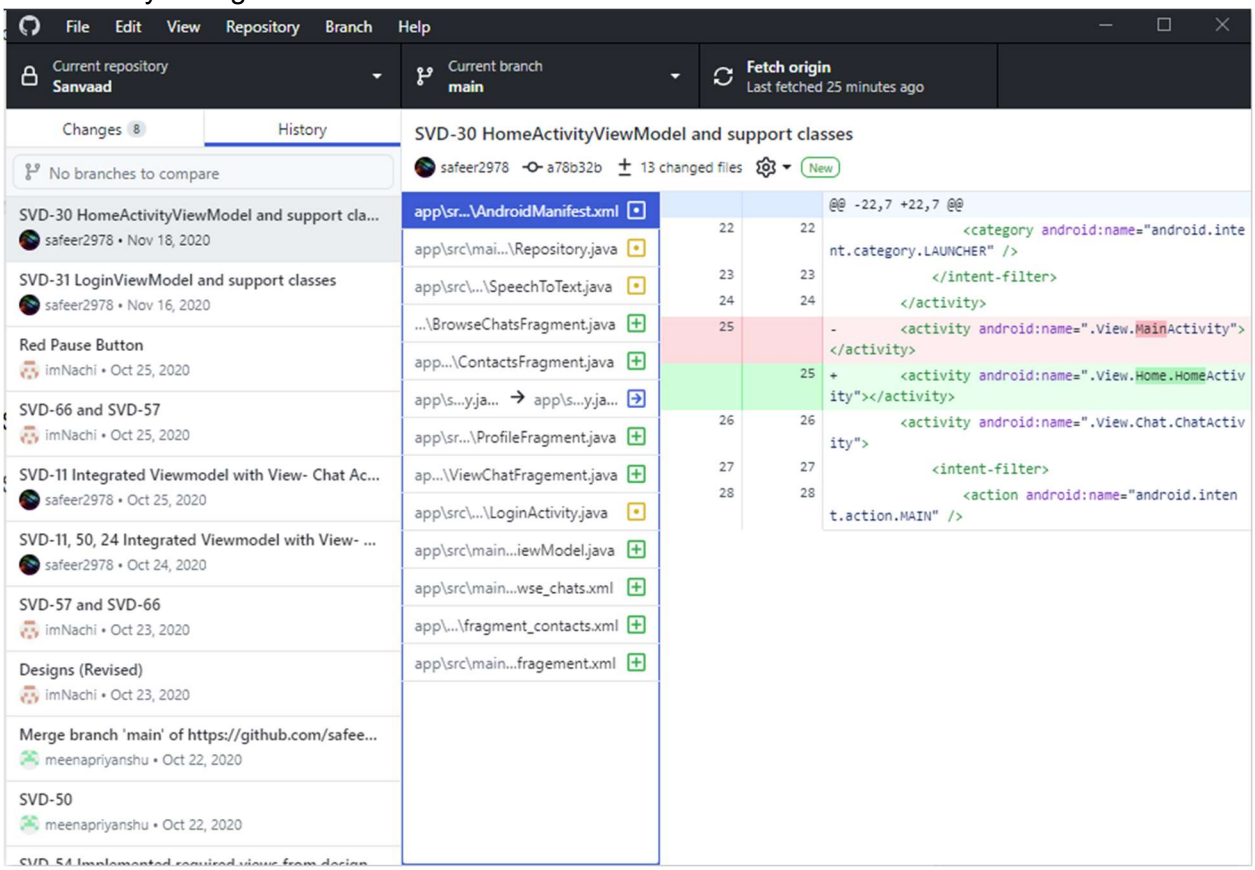


Version control

Entire codebase and relevant resource files were shared among team members using Git version control. It allowed us to track our individual contributions, and resolve conflicting commits. We made extensive use of git during the implementation and review phase of the project, ensuring everyone was on the same page.

For every commit, there was an associated task on the Jira scrum board. Hence, we followed a convention such that every commit would have the ID of the card which is associated with the changes made in the commit.

For e.g if SVD-28 is a task on jira pertaining to Login functionality, the commit consisting of login functionality changes would be labeled SVD-28



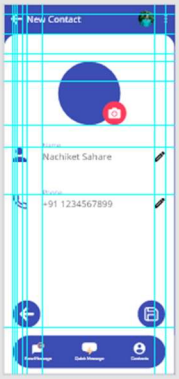
We specifically used Github for sharing a common repository and Github Desktop as GUI application instead of git CLI, mainly because we didn't unnecessarily want to waste time learning the CLI, and since GUI is already available and very intuitive to use.

UI Design

For User Interface Design, we made the use of software apps like Adobe XD, Adobe Photoshop, Adobe Illustrator.
Adobe XD is a vector based user experience design tool for web and android UI development. This Software is used to design and visualize the look and experience of the application. We implemented visual concepts through computer software.
Adobe Photoshop is a visual graphic designing software which was used to create graphical assets like buttons, screens, backgrounds.
Adobe Illustrator is a vector graphics design software which was used to create graphical illustrations and logos for the application.
These graphical illustrations are used to better accessibility and better navigation of the application. Also to enhance the look and feel of the application. We made illustrations which continue the theme of the application and make it visually pleasing.

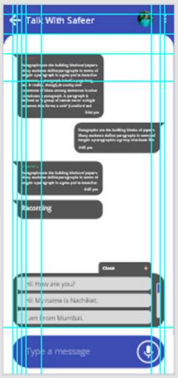
iPhone X, XS, 11 Pro – 1

iPhone X, XS, 11 Pro – 14

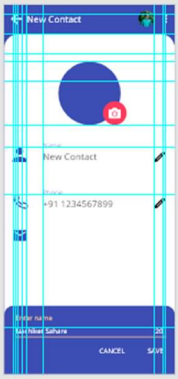


iPhone X, XS, 11 Pro – 15

iPhone X, XS, 11 Pro – 12

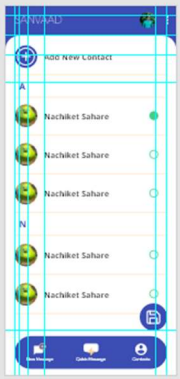


iPhone X, XS, 11 Pro – 16



iPhone X, XS, 11 Pro – 11

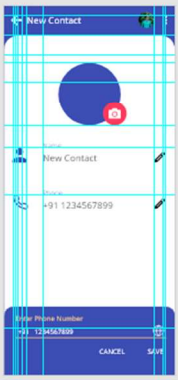
iPhone X, XS, 11 Pro – 5



iPhone X, XS, 11 Pro – 6

iPhone X, XS, 11 Pro – 17

iPhone X, XS, 11 Pro – 18



Collaboration

Effective communication and collaboration between members of the team are one of the most important aspects of smooth completion of any project. During the duration of this project, due to the challenges imposed by the pandemic situation all of the brainstorming and meetings had to take place online rather than physically. This is where platforms such as Whatsapp Web and Discord came into the equation as Whatsapp Web took responsibility for all the textual conversations, exchange of documents and other media concerning the project. While on the other hand the organising of meetings required a platform on which one can share audio, video as well as their screen's to convey their ideas which is where Discord came into action.

Google Docs played an important role as it was the go-to platform for the duration of the project whenever any documentation related task needed to be completed. One main reason for it being helpful is that it allowed us to work simultaneously on the same documents regardless of us being on different computers. The continuous updating of the changes made to the document made it very easy to work hand-in-hand for the members of the team.

Learning Resources

Apart from classroom ppts, specifically for android, we made use of youtube videos, udacity open courses, coding blogs such as Simplified Coding, Swiggy Bytes, etc.

Software & Hardware platforms

Software Platforms

1. **Android Studio**
   All development was done using Android studio IDE. Android studio has an inbuilt build automation system called Gradle and inbuilt support for git, which allowed us to share and maintain our code base across all team members.
2. **Github Desktop**
   Instead of using CLI for interacting with git, we chose to opt for github desktop. It's a GUI application, allowing us to make commits, push/pull to repository, and manage conflicts much easier since it has a smooth learning curve.
3. **Adobe XD**
   Adobe XD is a vector based user experience design tool which was used in the UI - UX designing part of the project. A Wireframe prototype was developed in Adobe XD to get a demo of the app.
4. **Adobe Photoshop**
   Adobe Photoshop is used to create artwork for our Project.
5. **Adobe Illustrator**
   Adobe Illustrator is used to create assets like buttons and logos for our Project.

Hardware Platforms

1. **Android phone**
   For debugging and testing, we made use of our own android phones.

Online Tools

1. **Jira**
   For project management we made use of Jira.
2. **Draw IO**
   All our diagrams were created using draw.io. It allowed us to collaborate on the same diagrams, while we were all on separate PCs.
3. **Docs**
   We made use of Google Docs for all our reports. It was also used for maintaining a single source of information.
4. **Whatsapp Web**
   All text based communication was done through a Whatsapp group.
5. **Discord**
   Discord allowed us effective communication in terms of voice, video and content sharing. Meetings were organised weekly at 7 pm to check on targets completed, deciding tasks for the next week and brainstorming.
6. **GitHub**
   All files were shared between team members using a single Github repository.

7. **Firebase Console**
   Database and user monitoring was done through Firebase console
8. **Google Cloud Platform**
   Since both the APIs we use are provided by GCP, we made use of Google cloud console.

Testing and Validation

Testing

Testing UI

The UI consists of 8 Screens total. Each of these screens consists of buttons, text inputs, text displays, and other ui components.

Test Methodology

Inorder to test the UI, we didn't make use of any automated ui testing tools. Instead, we just run the app and use it as per the test case procedure to check if the result is as expected.

For each of these screens, we have made test cases and formulated our procedure and results in the form of a table.
Each test case is in a tabular format consisting of attributes- test-case id, name, description, procedure, expected result, actual result, result description and screenshots.

For each Test case, if the application failed to meet the requirements, we deem the test as ‘Failed’. Each ‘Failed test is followed with a Fix section. The Fix table consists of attributes- description, commit id, new Screenshots if required..

We have tested the UI for the following test cases:

- Login
- Registration form test
- View Chat with Multiple Participants
- Back Button in Chats.
- Contact Tab
- Retrieve Contacts
- Retrieve Conversations
- Chats Button
- New Conversation
- Exit Chat
- Type Message
- Send Message
- Speech-to-text button
- Text-to-Speech Button
- Auto text-to-speech
- Add Chat Contacts
- Select Common Message
- Admin Common Message

<u>The Detail Tables of all these Test cases are present in APPENDIX 2</u>

Testing Data Model

The Data model is composed of 4 main units. These are encapsulated in 2 Data stores called UserDataStore and SpeechDataStore. These two Data Stores are then Encapsulated into a Singleton unit called Repository.

Thus in order to test the Data model, we will be testing the **User Data Store** and the **Chat Data Store**

Test Methodology.

For each public method in a data class, we call a private method in the constructor, which consists of all functions calling the functions with mock data. All results are then displayed using the **Log** in the Android Studio IDE.

Thus in each test case, we include the test name, the description of the test, the mock data used for the test, and the result with the Screenshots of Log result.

We have tested for the following cases:

- Contacts Data test
- Message Data test
- Common Message Test
- Text-to-Speech Test
- Speech to text Test

The Detail Tables of all these Test cases are present in APPENDIX 2

Validation

App Permissions

This application requires permissions to access the following resources on the device which would enhance the experience of the application. Microphone, Internet, Network state and media player are the required components of the application to run smoothly and efficiently as intended.
The permissions will help in smooth functioning of the APIs which are the main functionality of the application.

Performance

The speech to text result must be displayed to the user within 1500ms after the end of sentence. The text to speech result must be played within 800ms of user action. This performance is based on the interaction between the phone and the API. It could also differ if any particular permission

is not given to the application which could be a key component of the application. The optimal performance of application could be achieved by giving all permissions asked by the application.

Ease of Use

The application UI is simple and easy to interact with so that users can effectively conversate and navigate between chats. The UI of the application is designed in such a way that it could take advantage of all the API in an efficient way and would be easy to use for the user without learning anything new. We have used assets like buttons and tabs to simplify the design and easy navigation and interactions with the apps API. The process of adding, updating and accessing contacts should be similar to existing experience of users with smartphones. Creating new chats as well as viewing those chats are made easier with the UI. Application provides efficient speech to text service to the user.

Privacy

This project has validated all the non functional requirements that were set in the beginning in terms of privacy. The user's chat data is only stored on their device and not on any remote database, thus maintaining their privacy. Also new users have to authenticate themselves by signing in by their google accounts only after which they will be able to access the application and their data providing us with another layer of security. The data of a user who has logged in using his google account won't be accessible to any other user who logs in on the same application. Hence the functionalities regarding privacy are well validated.

API Subscriptions

For the API requirements for this project we have turned to Google for a solution. We purchased a subscription to Google's text-to-speech API and Google's speech-to-text API which served as the base for the functionality of this project.
Google's text-to-speech API converts text into natural-sounding speech using an API powered by Google's AI technologies.
The issue here however is that, we need to create a monetization strategy for the application before we publish this app on the market.
We used Firebase as our remote database. The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in real time. It allows building the app without needing a server and also makes the database accessible to users regardless of the platform. Firebase also provides us with Google OAuth which allows us to authenticate users by their Google accounts.
Thus with the help of the above API subscriptions we were able to realise the functionalities of the application successfully.

Interfaces

The system effectively uses speech-to-text API to convert the incoming voice into texts which are made available to the user in a conversation as to understand the speaker. The text-to-speech API works well in the chat as well where the user can choose to play a particular message as a voice through the speakers of the device so that the speaker can understand what the user wants to say. In this way the chat provides a way to interface with the text-to-speech and speech-to-text APIs for the user.

The application also successfully authenticates users based on their Google accounts which is a feature provided by Firebase database and then securely retrieves the required data from the remote cloud based Firebase database. In this way the system interfaces with the authentication service and the remote database. While, all the private data like the contacts, is stored locally on the user's device to not invade the user's privacy and is retrieved as needed by the application.

Thus the application validates all the required interfaces between all the APIs and the services used to provide the user all the promised functionalities.

Contributions

Individual's contribution

Our team consisted of three members. Individual contributions for each are given below.

