



**Luxor
University**

Murmuro

Team members:

Abdel-Hafiz Ibrahim
Abdel-Satar Ahmed
Ahmed Youeesf
Mohammed Najeh
Mohammed Asharaf
Omar Atef
Oraby Mahmoud

2 juillet 2020

Luxor University

**Faculty of Computers
and Information**

**Computer Science
Department**



Project Advisors:

**Dr-Mohammed Atta
Khafagy**

Dr-Safynaz Abdelfattah

Sommaire

Titre	1
Sommaire	2
1 Abstract	3
2 Introduction	4
3 Analysis and Design	4
3.1 Updated Functional Requirements	4
3.2 Updated Non-Functional Requirements	5
3.3 Updated Use Case Requirements	6
3.4 Design Classes	10
3.5 Sequence Diagram	10
3.6 Software Architecture	11
4 Prototype description	11
4.1 Implementation Platform	12
4.2 Mapping between requirements and implemented functions	12
4.3 Implementation details	12
4.3.1 Neural Network Model	12
4.3.2 Authorization	13
4.3.3 Chatting	16
4.3.4 Live Translation	22
4.3.5 Application main	24
4.4 Actual database schema	25
5 Testing	25
5.1 Expected test scenarios	25
5.2 Unit test	25
5.3 Functional test	25
5.4 Usability test	25
6 Deployment of the system	25
7 Limitation of the system	25
8 Conclusion and further work	25
9 Appendix	25
9.1 <i>getMessagesAdapter</i> method	25
9.2 <i>onActivityCreated</i> method	41
Bibliographie	46

1 Abstract

Hearing loss, also known as hearing impairment, is a partial or total inability to hear. A deaf person has little to no hearing. Hearing loss may occur in one or both ears. As of 2013 hearing loss affects about 1.1 billion people to some degree. It causes disability in 5% (360 to 538 million) of the world and moderate to severe disability in 124 million people. Most of the deaf cannot read or write and also they cannot communicate with other normal people. Hearing impaired people use sign language to communicate with each other. Sign language employs signs made with the hands and other movements, including facial expressions and postures of the body. There are many different sign languages as, for example, British and American sign languages. In this project we are going to employ machine learning approaches to translate American Sign Language to normal English language to be understood by non-deaf people and also translate normal English to sign language to be understood by deaf people. Also designing is a chat application for deaf/normal people with a live translation feature.

2 Introduction

Signing has always been part of human communications. The use of gestures or signs is not tied to ethnicity, age, or gender. Infants use gestures as a primary means of communication until their speech muscles are mature enough to articulate meaningful speech. For millennia, deaf people have created and used signs among themselves. These signs were the only form of communication available for many deaf people. Within the variety of cultures of deaf people all over the world, signing evolved to form complete and sophisticated languages. These languages have been learned and elaborated by succeeding generations of deaf children. Normally, there is no problem when two deaf persons communicate using their common sign language. The real difficulties arise when a deaf person wants to communicate with a non deaf person. Usually both will get frustrated in a very short time [3] Additionally, these individuals may have difficulty listening in classrooms or conferences, ordering in restaurants, watching TV or movies, listening to music, speaking on the telephone, etc. Current Solutions include communicating with pen and paper ; however, this method is quite slow and inconvenient. Furthermore, Some hard of hearing individuals may have difficulty communicating with written language as there is no commonly used written form of sign language[2] To date, most work on sign language recognition has employed expensive wired "datagloves" which the user must wear [4] In addition, these systems have mostly concentrated on finger signing, in which the user spells each word with finger signs corresponding to the letters of the alphabet [1] However, most signing does not involve finger spelling but instead, gestures which represent whole words, allowing signed conversations to proceed at about the pace of spoken conversation. In this project, we use deep learning methods to build a deep neural network for recognizing American sign language. Our model is trained on a data set of American Sign Language Linguistic Research Project (ASLLRP) of Boston university which is available at <http://www.bu.edu/asllrp/> and ***** **reference data set of dr.Atta ******* In addition a 3D avatar is designed to translate normal english language into American Sign Language. Both the deep neural network model with the 3D avatar constitutes the translation engine. Hence, The translation engine will translate signs into normal English text and translate normal English text into a sign. This translation engine can be deployed in different platforms whether as a desktop software, a web application, or as a mobile application. For a comprehensive demo of how the engine works, we have applied the translation engine in a chat application for mobile devices with a live translation feature.

3 Analysis and Design

In the following sub-sections, we provide the updated design requirements for the translation engine.

3.1 Updated Functional Requirements

Hand and face detection.

This process is required for recognizing a sign from a deaf as it is required to isolate hand and face from the other objects captured from the scene where the deaf exists.

Translating captured sign.