Safeer Khan

- Project planning and management, task assignment.
- Design and implementation of all interactions with Speech to text and Text to speech APIs
- Design and implementation of remote database and Authentication serve with Firebase.
- Design and implementation of Viewmodel Classes and UI interactions with the model.
- Testing of Data Model and Login UI
- Fixing Failed tests.

Priyanshu Meena

- Primarily worked on User Datastore. Designing and implementation of data retrieval methods from Firebase using Room.
- Implementation of Local Database model using Room and associated methods.
- Designing and implementation of Firebase Database which included creating and forming relational structure between the entities.
- Implementation of various recycler views in the chat functionality throughout the application.
- Testing of Chat Activity UI.

Nachiket Sahare

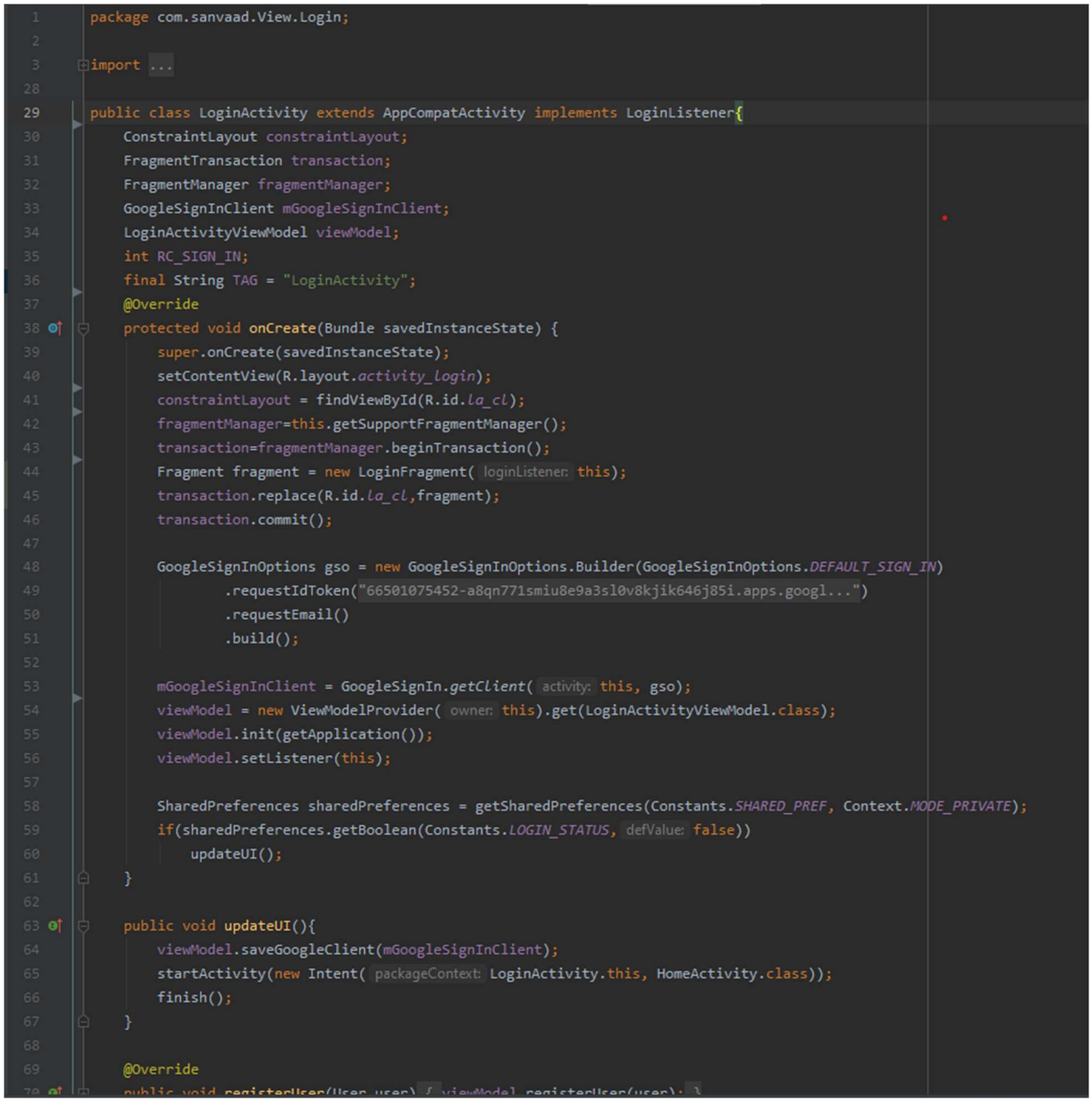
- Designing the entire User Interface and User Experience for the App from scratch.
- Implementing the entire User Interface Design into Front End XML in Android studio.
- Creating all the graphic assets for the project, including screens, icons, backgrounds, and the App Logo.
- Tested the UI in the Main functionality module.
- Fixing UI scalability issues.

Common Contributions

- All diagrams including DFDs, Use case diagrams, Class and Object diagrams, Entity relationship diagrams, Application Architecture, Sequence diagrams were brainstormed by the team and was agreed upon by all team members.
- All documents and reports, including Assignments, SRS, and this report were written together while on discord conference.
- All contributions to the code repository were recorded under git version control.

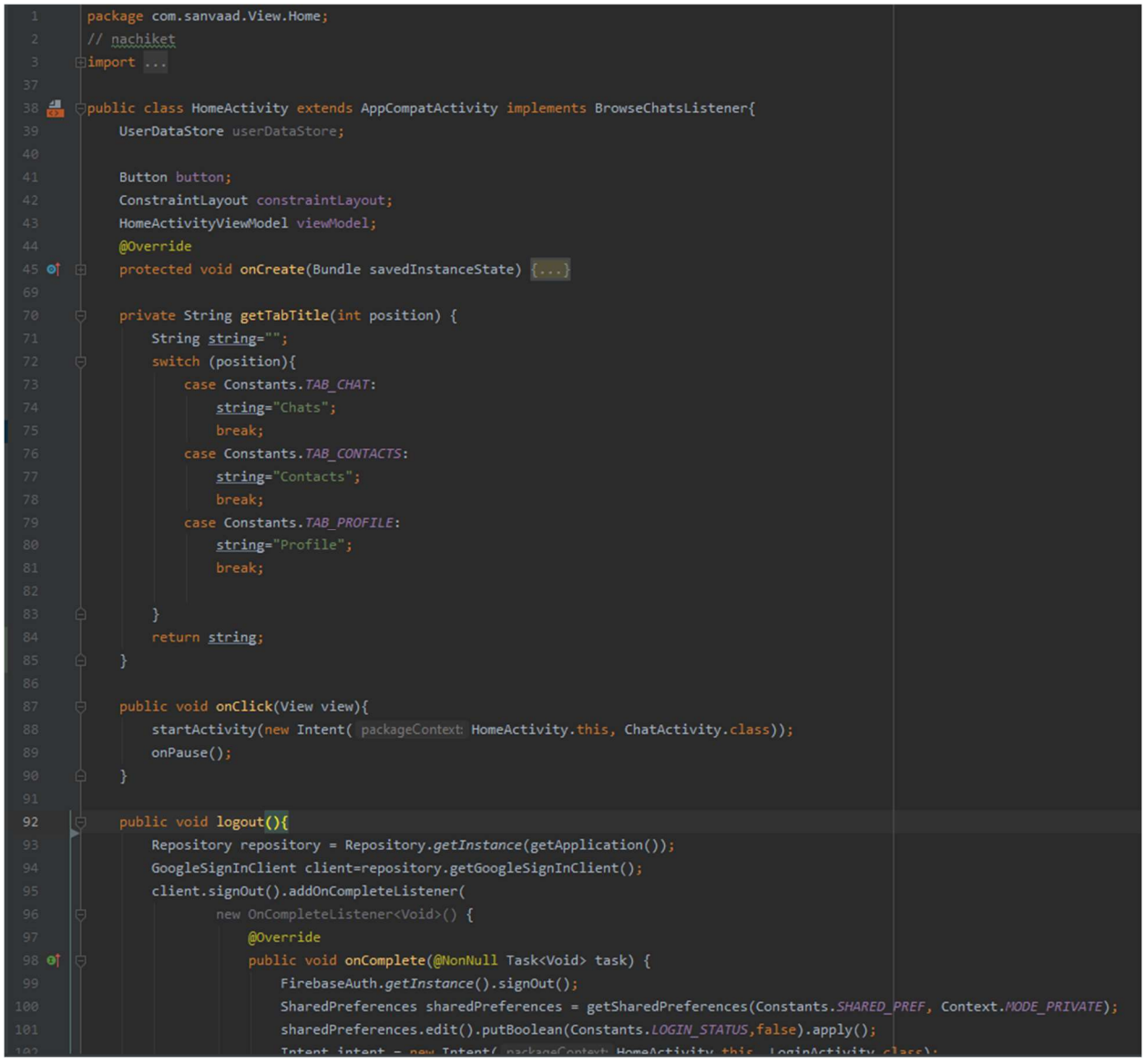
Code snippets

LoginActivity



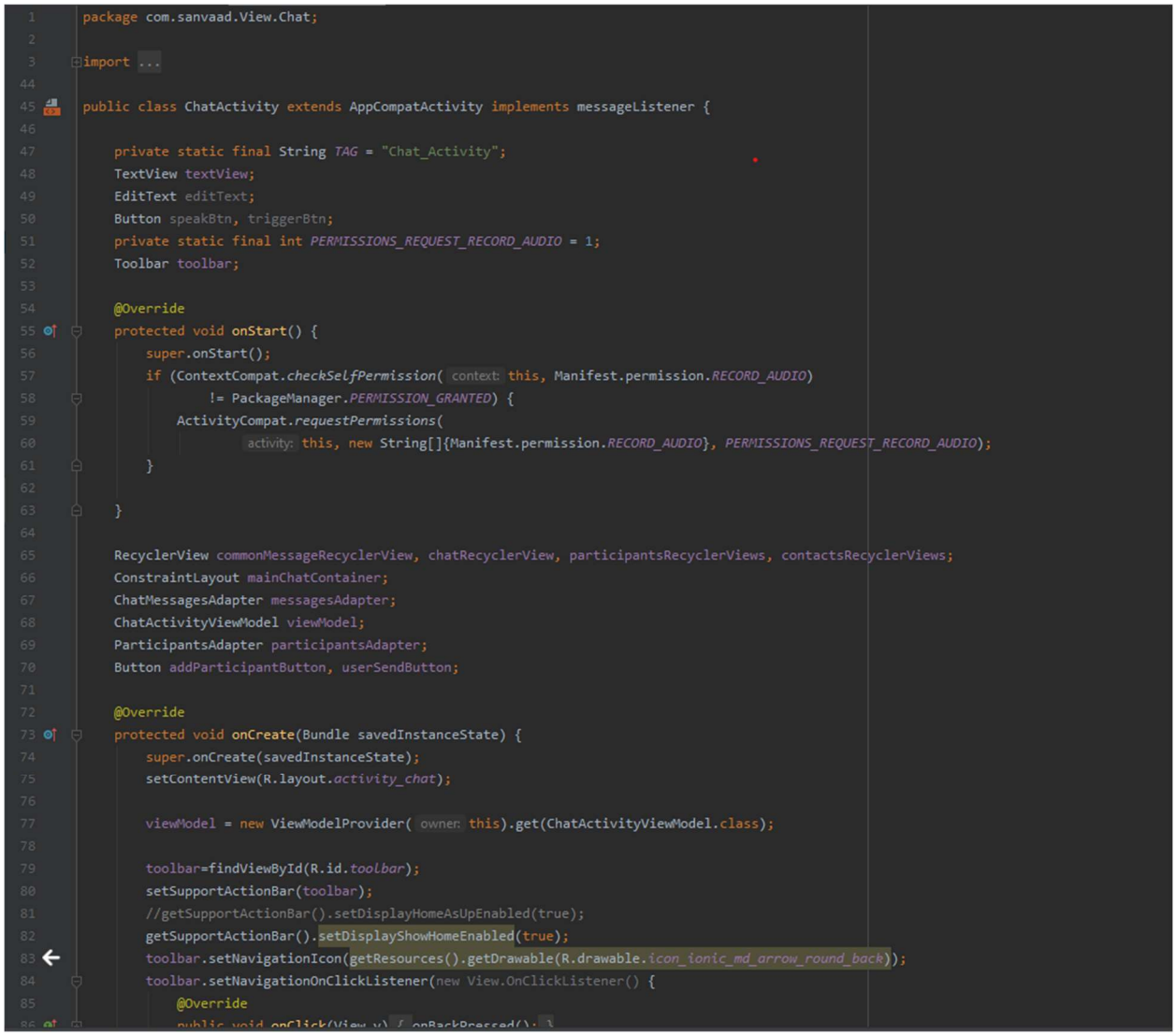
```java
package com.sanvaad.View.Login;

import ...

public class LoginActivity extends AppCompatActivity implements LoginListener{
    ConstraintLayout constraintLayout;
    FragmentTransaction transaction;
    FragmentManager fragmentManager;
    GoogleSignInClient mGoogleSignInClient;
    LoginActivityViewModel viewModel;
    int RC_SIGN_IN;
    final String TAG = "LoginActivity";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        constraintLayout = findViewById(R.id.la_cl);
        fragmentManager=this.getSupportFragmentManager();
        transaction=fragmentManager.beginTransaction();
        Fragment fragment = new LoginFragment( loginListener: this);
        transaction.replace(R.id.la_cl,fragment);
        transaction.commit();

        GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                .requestIdToken("66501075452-a8qn771smiu8e9a3sl0v8kjik646j85i.apps.googl...")
                .requestEmail()
                .build();

        mGoogleSignInClient = GoogleSignIn.getClient( activity: this, gso);
        viewModel = new ViewModelProvider( owner: this).get(LoginActivityViewModel.class);
        viewModel.init(getApplication());
        viewModel.setListener(this);

        SharedPreferences sharedPreferences = getSharedPreferences(Constants.SHARED_PREF, Context.MODE_PRIVATE);
        if(sharedPreferences.getBoolean(Constants.LOGIN_STATUS, defValue: false))
            updateUI();
    }

    public void updateUI(){
        viewModel.saveGoogleClient(mGoogleSignInClient);
        startActivity(new Intent( packageContext: LoginActivity.this, HomeActivity.class));
        finish();
    }

    @Override
    public void registerUser(User user) { viewModel.registerUser(user); }
```

HomeActivity



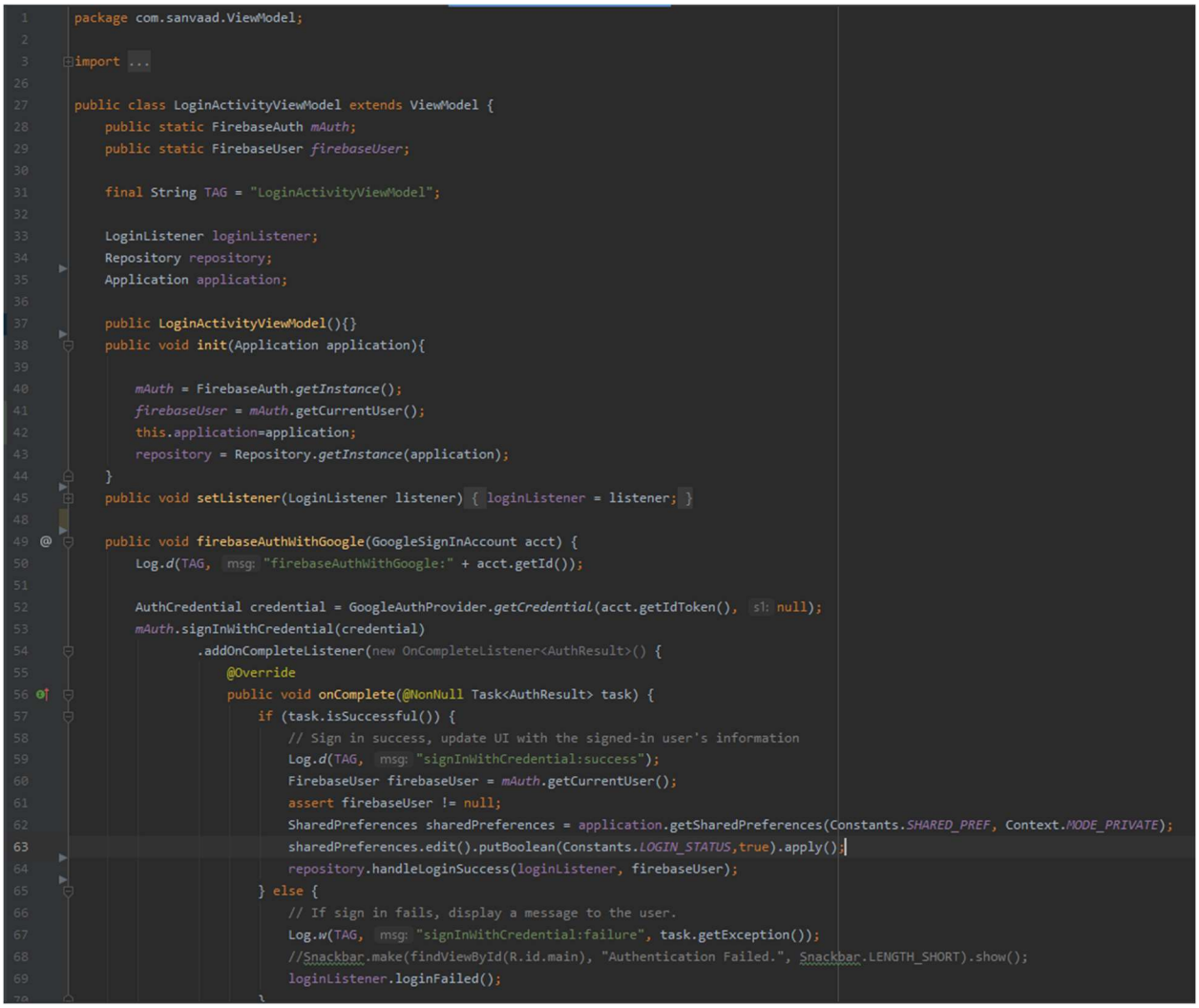
```java
package com.sanvaad.View.Home;
// nachiket
import ...

public class HomeActivity extends AppCompatActivity implements BrowseChatsListener{
    UserDataStore userDataStore;

    Button button;
    ConstraintLayout constraintLayout;
    HomeActivityViewModel viewModel;
    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    private String getTabTitle(int position) {
        String string="";
        switch (position){
            case Constants.TAB_CHAT:
                string="Chats";
                break;
            case Constants.TAB_CONTACTS:
                string="Contacts";
                break;
            case Constants.TAB_PROFILE:
                string="Profile";
                break;

        }
        return string;
    }

    public void onClick(View view){
        startActivity(new Intent( packageContext: HomeActivity.this, ChatActivity.class));
        onPause();
    }

    public void logout(){
        Repository repository = Repository.getInstance(getApplication());
        GoogleSignInClient client=repository.getGoogleSignInClient();
        client.signOut().addOnCompleteListener(
                new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        FirebaseAuth.getInstance().signOut();
                        SharedPreferences sharedPreferences = getSharedPreferences(Constants.SHARED_PREF, Context.MODE_PRIVATE);
                        sharedPreferences.edit().putBoolean(Constants.LOGIN_STATUS,false).apply();
                        Intent intent = new Intent( packageContext: HomeActivity.this, LoginActivity.class);
```

Chat Activity



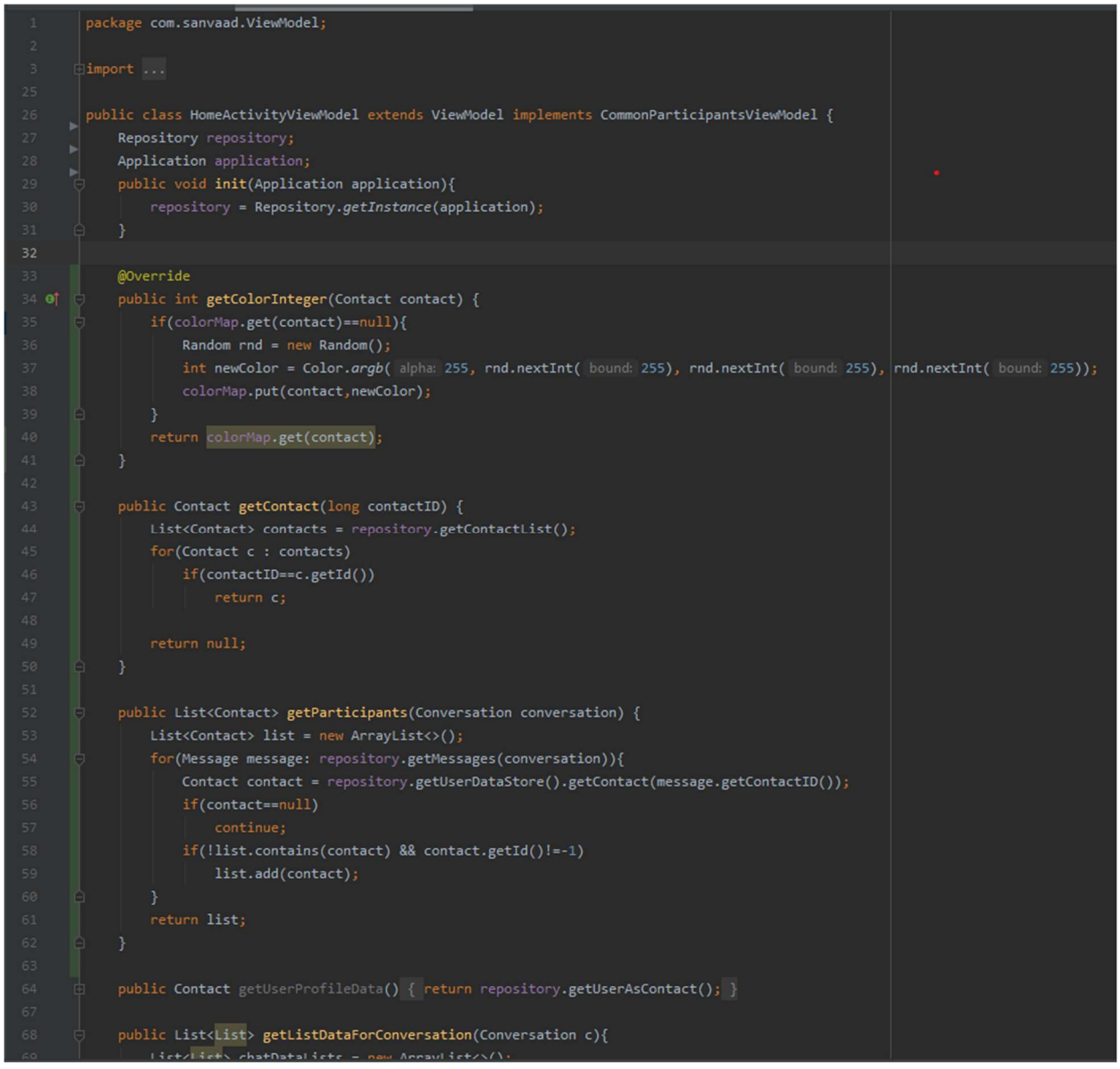
```java
package com.sanvaad.View.Chat;

import ...

public class ChatActivity extends AppCompatActivity implements messageListener {

    private static final String TAG = "Chat_Activity";
    TextView textView;
    EditText editText;
    Button speakBtn, triggerBtn;
    private static final int PERMISSIONS_REQUEST_RECORD_AUDIO = 1;
    Toolbar toolbar;

    @Override
    protected void onStart() {
        super.onStart();
        if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.RECORD_AUDIO)
                != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(
                    activity: this, new String[]{Manifest.permission.RECORD_AUDIO}, PERMISSIONS_REQUEST_RECORD_AUDIO);
        }

    }

    RecyclerView commonMessageRecyclerView, chatRecyclerView, participantsRecyclerViews, contactsRecyclerViews;
    ConstraintLayout mainChatContainer;
    ChatMessagesAdapter messagesAdapter;
    ChatActivityViewModel viewModel;
    ParticipantsAdapter participantsAdapter;
    Button addParticipantButton, userSendButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chat);

        viewModel = new ViewModelProvider( owner: this).get(ChatActivityViewModel.class);

        toolbar=findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        //getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setDisplayShowHomeEnabled(true);
        toolbar.setNavigationIcon(getResources().getDrawable(R.drawable.icon_ionic_md_arrow_round_back));
        toolbar.setNavigationOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { onBackPressed(); }
```

LoginActivityViewModel



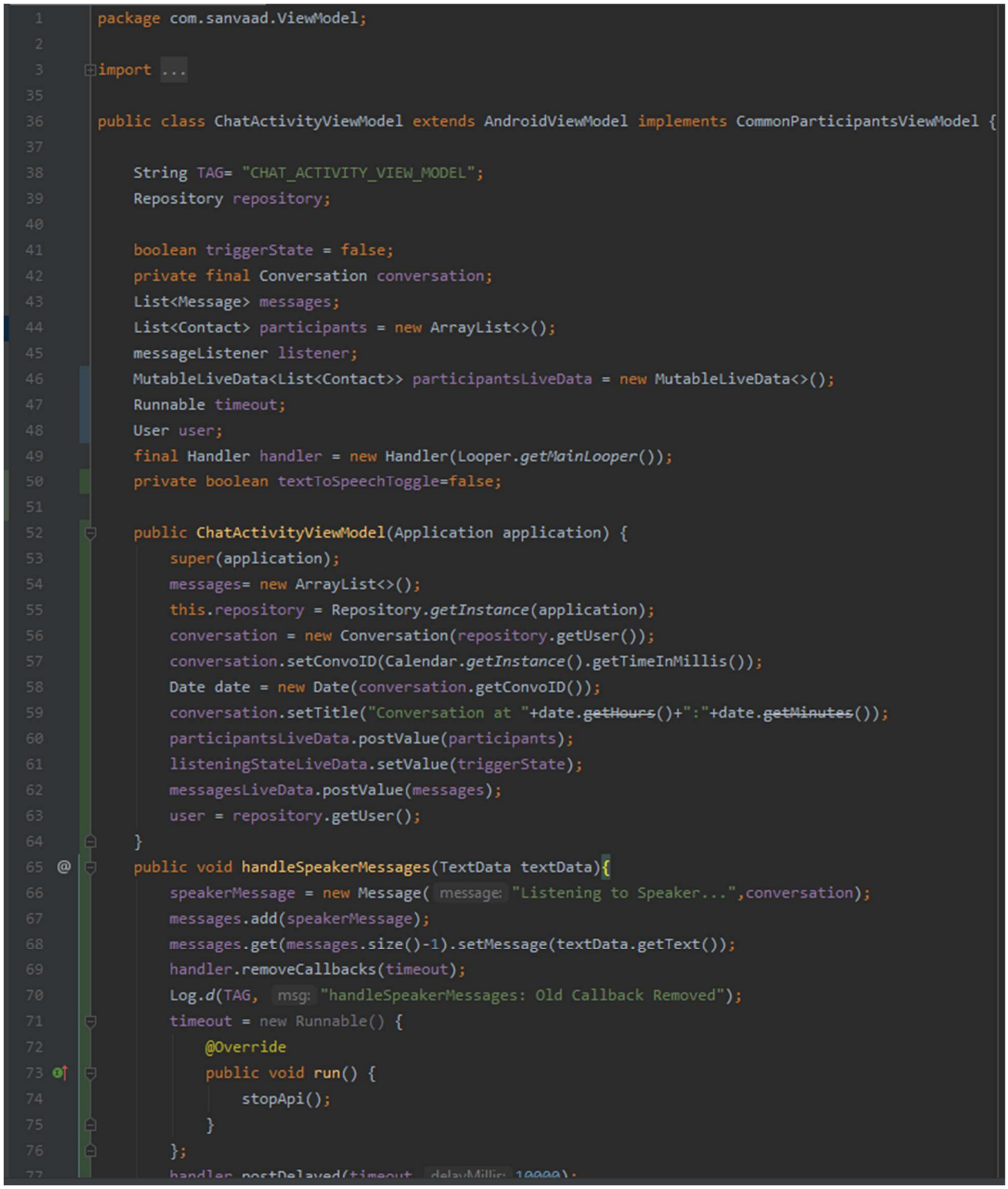
```java
package com.sanvaad.ViewModel;

import ...

public class LoginActivityViewModel extends ViewModel {
    public static FirebaseAuth mAuth;
    public static FirebaseUser firebaseUser;

    final String TAG = "LoginActivityViewModel";

    LoginListener loginListener;
    Repository repository;
    Application application;

    public LoginActivityViewModel(){}
    public void init(Application application){

        mAuth = FirebaseAuth.getInstance();
        firebaseUser = mAuth.getCurrentUser();
        this.application=application;
        repository = Repository.getInstance(application);
    }
    public void setListener(LoginListener listener) { loginListener = listener; }

    public void firebaseAuthWithGoogle(GoogleSignInAccount acct) {
        Log.d(TAG,  msg: "firebaseAuthWithGoogle:" + acct.getId());

        AuthCredential credential = GoogleAuthProvider.getCredential(acct.getIdToken(),  s1: null);
        mAuth.signInWithCredential(credential)
                .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if (task.isSuccessful()) {
                            // Sign in success, update UI with the signed-in user's information
                            Log.d(TAG,  msg: "signInWithCredential:success");
                            FirebaseUser firebaseUser = mAuth.getCurrentUser();
                            assert firebaseUser != null;
                            SharedPreferences sharedPreferences = application.getSharedPreferences(Constants.SHARED_PREF, Context.MODE_PRIVATE);
                            sharedPreferences.edit().putBoolean(Constants.LOGIN_STATUS,true).apply();
                            repository.handleLoginSuccess(loginListener, firebaseUser);
                        } else {
                            // If sign in fails, display a message to the user.
                            Log.w(TAG,  msg: "signInWithCredential:failure", task.getException());
                            //Snackbar.make(findViewById(R.id.main), "Authentication Failed.", Snackbar.LENGTH_SHORT).show();
                            loginListener.loginFailed();
                        }
```