After detecting hand and face from a sequence of frames, the hand and face are passed to the model to be classified and output the label. The outputted label is the translation result and corresponds to the English meaning of the sign.

Providing an animated 3D model (avatar) that will interact with deaf people.

Not all deaf people can read or write and most of the deaf feel more comfortable communicating with their language. Providing an animated avatar would achieve that goal to make the application more interactive, easy and comfortable for use.

Converting text to speech and speech to text.

For a normal user, it is convenient to record a voice to be translated to sign, this process requires that the voice is converted to a text in order to be inputted to the engine so the avatar is animated corresponding to the input text. Further, when the sign is translated into text, the text is converted to speech so a normal person can hear what is being said.

Provide live translation from one language to another.

Live Translation is a feature embedded in the application that allows users to directly translate sign language to speech or speech to sign. This feature is useful in situations where people are talking to each other face to face such as a meeting, or friends conversing with each other.

Chatting.

Chatting is the common way for communicating with friends. The incoming messages are represented by the avatar so deaf can understand it, the avatar will be part of the chatting screen. Also, a deaf can send messages by capturing the hand motion via camera. As most deaf cannot read or write, we provide them with a keyboard that has the representation of a letter in sign language as in Fig 1.

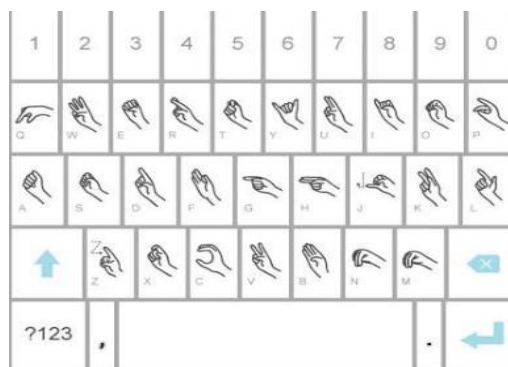


Figure 1. keyboard.

3.2 Updated Non-Functional Requirements

Software Quality Attributes.

- Scalability : Application should be able to provide instant messaging services to many users at any given time. The translation model will retain a large set of common words to achieve elastic translation.
- Robustness : In case a user's device crashes, a backup of their chat history must be stored on remote database servers to enable recoverability. The translation model will be built carefully to a large set of vocabulary to produce unambiguous, clear translation.

- **Reliability** : Application should translate messages correctly and sends messages instantly. The translation model will be updated regularly to achieve better translation accuracy.

Performance Requirements.

The application is light on memory, and will not require any intensive computations. At least 1-MB internet speed is required for chatting. The size of the avatar depends on the it's file format. In table 1 we provide the average size of the avatar, with a dictionary of 60 vocabulary each is 20 frames, for different file formats.

File format	Size in MB
Raw (.blend)	40
FBX (.fbx)	74
Wavefront (.obj)	8.5
Collada (.dae)	17.6
Alembic (.abc)	9
Universal Scene Description (.usd, .usdc, .usda)	14.6
Motion capture (.bvh)	78.5
STL (.stl)	4.5
glTF 2 (.gltf)	14
Rendered video	0.075/video
Rendered GIF	0.090/file

Table 1. Avatar size

***** screenshot for app profiler in android studio *****
 ***** provide application and model size *****
 ***** translation time *****

Safety and Security Requirements.

The application will use end-to-end data encryption to attain confidentiality for data when sent over the internet. Users' data will be encrypted in the database.

*****screenshot of encrypted data in firebase *****

3.3 Updated Use Case Requirements

The full use case diagram of the application system is depicted in the following figure 2.

Use Case1 (Chat).

Purpose : Enabling the user to chat with the others. This includes exchanging text messages, signs, PDFs, pictures, etc.

Requirements Traceability : Enabling users to communicate with their contacts.

Priority : High.

Preconditions : Registered user, Internet connection, open chat.

Post conditions : Users can communicate with each other.

Actors : Deaf/normal user.

Extends : None.

Flow of Events :

1. *Basic Flow* – Once the user logged in application, he/she can see available contacts and