HomeActivityViewModel



```java
package com.sanvaad.ViewModel;

import ...

public class HomeActivityViewModel extends ViewModel implements CommonParticipantsViewModel {
    Repository repository;
    Application application;
    public void init(Application application){
        repository = Repository.getInstance(application);
    }

    @Override
    public int getColorInteger(Contact contact) {
        if(colorMap.get(contact)==null){
            Random rnd = new Random();
            int newColor = Color.argb( alpha: 255, rnd.nextInt( bound: 255), rnd.nextInt( bound: 255), rnd.nextInt( bound: 255));
            colorMap.put(contact,newColor);
        }
        return colorMap.get(contact);
    }

    public Contact getContact(long contactID) {
        List<Contact> contacts = repository.getContactList();
        for(Contact c : contacts)
            if(contactID==c.getId())
                return c;

        return null;
    }

    public List<Contact> getParticipants(Conversation conversation) {
        List<Contact> list = new ArrayList<>();
        for(Message message: repository.getMessages(conversation)){
            Contact contact = repository.getUserDataStore().getContact(message.getContactID());
            if(contact==null)
                continue;
            if(!list.contains(contact) && contact.getId()!=-1)
                list.add(contact);
        }
        return list;
    }

    public Contact getUserProfileData() { return repository.getUserAsContact(); }

    public List<List> getListDataForConversation(Conversation c){
        List<List> chatDataLists = new ArrayList<>();
```

ChatActivityViewModel

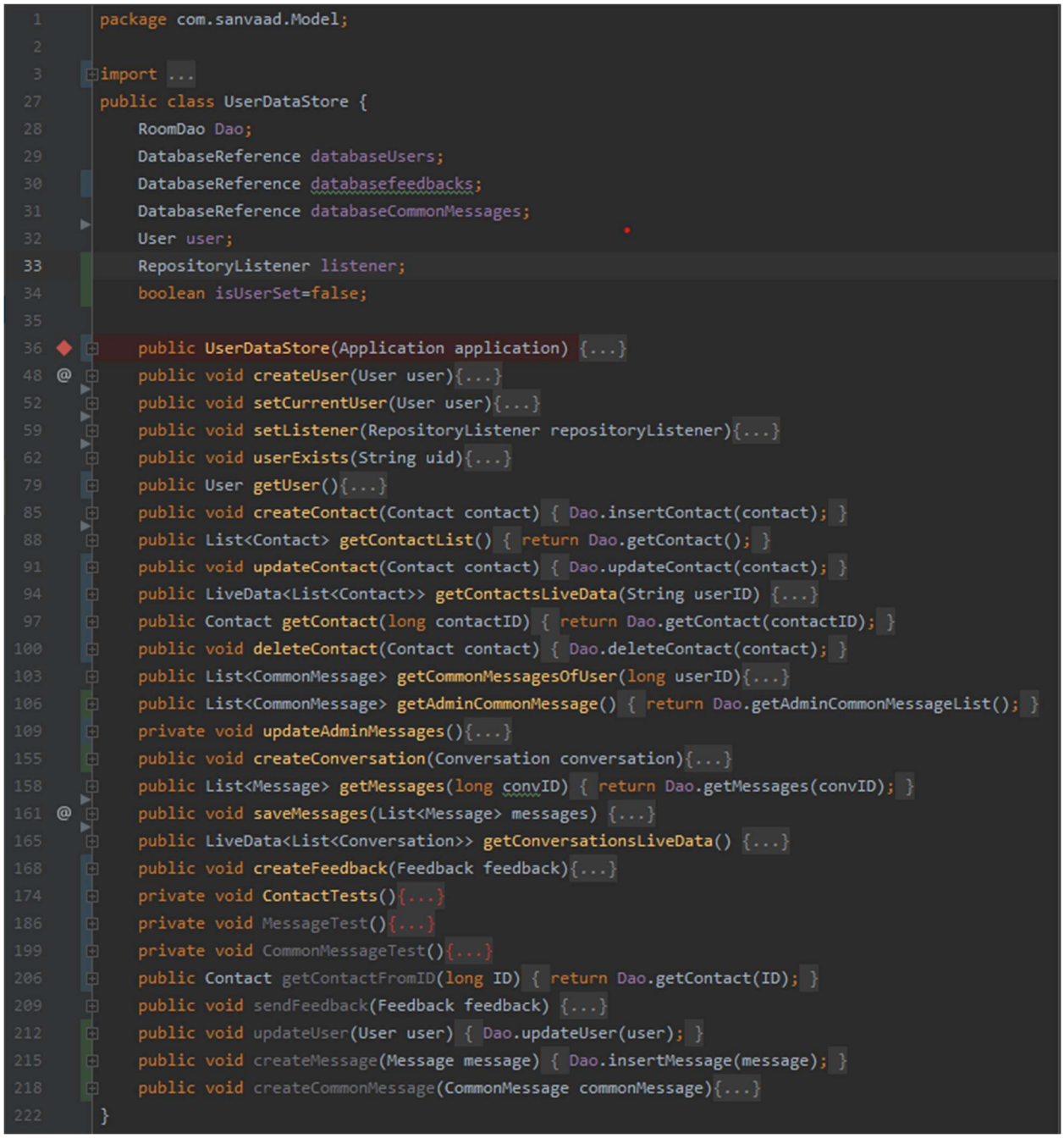


```
package com.sanvaad.ViewModel;

import ...

public class ChatActivityViewModel extends AndroidViewModel implements CommonParticipantsViewModel {

    String TAG= "CHAT_ACTIVITY_VIEW_MODEL";
    Repository repository;

    boolean triggerState = false;
    private final Conversation conversation;
    List<Message> messages;
    List<Contact> participants = new ArrayList<>();
    messageListener listener;
    MutableLiveData<List<Contact>> participantsLiveData = new MutableLiveData<>();
    Runnable timeout;
    User user;
    final Handler handler = new Handler(Looper.getMainLooper());
    private boolean textToSpeechToggle=false;

    public ChatActivityViewModel(Application application) {
        super(application);
        messages= new ArrayList<>();
        this.repository = Repository.getInstance(application);
        conversation = new Conversation(repository.getUser());
        conversation.setConvoID(Calendar.getInstance().getTimeInMillis());
        Date date = new Date(conversation.getConvoID());
        conversation.setTitle("Conversation at "+date.getHours()+":"+date.getMinutes());
        participantsLiveData.postValue(participants);
        listeningStateLiveData.setValue(triggerState);
        messagesLiveData.postValue(messages);
        user = repository.getUser();
    }
    public void handleSpeakerMessages(TextData textData){
        speakerMessage = new Message( message: "Listening to Speaker...",conversation);
        messages.add(speakerMessage);
        messages.get(messages.size()-1).setMessage(textData.getText());
        handler.removeCallbacks(timeout);
        Log.d(TAG,  msg: "handleSpeakerMessages: Old Callback Removed");
        timeout = new Runnable() {
            @Override
            public void run() {
                stopApi();
            }
        };
        handler.postDelayed(timeout,  delayMillis: 10000);
```

Repository

```java
package com.sanvaad.Model;

import ...


public class Repository implements RepositoryListener{

    SpeechFunctionDataStore speechFunctionDataStore;
    UserDataStore userDataStore;
    User user;
    public UserDataStore getUserDataStore(){
        return userDataStore;
    }

    Repository(Application application){
        speechFunctionDataStore = new SpeechFunctionDataStore(application.getApplicationContext());
        userDataStore = new UserDataStore(application);
        userDataStore.setListener(this);
        updateUser(application);
    }
    private static Repository INSTANCE;

    public static Repository getInstance(Application application){
        if(INSTANCE == null){
            INSTANCE = new Repository(application);
        }
        return INSTANCE;
    }
    public LiveData<TextData> getTextData(){
        return speechFunctionDataStore.getTextData();
    }

    public void stopListening(){
        speechFunctionDataStore.onStop();
    }
    public void triggerListening(boolean isListening){
        if (isListening)
            speechFunctionDataStore.onStop();
        else
            speechFunctionDataStore.onStart();
    }
    public void speakText(String s) { speechFunctionDataStore.playTextToSpeech(s); }
    public User getUser(){...}
    public void saveMessages(List<Message> messages) { userDataStore.saveMessages(messages); }
```

User DataStore

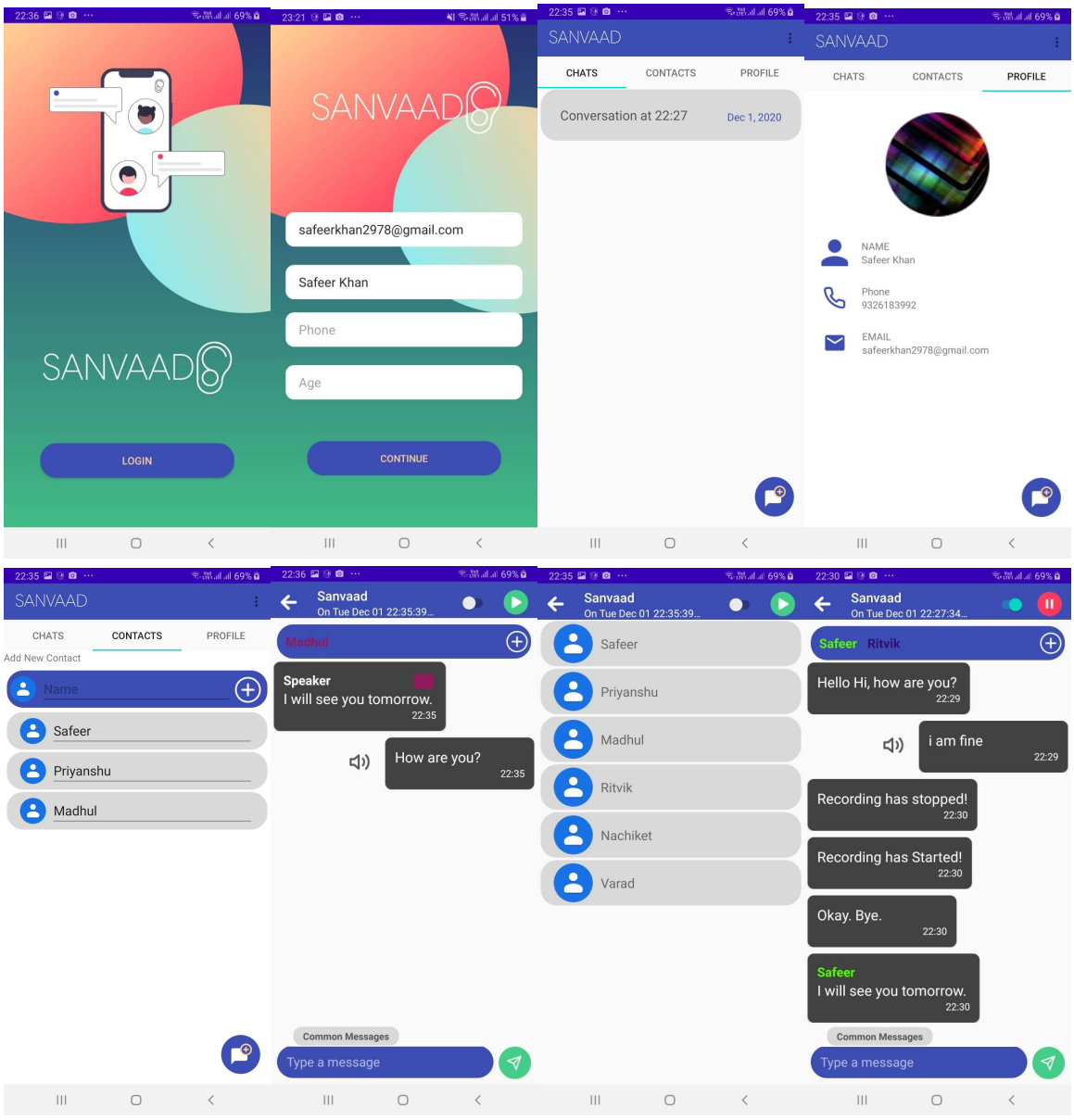


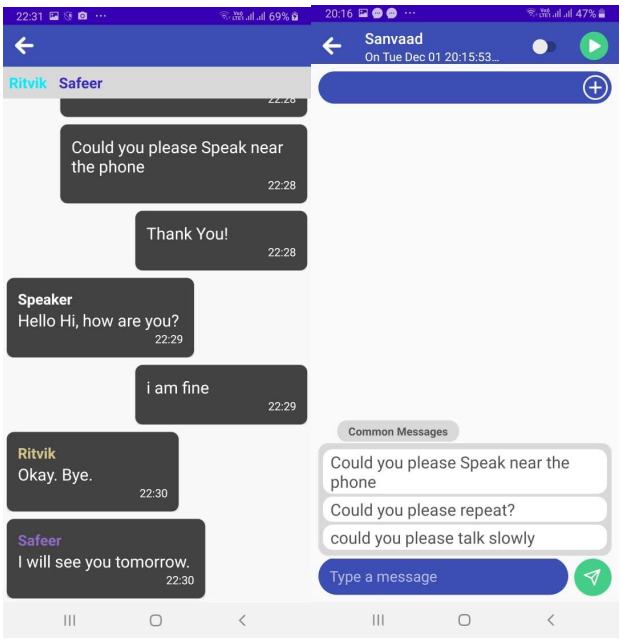
```
package com.sanvaad.Model;

import ...
public class UserDataStore {
    RoomDao Dao;
    DatabaseReference databaseUsers;
    DatabaseReference databasefeedbacks;
    DatabaseReference databaseCommonMessages;
    User user;
    RepositoryListener listener;
    boolean isUserSet=false;

    public UserDataStore(Application application) {...}
    public void createUser(User user){...}
    public void setCurrentUser(User user){...}
    public void setListener(RepositoryListener repositoryListener){...}
    public void userExists(String uid){...}
    public User getUser(){...}
    public void createContact(Contact contact) { Dao.insertContact(contact); }
    public List<Contact> getContactList() { return Dao.getContact(); }
    public void updateContact(Contact contact) { Dao.updateContact(contact); }
    public LiveData<List<Contact>> getContactsLiveData(String userID) {...}
    public Contact getContact(long contactID) { return Dao.getContact(contactID); }
    public void deleteContact(Contact contact) { Dao.deleteContact(contact); }
    public List<CommonMessage> getCommonMessagesOfUser(long userID){...}
    public List<CommonMessage> getAdminCommonMessage() { return Dao.getAdminCommonMessageList(); }
    private void updateAdminMessages(){...}
    public void createConversation(Conversation conversation){...}
    public List<Message> getMessages(long convID) { return Dao.getMessages(convID); }
    public void saveMessages(List<Message> messages) {...}
    public LiveData<List<Conversation>> getConversationsLiveData() {...}
    public void createFeedback(Feedback feedback){...}
    private void ContactTests(){...}
    private void MessageTest(){...}
    private void CommonMessageTest(){...}
    public Contact getContactFromID(long ID) { return Dao.getContact(ID); }
    public void sendFeedback(Feedback feedback) {...}
    public void updateUser(User user) { Dao.updateUser(user); }
    public void createMessage(Message message) { Dao.insertMessage(message); }
    public void createCommonMessage(CommonMessage commonMessage){...}
}
```

Speech data store

```java
package com.sanvaad.Model.Speech;

import android.content.Context;
import androidx.lifecycle.LiveData;
import com.sanvaad.Model.TextData;
import java.io.IOException;

public class SpeechFunctionDataStore {
    final TextToSpeech textToSpeech;
    SpeechToText speechToText;

    public SpeechFunctionDataStore(Context context){
        this.textToSpeech = new TextToSpeech(context);
        speechToText = new SpeechToText(context);
        playTextToSpeech("");
    }
    public void onStart(){
        speechToText.onStart();
    }
    public void onStop(){
        speechToText.onStop();
    }
    public LiveData<TextData> getTextData(){
        return speechToText.getTextData();
    }
    public void playTextToSpeech(String text){
        speechToText.onPause();
        textToSpeech.playText(text);
    }
}
```

Screenshots of output




22:31
69%
Ritvik
Safeer
Could you please Speak near the phone
22:28
Thank You!
22:28
Speaker
Hello Hi, how are you?
22:29
i am fine
22:29
Ritvik
Okay. Bye.
22:30
Safeer
I will see you tomorrow.
22:30
20:16
47%
Sanvaad
On Tue Dec 01 20:15:53...
Common Messages
Could you please Speak near the phone
Could you please repeat?
could you please talk slowly
Type a message

Appendix 1: Use Cases

- Login

Use Case Name:	Login User
Summary:	In order to keep a record of all the conversations a user must login so that the system would be only accessible to that particular user and determine his access level.
Basic Flow:	1. The Use case starts when an existing user indicates that he wants to login. 2. The System requests to login with google. 3. The User enters his username and password. 4. The system verifies the username and password against all registered users. 5. The system starts a login session and displays the Main landing page of the system.
Alternative Flow:	1. If the username is invalid, the use case goes back to step2 of Basic Flow. 2. If the password is invalid the system requests that the user re-enter the password.When the user enters another password the use case continues with Step4 using the original username and new password.
Extension Points:	Authenticate
Preconditions:	The User is registered.
Postconditions:	The user can now obtain data and perform functions according to his registered access level.
Business Rules:	Some data and functions are restricted to certain types of users or users with a particular access level.

- Register

Use Case Name:	Register User
Summary:	In order to keep a record of all the conversations a new user must register so that the system would be only accessible to that particular user and determine his access level.
Basic Flow:	1. The Use case starts when an existing user indicates that he wants to register. 2. The System requests the username and password. 3. The User enters his username and password. 4. The system checks that the username does not duplicate any existing registered usernames. 5. The system requests a name(*), Phone Number(*) or email address(*), password(*). Items marked by (*) are required. 6. The User enters all the required information. 7. The system starts a login session and displays the Main landing page of the system.
Alternative Flow:	1. If the username duplicate an existing username the system displays a message and the use case goes back to step 2 of Basic Flow. 2. If the user does not enter a required field, a message is displayed and the use case repeats step 4.
Extension Points:	none
Preconditions:	none
Postconditions:	The user can now obtain data and perform functions according to his registered access level.
Business Rules:	Some data and functions are restricted to certain types of users or users with a particular access level.

- Authenticate

Use Case Name:	Authenticate
Actor	Firebase
Summary:	Users are authenticated using Google OAuth provided by Firebase. Authenticate manages user sessions through the inbuilt FirebaseAuth instances.
Basic Flow:	1. Provide user with sign-in client when triggered 2. Perform the Oauth session process 3. check for user existence in database 4. return a true signal
Alternative Flow:	1. Check for user signed-in status. 2. If a user signed-in, consider the session valid and on going.
Extension Points:	NA
Preconditions:	User is not authenticated
Postconditions:	User is authenticated

- BrowseChats and View Chats

Use Case Name:	BrowseChats and View Chats
Actor	User
Summary:	To access your previous conversations you could view your chats tab..
Basic Flow:	1. The Use case starts when an existing user wants to access their previous conversations. 2. It navigates through the Chats Tab and selects the desired conversation. 3. Click on your desired conversation. And it will display your chat.
Alternative Flow:	1. If there are no previous conversations , you can start a conversation and it will be displayed in the chats tab.
Extension Points:	none
Preconditions:	You should have a previous conversation
Postconditions:	none
Business Rules:	This data is required to maintain the functionality of the system.

- Add Contacts

Use Case Name:	Add contact
Actor	User
Summary:	The name and contact details are added in your device.
Basic Flow:	1. The Use case starts when a registered user wants to add a new contact and go to the contact tab 2. And add the contact details like name, Phone number etc.
Alternative Flow:	None
Extension Points:	None
Preconditions:	None
Postconditions:	None
Business Rules:	This data is required to maintain the functionality of the system.

- View Profiles

Use Case Name:	View Profile
Actor	User
Summary:	The name and contact details are added in your device.
Basic Flow:	1. The Use case starts when a registered user wants to view his own profile. 2. The User goes to the Profile tab of home screen and it displays all the profile details like Name, Phone No. and email id from which you have registered.
Alternative Flow:	None
Extension Points:	None
Preconditions:	The User must be registered.
Postconditions:	None
Business Rules:	This data is required to maintain the functionality of the system.

- Send Feedback

Use Case Name:	Feedback
Actor	User
Summary:	In order to maintain the smooth functionality of the system, we obtain feedback from the end user to know about any system bugs or glitches.
Basic Flow:	1. The Use case starts when an existing user indicates that he wants to send feedback. 2. The System takes the user to the feedback form. 3. The User enters all the fields of the feedback form and submits the feedback form. 4. The system receives the feedback from the user and sends it to the Technical Team.
Alternative Flow:	1. If the required fields are not submitted a message is displayed to submit the required details.
Extension Points:	none
Preconditions:	The User is registered.

Postconditions:	none
Business Rules:	This data is required to maintain the functionality of the system.

- Retrieve User Data

Use Case Name:	Retrieve Common Message Data
Actor	Firebase
Summary:	Common messages are recorded and stored in Remote Datastore. When these are updated, the changes must reflect in the mobile client as well.
Basic Flow:	1. Authenticate the user 2. Check for network availability 3. Retrieve Messages 4. update local database.
Alternative Flow:	NA
Extension Points:	NA
Preconditions:	Common messages on devices are out of date
Postconditions:	Common messages are consistent with remote database
Business Rules:	Common messages.

- Record Feedback

Use Case Name:	Feedback
Summary:	In order to maintain the smooth functionality of the system, we obtain feedback from the end user to know about any system bugs or glitches.
Basic Flow:	1. The Use case starts when an existing user indicates that he wants to send feedback. 2. The System takes the user to the feedback form. 3. The User enters all the fields of the feedback form and submits the feedback form. 4. The system receives the feedback from the user and sends it to the Technical Team.
Alternative Flow:	1. If the required fields are not submitted a message is displayed to submit the required details.

Extension Points:	none
Preconditions:	The User is registered.
Postconditions:	none
Business Rules:	This data is required to maintain the functionality of the system.

- Start new chat with message or common message.

Use Case Name:	New Chat, Enter Message or Select Common Message
Summary:	This allows the user to create or start a new chat. User can type messages. Alternatively, they can also use common messages.
Basic Flow:	1. The Use case starts when the user clicks on the start new chat button. 2. The system then creates a new conversation between the user and a contact/stranger. 3. This conversation contains the messages or interactions between the user and the second person. 4. User clicks on the type message button. 5. Then after the keyboard opens up the user can enter a message.
Alternative Flow:	1. The Use case starts when the user clicks on the start new chat button. 2. The system then creates a new conversation between the user and a contact/stranger. 3. This conversation contains the messages or interactions between the user and the second person. 4. The user first clicks on the common messages tab. 5. From the pop-up menu of common messages the user selects a message which is then sent to the chat.
Extension Points:	None
Preconditions:	Authenticate, Retrieve common messages
Postconditions:	None
Business Rules:	This helps the user to start a new chat conversation which is one of the main functionality of the application.

- Retrieve common message Data

Use Case Name:	Retrieve User Data
Actor	Firebase
Summary:	User Preferences stored at Remote database are retrieved
Basic Flow:	1. Authenticate the user and check for network availability 2. Retrieve user data, if exists 3. update local database
Alternative Flow:	NA
Extension Points:	NA
Preconditions:	User data is out of date
Postconditions:	User data in in sync with backend
Business Rules:	User data sync with backend

- Play Message

Use Case Name:	Play Message
Summary:	Allows the user to enter text and play the message using Text-To-Speech
Basic Flow:	1. The Use case starts when a new conversation is created 2. User first types their message in a textfield. 3. The user then clicks the play button. 4. The system then calls the text-to-speech, and the message is played
Alternative Flow:	NONE
Extension Points:	none
Preconditions:	Create conversation, text-to-speech
Postconditions:	none
Business Rules:	This use case completes the other half of the chat function

- Convert Text to Speech

Use Case Name:	Convert Text To Speech
Actor	Text to Speech API
Summary:	The api will return an audio clip upon a text request
Basic Flow:	1. Receive a text input 2. Return an audio clip
Alternative Flow:	NA
Extension Points:	NA
Preconditions:	User has written their message
Postconditions:	Message in audio format is played
Business Rules:	Mute User can communicate with contact

- Assign Contact to Message

Use Case Name:	Assign Contact
Summary:	Allows the user to add a pre-saved contact to a chat conversation.
Basic Flow:	1. The Use case starts when a new conversation is created. 2. User clicks on the add contact button 3. From the list of contacts available, the user selects a contact to add to the chat.
Alternative Flow:	None
Extension Points:	The user can be provided a feature to add a contact in the assign contact feature
Preconditions:	Create conversation, add contact
Postconditions:	none
Business Rules:	This use case helps complete the chat functionality as it makes it possible for the user to assign a contact to the conversation or chat.