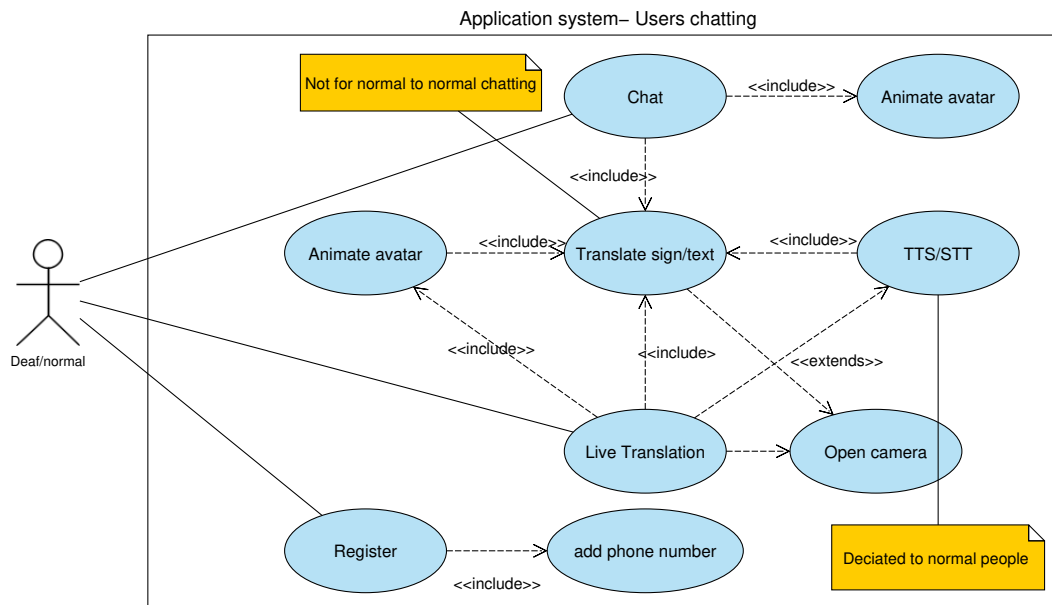


Figure 2. Use case diagram.

chooses one to chat with. If a normal person is chatting with a deaf person, the normal person sends a message then the message is represented by avatar animation at deaf side. If the deaf wants to send a message, he/she enters his/her sign via camera then the sign is translated to text using the translation model.

2. *Alternative Flow* – users can communicate only with text messages without translation from one form to another, or both users can chat with sign.

3. *Exceptions* – Users cannot contact unregistered contacts.

Includes : Translate sign/text, and animate avatar.

Notes/Issues : None.

Use Case2 (Animate avatar).

Purpose : Triggering avatar to perform the motions and gestures so deaf can understand exchanged messages.

Requirements Traceability : Making deaf understand other people.

Priority : High.

Preconditions : Requires an input be formatted English text.

Post conditions : Deaf can understand other people.

Actors : None.

Extends : None.

Flow of Events :

1. *Basic Flow* – The avatar gets as input a valid English text, the avatar then represents each word separately.

2. *Alternative Flow* – If the avatar received an unknown or misspelled English word, the avatar represents them character by character.

3. *Exceptions* – Only text of English characters and numbers is a valid input to the avatar.

Includes : Translate sign/text .

Notes/Issues : None.

Use Case3 (TTS/STT).

Purpose : Convert text to speech and speech to text.

Requirements Traceability : A normal user can get the initiation that a deaf is talking to him.

Priority : High.

Preconditions : The input must be a text or speech.

Post conditions : Normal people can understand the deaf.

Actors : None.

Extends : None.

Flow of Events :

1. *Basic Flow* – The normal user can choose to convert incoming messages to speech so he can hear it instead of read.

2. *Alternative Flow* – The normal user can choose to input speech to communicate with others, so the speech must be converted to text.

3. *Exceptions* – The input must be clear and correct otherwise the output will be ambiguous.

Includes : Translate sign/text.

Notes/Issues : This feature is dedicated only for normal people.

Use Case4 (Live Translation).

Purpose : This use case for translating a text or sign to another format. The input and output is in the same device, no transmission of the data.

Requirements Traceability : Deaf and normal can communicate face to face without an intermediate human translator.

Priority : High.

Preconditions : The input must be a text, speech or sign.

Post conditions : Normal and deaf can understand each other easily.

Actors : Deaf/normal user.

Extends : TTS/STT.

Flow of Events :

1. *Basic Flow* – The user can input a speech to application, the speech is translated to text, text is passed to the translation model, the model passes its output to the avatar, and finally the avatar represents the corresponding sign language.

2. *Alternative Flow* – Deaf users can use the camera to capture the sign, sign input to translator model, translator outputs a text which may be passed to be converted to speech.

3. *Exceptions* – The input must be clear and correct otherwise the output will be ambiguous.

Includes : Animate avatar, translate sign/text, and open camera.

Notes/Issues : None.

Use Case5 (Translate sign/text).

Purpose : Translate from normal English language to American Sign Language and vice versa.

Requirements Traceability : Making deaf and normal people understand each other.

Priority : High.

Preconditions : An input as English text or sequence of image frames represents a sign.

Post conditions : Translated English from sign or translated sign from text.

Actors : None.

Extends : None.

Flow of Events :

1. *Basic Flow* – This use case is to input an English text, pass the text to the translator model,

the model then outputs a text with a specific format representing movements that the avatar will do.

2. *Alternative Flow* – It may input a sequence of image frames that represent a sign, pass the frames to the translator model, then output a sequence of text that represents the corresponding meaning in English.