- Speak

Use Case Name:	Speak Message

Summary:	The speaker's voice is recorded and sent to the api for Speech-to-text processing. The result is then displayed to the user.
Basic Flow:	1.The Use case starts when new Conversation is created. 2. Speaker's voice is detected using the Voice Activity Detection library. 3. The audio consisting of the speaker's voice is then sent to API. 4. The result of the API is displayed to the user.
Alternative Flow:	None.
Extension Points:	None
Preconditions:	Create conversation, convert speech-to-text
Postconditions:	None
Business Rules:	This use case completes one half of the chat function

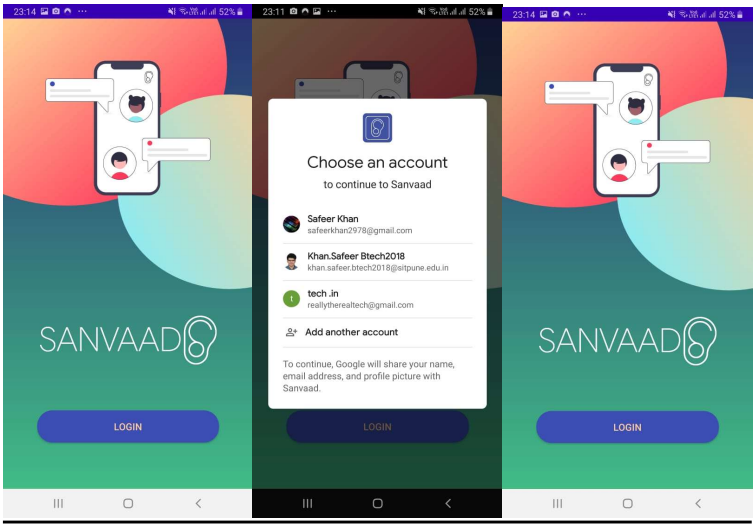
- Convert Speech to Text

Use Case Name:	Convert Speech to Text
Actor	Speech to Text API
Summary:	The api will will receive audio stream, to which it will transcribe and return the result
Basic Flow:	1. Initialize audio Stream 2. Return a transcription
Alternative Flow:	NA
Extension Points:	NA
Preconditions:	Contact is Speaking
Postconditions:	Contact message is delivered to user
Business Rules:	Transcribing Contact's Message

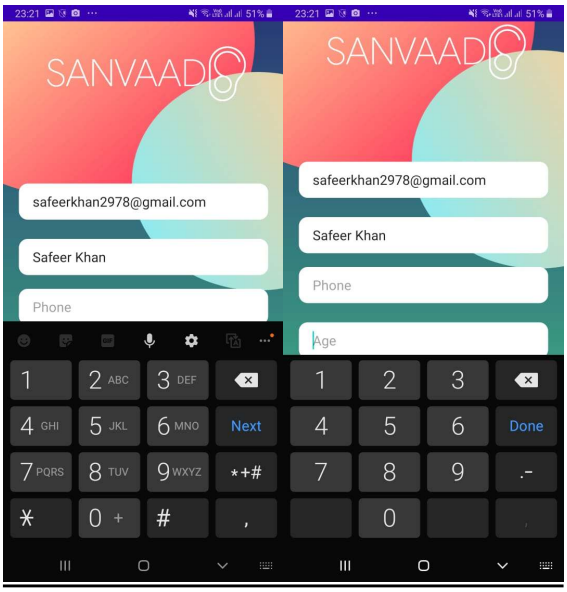
Appendix 2: Test Cases:

UI Test Cases:

- Login

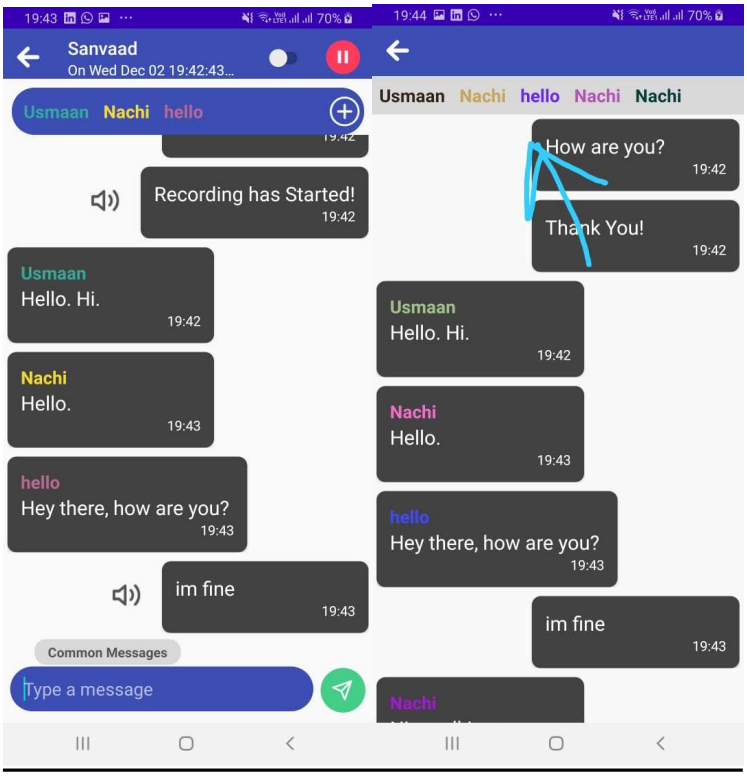
Test Case ID	**1**
Name	Login
Module	UI-Login
Description	This test case is to check if login function is working correct.
Test Procedure	1. Click on the Login button on UI 2. This should trigger the google client. 3. Sign in to google client
Expected Result	Login through google will lead to register or home screen.
Actual Result	**Passed**
Actual Result Description	Login is triggering to Registration Page, and to home screen directly if user is already registered.
Screenshots	

- Registration form test

Test Case ID	**2**
Name	Registration Form Test
Module	UI-Login
Description	The registration form consists a form with several fields. For each field there are certain input such as age and phone number has constraints.
Test Procedure	1. login 2. Check if known data is auto filled 3. check if Input constraints are matched
Expected Result	All fields with number inputs must show number keypad.
Actual Result	**Passed**
Actual Result Description	Result as expected
Screenshots	

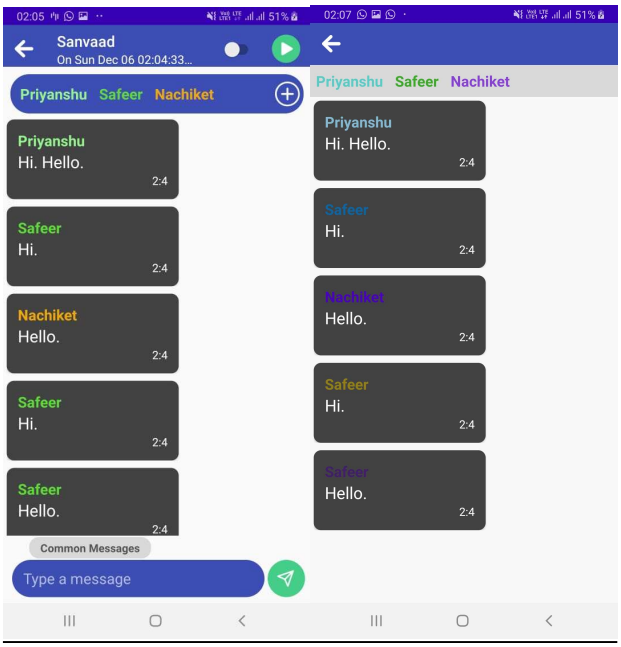
- View Chat with Multiple Participants

Test Case ID	**3**
Name	View Chat with Multiple Participants
Module	UI-Main Functionality
Description	This Functionality is added to add multiple participants to the

	conversation. Or add a pre-existing participant from the contact list.
Test Procedure	1. Start a conversation. 2. Press the + button to add participants in the conversation. 3. Select from your contact list or add. 4. The selected contact should be reflected in the participant list.
Expected Result	The Expected result was to add any existing contacts or add a new contact to the Participant list.
Actual Result	**Failed**
Actual Result Description	If the same contact is selected twice it is being added twice with the same name.
Screenshots	

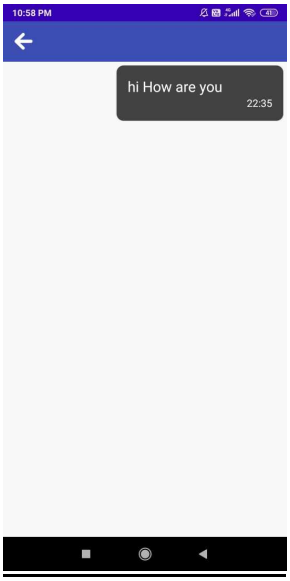
- FIX-View Chat with Multiple Participants

Test Case ID	**Fix-1**
Name	**FIX**-View Chat with Multiple Participants
Commit ID	**8bcb2372 safeer2978**

Description	**Minor issue- id not specified**
Result	**Fixed**
Screenshots	

- Back Button in Chats.

Test Case ID	**4**
Name	Back Button in Chats.
Module	UI-Main Functionality
Description	The Back button should take the user to the home screen.
Test Procedure	1. Login and select any previous chats. 2. Click the back button to go to the home page
Expected Result	The expected result is to go back to the Home page
Actual Result	**Failed**
Actual Result Description	The page is stuck at the conversation screen, there is no possible way to go back to home screen rather than force quitting.

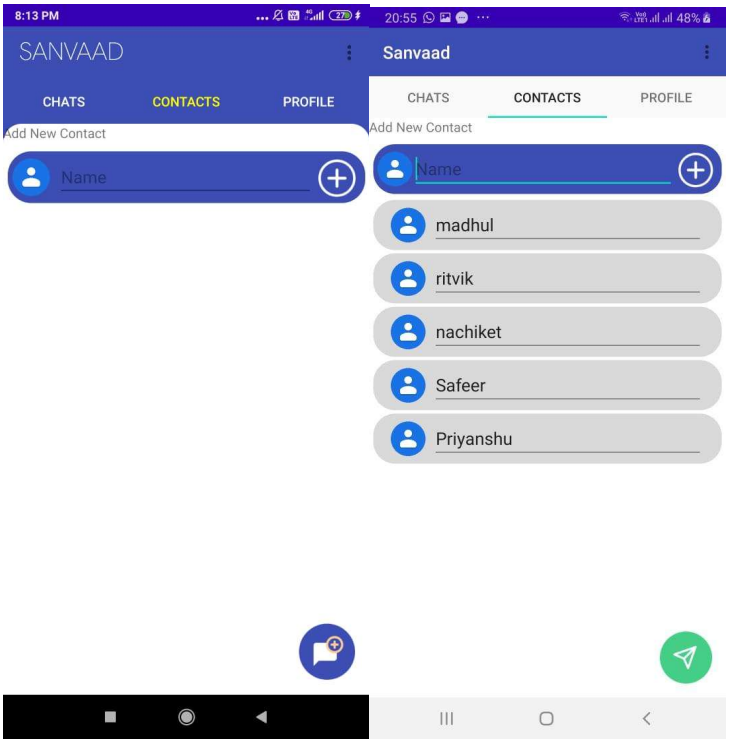
Screenshots	

- Fix: Back Button

Test Case ID	**Fix-2**
Name	**FIX-**Back Button
Commit ID	**e2f083e9 safeer2978**
Description	**On back function not attached to the view Component**
Result	**Fixed**
ScreenShots	**Not Required**

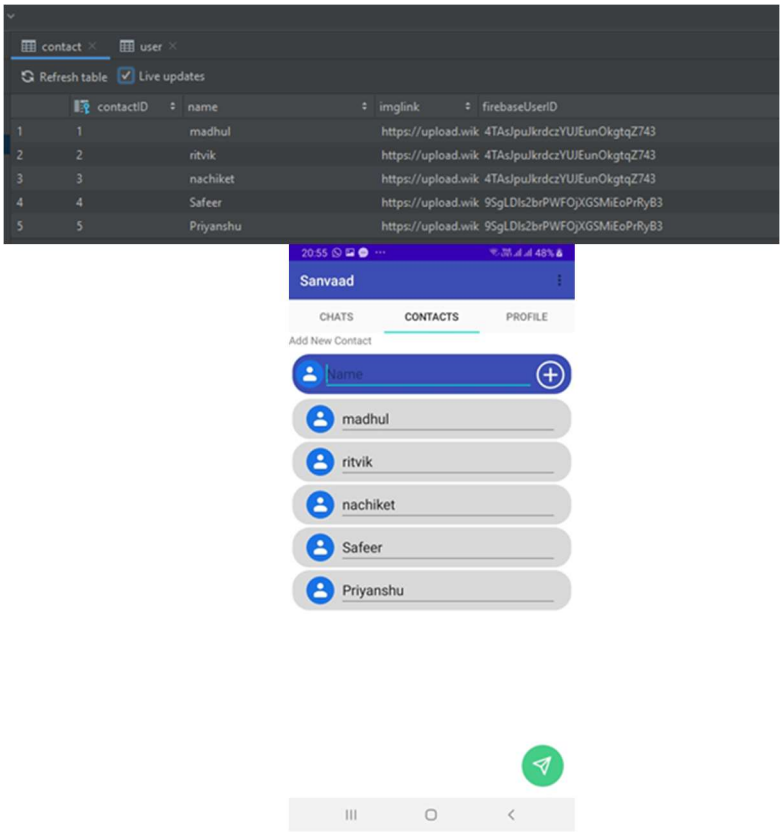
- Contact Tab

Test Case ID	**5**
Name	Contact Tab
Module	UI-Main Functionality
Description	List of All contacts belonging to the user alone must be retrieved. Contacts of other users must not be accessible.
Test Procedure	1. Login and click on the Contact tab and access your contact tab.

Expected Result	Expected result is to display all the accessible contact details which are stored on your phone or your account.
Actual Result	**Passed**
Actual Result Description	It displays all the contact details.
Screenshots	

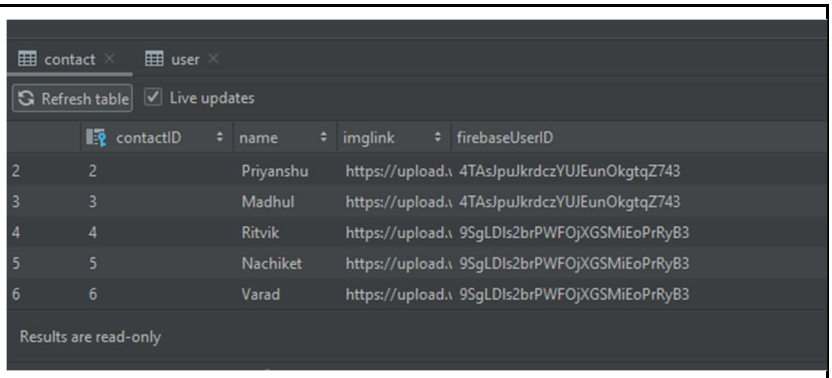
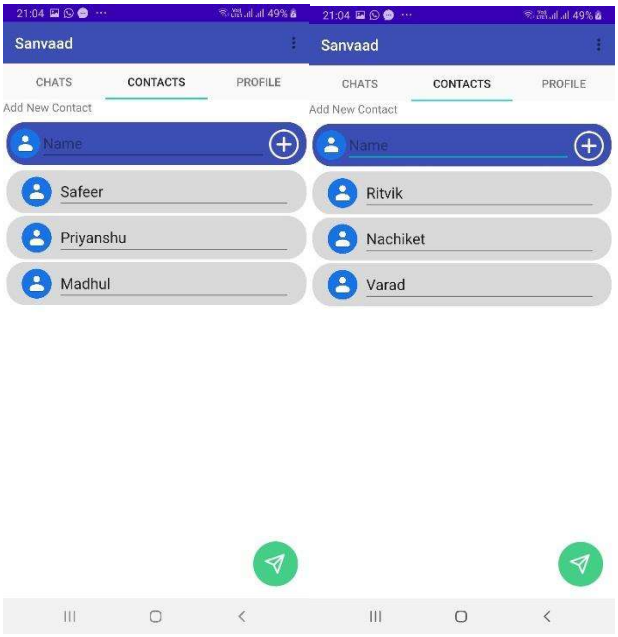
- Retrieve Contacts

Test Case ID	**6**
Name	Retrieve Contacts
Module	UI-Main Functionalities
Description	List of All contacts belonging to the user alone must be retrieved. Contacts of other users must not be accessible
Test Procedure	1. Login and go to the contacts tab. 2. add one or two contacts. 3. Logout and Sign in Again with a different account 4. Again switch to the contacts tab

Expected Result	Contacts added by the previous user are not visible to the new user.
Actual Result	**Failed**
Actual Result Description	Notice below that contacts belonging to different users are stored in a local database, however, the UI presents all contacts to the user. This is not desired and hence needs Fixing
Screenshots	

- Fix

Test Case ID	**Fix-3**
Name	**FIX-**View Chat with Multiple Participants
Commit ID	**63c9e0b0 safeer2978**
Description	**Minor issue, no id specified**
Result	**Fixed**

Screenshots	 

Retrieve Conversations

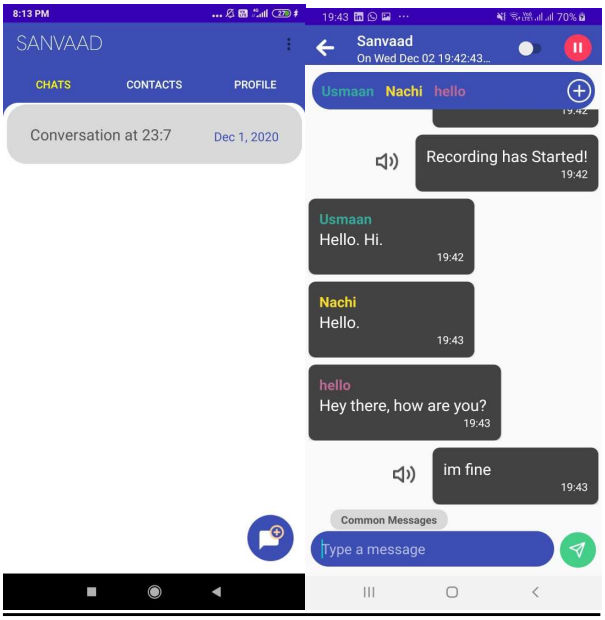
Test Case ID	**7**
Name	Retrieve Conversations
Module	UI-Main Funtaionalitis
Description	Chats are stored locally on device Database. Chats for each user must only be available to them, since chats are private data
Test Procedure	Check on Chat Tab in home.
Expected Result	Only user's chats are visible
Actual Result	Failed
Actual Result Description	Users chats are visible to any user who signs in on the same device.
Screenshots	

- Fix-Retrieve Conversations

Test Case ID	**Fix-4**
Name	**FIX-**Retrieve Conversations
Commit ID	**10b95182 safeer2978**

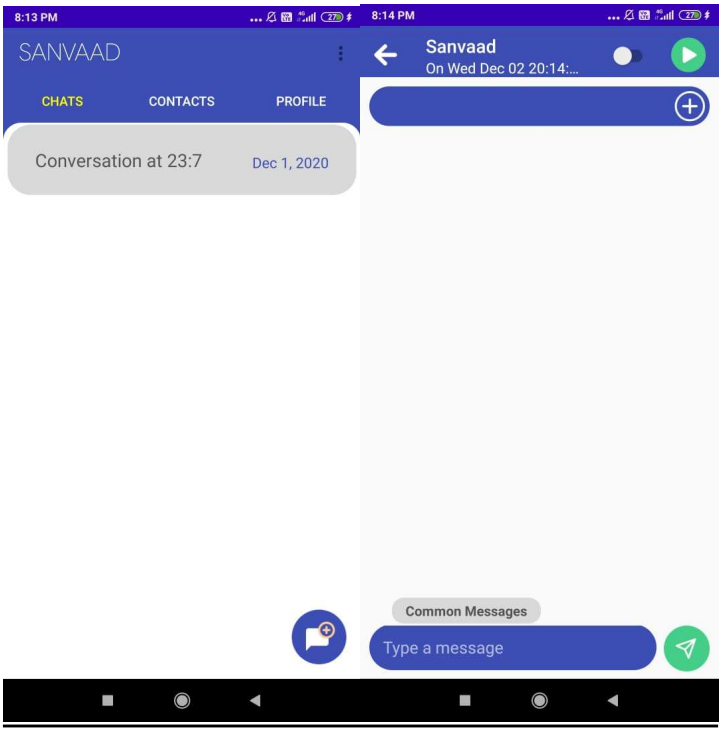
Description	**Id not specified, minor issue**
Result	**Fixed**

- Chats Button

Test Case ID	8
Name	Chats Button
Module	UI-Main Functionality
Description	List all the Previous conversations which can be individually accessible.
Test Procedure	1. Login and go to the chats tab. 2. Click on any of your previous chats.
Expected Result	After accessing your previous conversations from the Chats tab you should be displayed with your previous conversations but you can't modify the chats.
Actual Result	**Passed**
Actual Result Description	It successfully shows my previous conversation.
Screenshots	

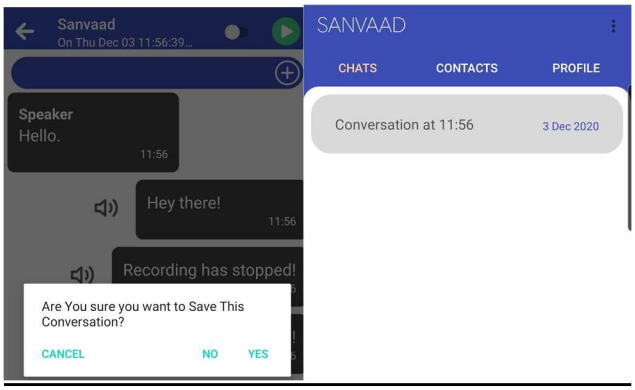
- New Conversation

Test Case ID	9
Name	New Conversation
Module	UI-Main Functionality

Description	It Initiates a new chat room where you could use the main chat functionality.(Speech to text and text to speech)
Test Procedure	1. On the home screen press the New Conversation button on the bottom right. 2. Start a conversation.
Expected Result	It should take the user to a new chatroom
Actual Result	**Passed**
Actual Result Description	After clicking that button it takes the user to a new chatroom where we can use the functionality of the app.
Screenshots	

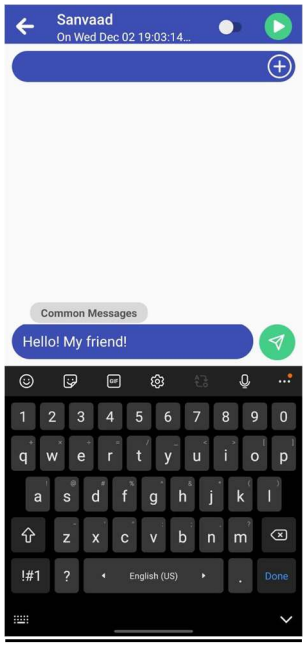
- Exit Chat

Test Case ID	**10**
Name	Exit Chat
Module	UI-Chat
Description	This button allows the user to exit a chat and go back to the home screen of the app while giving an option to save the conversation.

Test Procedure	1. After finishing a conversation click on the back button. 2. Select “Yes” to save the conversation or “No” to discard it.
Expected Result	As the user presses the button the app should give a prompt to the user, if they want to save the conversation for viewing it later.
Actual Result	**Passed**
Actual Result Description	It was observed that when a user exits the conversation a prompt shows up asking the user to either save or discard the conversation. The same can be referred to in the screenshots below.
Screenshots	

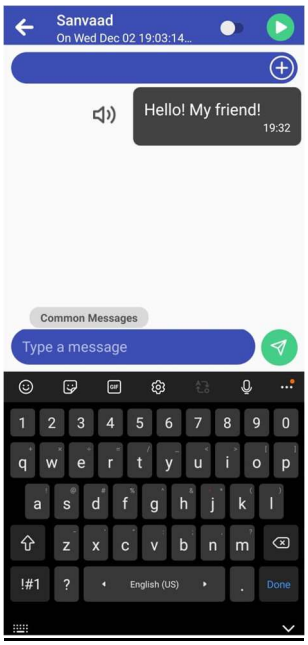
- Type Message

Test Case ID	**11**
Name	Type Message
Module	UI-Chat
Description	Should open the devices keyboard and allow the user to type in a message for the conversation.
Test Procedure	1. Click on the “Type a message” button. 2. Use the keyboard to type a message in the chat.
Expected Result	Messages written by the user should be visible in the type “Type a message” button and ready to be sent.
Actual Result	**Passed**

Actual Result Description	One can notice that the user is able to type a message and review the message before sending it which was our expected result.
Screenshots	

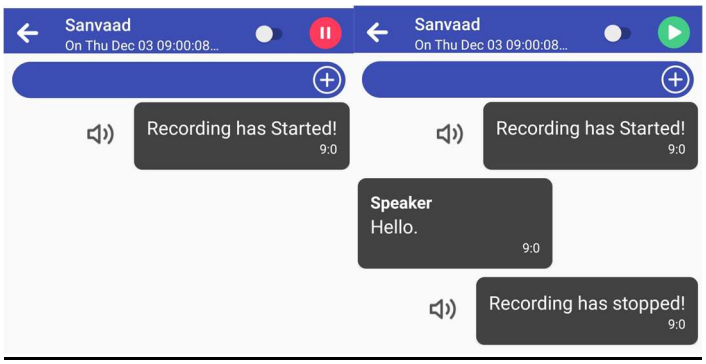
- Send Message

Test Case ID	**12**
Name	Send Message
Module	UI-Chat
Description	Allows the user to send the message which is already typed in the type message button to the chat.
Test Procedure	1. Type a message in the type message button. 2. Click on the send message button.
Expected Result	Messages written by the user should be sent into the chat .
Actual Result	**Passed**
Actual Result Description	As one can see in the screenshot below the message typed by the user is sent to the chat which was our expected result.

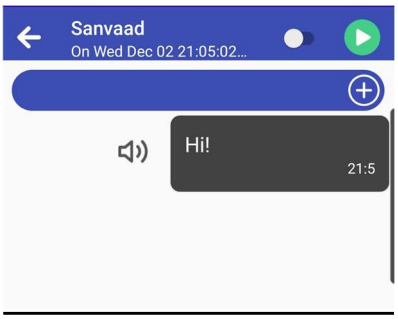
Screenshots	

- Speech-to-text button

Test Case ID	**13**
Name	Speech-to-text
Module	UI-Chat
Description	This button will listen to the voice of the speaker and interpret their words and send them in the chat as a message .
Test Procedure	1. Press the Play button on the top-right corner of your screen. 2. Ask the speaker to speak something near the phone. 3. See if what they say appears as a message in the chat.
Expected Result	The speech-to-text feature should turn on when the user presses the play button and stop when he/she presses the pause button and also provide the text for what the speaker spoke.
Actual Result	**Passed**
Actual Result Description	As soon as the “Play” button is pressed the app starts listening to the speaker voice and stops when the user presses “pause” and also provides the text output for the speech input which was the expected result.

Screenshots	

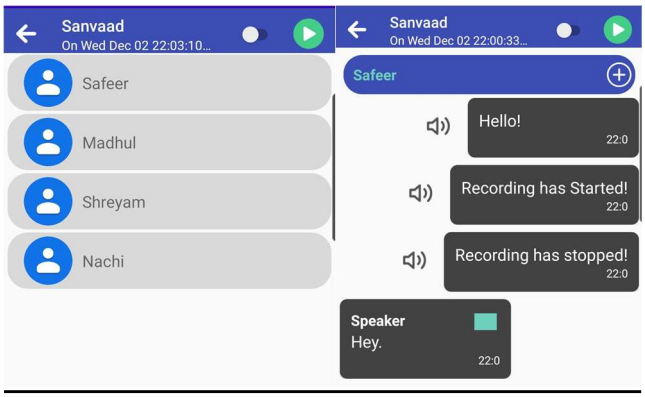
- Text-to-Speech Button

Test Case ID	**14**
Name	Text-to-Speech Button
Module	UI-Chat
Description	Whenever the user clicks on the text to speech button the app reads aloud the text sent by the user.
Test Procedure	1. Type a message in the type message button. 2. Click on the send message button. 3. Click on the text-to-speech button.
Expected Result	The message respective to which the text-to-speech button is clicked is read aloud and can be heard as a voice from the device speaker.
Actual Result	**Passed**
Actual Result Description	When clicked on the text-to-speech button a voice is heard reading the message from the chat which was the desired outcome.
Screenshots	

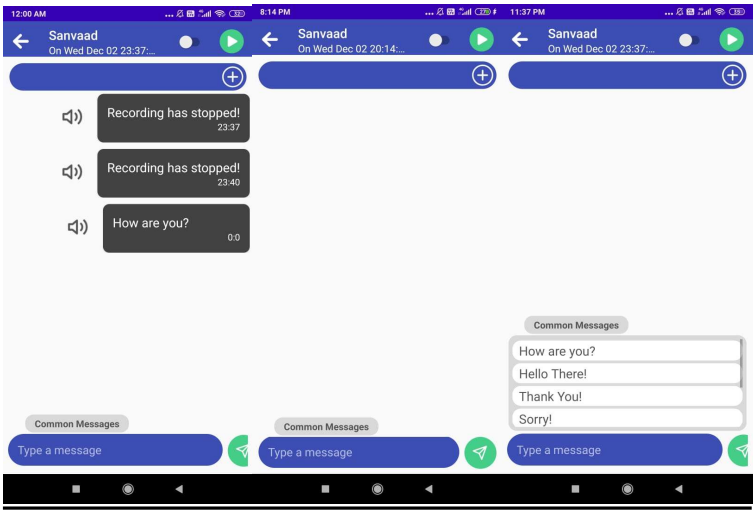
- Auto text-to-speech

Test Case ID	**15**
Name	Auto text-to-speech
Module	UI-Chat
Description	When this toggle is in “on” position all the texts sent to the chat by the user are automatically read out without pressing the text-to-speech button.
Test Procedure	1. Toggle the auto text-to-speech in “on” state. 2. Send a few texts to the chat.
Expected Result	The auto text-to-speech should read aloud the messages which are sent after it is turned on without the use of text-to-speech button. It should not read all the messages of the conversation.
Actual Result	**Passed**
Actual Result Description	It was observed that when the toggle is turned on the texts in the chat which were sent after it were read aloud which was the desired result.
Screenshots	

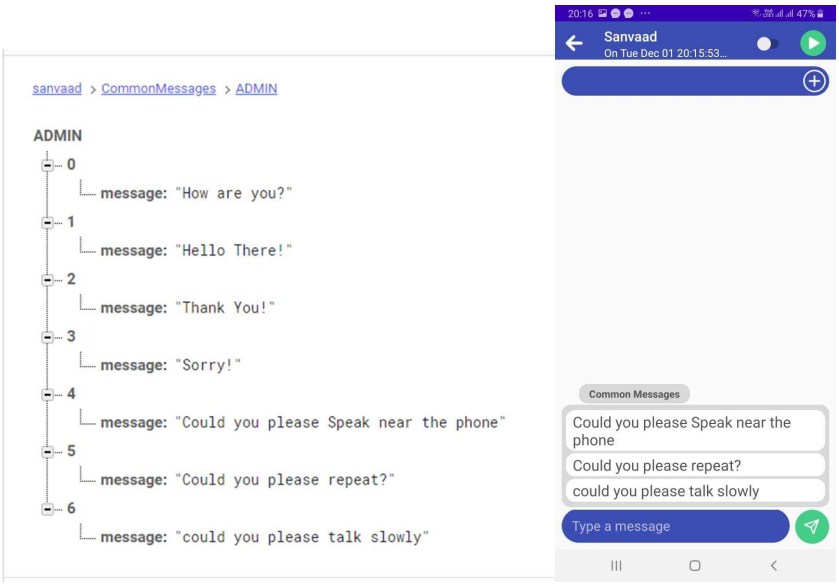
- Add Chat Contacts

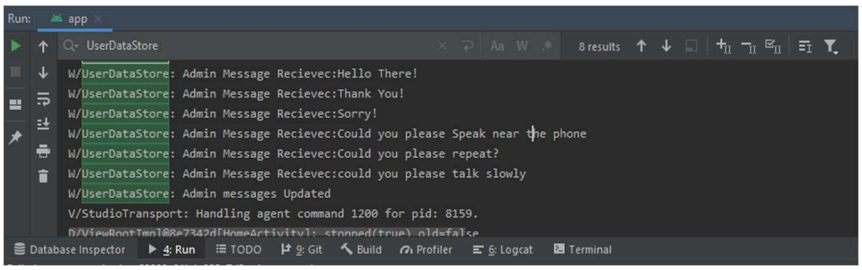
Test Case ID	**16**
Name	Add Chat Contacts
Module	UI-Chat
Description	List of all the contacts added by the user should be received and the user is able to choose from the contacts whom to add to the chat conversation.
Test Procedure	1. Click on the add contact button in the chat activity screen. 2. From the list of contacts, select a contact to add to the conversation.
Expected Result	The chosen contact should be added as a member of the conversation.
Actual Result	**Passed**
Actual Result Description	By referring to the screenshots of the app given below one can notice that a contact from the list of contacts is added to the conversation.
Screenshots	

- Select Common Message

Test Case ID	**17**
Name	Select Common Message
Module	UI-Main Functionality
Description	It is used to use any pre recorded messages or we can even add common messages.
Test Procedure	1. Start a new conversation. 2. Click on the common message tab and select a message.
Expected Result	It is expected to send the selected message in the chat room.
Actual Result	**Passed**
Actual Result Description	We got the list of the pre recorded messages.
Screenshots	

- Admin Common Message

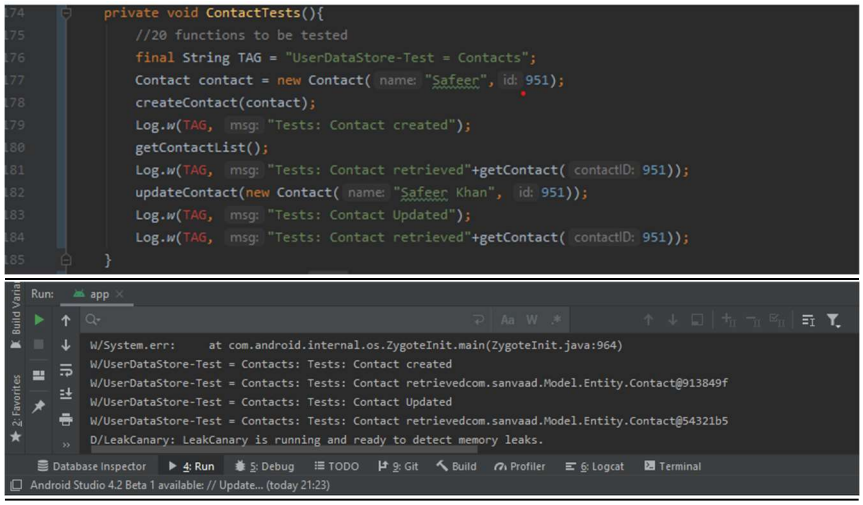
Test Case ID	**18**
Name	Admin Common messages
Module	DataStore-Chat
Description	List of All Common messages, added by the user and the Admin must be retrieved from the Remote data server
Test Procedure	1. Update the Admin Common Messages. 2. Check if Changes are reflected
Expected Result	Data at Remote DB and Local DB is consistent.
Actual Result	Passed
Actual Result Description	We use Log Cat, which is an android Tool to check if changes are reflected or not. As you can see below, we have added an entry to existing NoSQL DB. The same changes are reflected in phone as well.
Screenshots	

Run:
app
UserDataStore
8 results
W/UserDataStore: Admin Message Recievec:Hello There!
W/UserDataStore: Admin Message Recievec:Thank You!
W/UserDataStore: Admin Message Recievec:Sorry!
W/UserDataStore: Admin Message Recievec:Could you please Speak near the phone
W/UserDataStore: Admin Message Recievec:Could you please repeat?
W/UserDataStore: Admin Message Recievec:could you please talk slowly
W/UserDataStore: Admin messages Updated
V/StudioTransport: Handling agent command 1200 for pid: 8159.
Database Inspector
4: Run
TODO
9: Git
Build
Profiler
6: Logcat
Terminal

Data Model Test Cases:

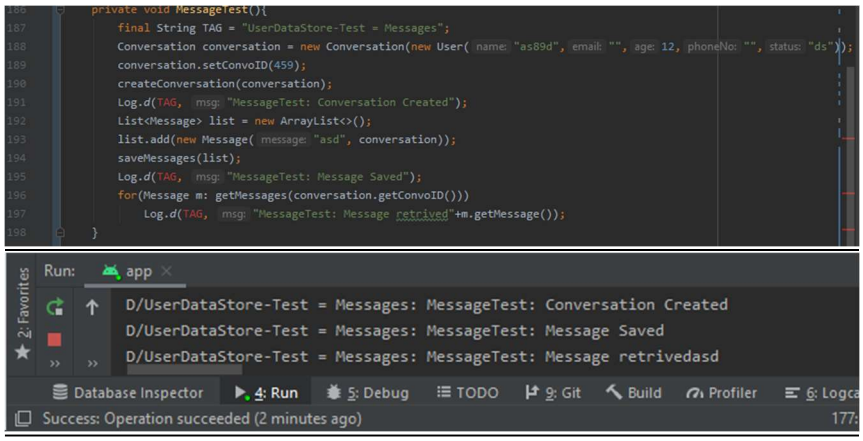
- Contacts Data test

Test Case ID	19
Name	Contacts Data test
Module	DATA-Main functionalities
Description	This case is to test if contacts are being added, saved and retrieved
Test Procedure	Program based approach. Algorithm 1. Create a contact 2. Save Contact 3. update contact 4. Read contact 5. Record Logs
Expected Result	Logs must reflect correct results
Actual Result	Passed
Actual Result Description	Output reflects expected result
Screenshots	

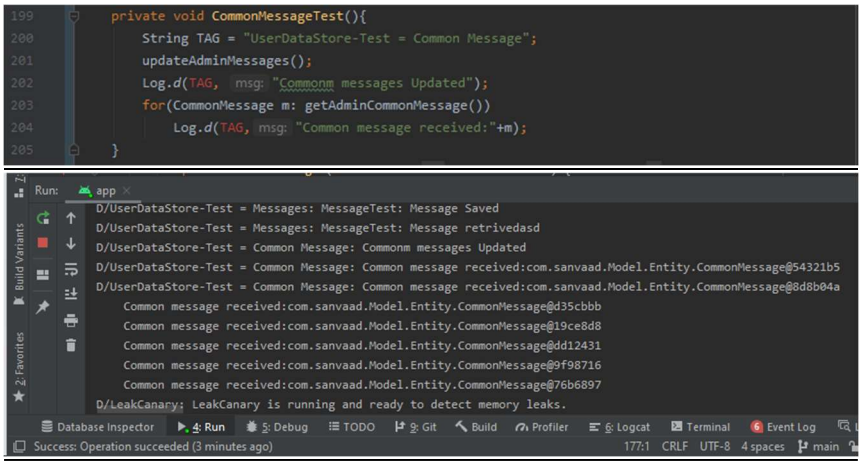
```
private void ContactTests(){
    //20 functions to be tested
    final String TAG = "UserDataStore-Test = Contacts";
    Contact contact = new Contact( name: "Safeer", id: 951);
    createContact(contact);
    Log.w(TAG,  msg: "Tests: Contact created");
    getContactList();
    Log.w(TAG,  msg: "Tests: Contact retrieved"+getContact( contactID: 951));
    updateContact(new Contact( name: "Safeer Khan",  id: 951));
    Log.w(TAG,  msg: "Tests: Contact Updated");
    Log.w(TAG,  msg: "Tests: Contact retrieved"+getContact( contactID: 951));
}
```

```
Run: app
W/System.err:     at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:964)
W/UserDataStore-Test = Contacts: Tests: Contact created
W/UserDataStore-Test = Contacts: Tests: Contact retrievedcom.sanvaad.Model.Entity.Contact@913849f
W/UserDataStore-Test = Contacts: Tests: Contact Updated
W/UserDataStore-Test = Contacts: Tests: Contact retrievedcom.sanvaad.Model.Entity.Contact@54321b5
D/LeakCanary: LeakCanary is running and ready to detect memory leaks.
Database Inspector  4: Run  5: Debug  TODO  9: Git  Build  Profiler  6: Logcat  Terminal
Android Studio 4.2 Beta 1 available: // Update... (today 21:23)
```

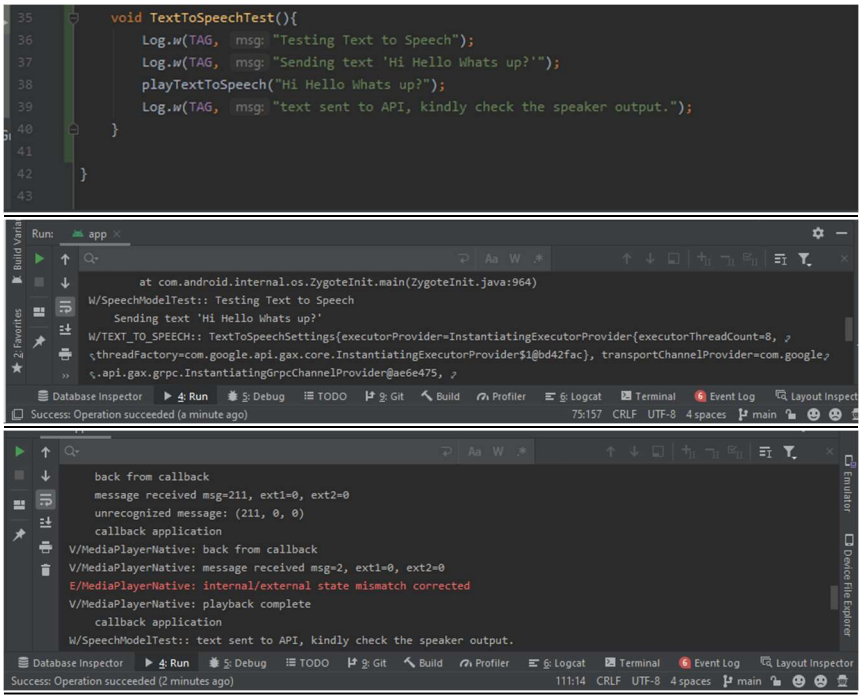
- Message Data test

Test Case ID	20
Name	Message Data test
Module	Data-Chat functionality
Description	We test if the messages are store and read from the database
Test Procedure	We create a test function which follows the following algorithm 1. create a message 2. create a conversation 3. save the conversation and message 4. record the logs 5. read the message from database
Expected Result	Message and Conversation are save and read in Log
Actual Result	Passed
Actual Result Description	Logs reflect the expected result
Screenshots	

- Common Message Test

Test Case ID	21
Name	Common Message Test
Module	Data-Chat functionality
Description	We test if common messages are created, stored and read
Test Procedure	test function that follows the following algorithm 1. Call common message update function 2. make logs
Expected Result	Logs show the result
Actual Result	Passed
Actual Result Description	Logs reflect the expected result.
Screenshots	

- Text-to-Speech Test

Test Case ID	22
Name	Text-to-Speech Test
Module	Data/API - Chat
Description	the text to speech must be tested as per the data flow requirement.
Test Procedure	We create a test function which send a String to the API and then listen to the speaker on the phone for audio.
Expected Result	String sent must be played by the phone speaker.
Actual Result	Passed
Actual Result Description	Message is played from the device speaker
Screenshots	

- Speech to text Test

Test Case ID	23
Name	Speech to text Test
Module	DATA/API- Chat
Description	We test the Speech to text functionality to see if its working.
Test Procedure	We use the app and open the chat activity. We then say “Hi, hello, how are you” to the phone and see if the output comes as expected.
Expected Result	The message gets printed on the screen
Actual Result	Passed
Actual Result Description	Screen shots reflect the actual result.
Screenshots