3. *Exceptions* – Only text and images are the valid format for translation, other formats such as voice is not allowed. Voice data must be converted to text first to be input to the translator model.

Includes : open camera.

Notes/Issues : None.

Use Case6 (Register).

Purpose : Register the user to the system so he/she can use the application.

Requirements Traceability : This is the first step for connecting users to his/her friends.

Priority : High.

Preconditions : Internet connection.

Post conditions : The user is registered at the system.

Actors : Deaf/normal user.

Extends : None.

Flow of Events :

1. *Basic Flow* – The user downloads the application from the market, run application. If it is the first time to run the app, the first screen appears is the registration screen. Otherwise the application will show its default screen.

2. *Alternative Flow* – If no Internet connection, the application will prompt the user to have an internet connection.

3. *Exceptions* – None.

Includes : Add phone number.

Notes/Issues : None.

Use Case7 (Add phone number).

Purpose : Having a unique identifier for registering the user to the system, also used by other users to connect with him.

Requirements Traceability : A complementary step for registering users.

Priority : High.

Preconditions : User should have a valid phone number, Internet connection.

Post conditions : The user is registered at the system.

Actors : Deaf/normal user.

Extends : None.

Flow of Events :

1. *Basic Flow* – The user enters his/her phone number to the required field, then the system verifies that phone number is valid and is owned by the current user by sending a unique number via SMS.

2. *Alternative Flow* – If the user entered a wrong number, he/she is prompted to enter it again and again until verification is done.

3. *Exceptions* – None.

Includes : None.

Notes/Issues : None.

3.4 Design Classes

NOT YET
WAIT ORABY

3.5 Sequence Diagram

The sequence diagram of the application is depicted in figure 3.

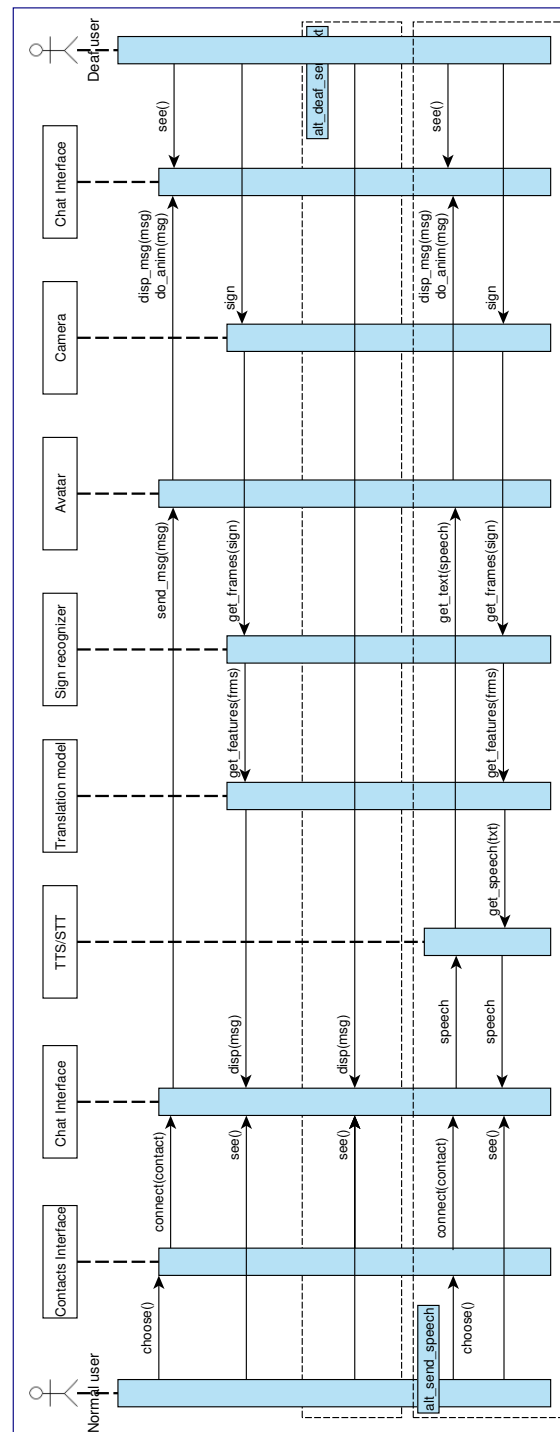


Figure 3. Sequence Diagram.

3.6 Software Architecture

The context diagram of the application is depicted in figure 4. When the deaf user sends a message to an ordinary person, his message is passed to ML model, which is the neural network model that translates sign into text, then the translated message is sent to other side. On the other hand when an ordinary user sends a message to a deaf, the message is passed to the avatar, which is a rigged character used for expressing the sign language. Message exchanging is done between the end users through chat interface, on the other hand the chat interface, as part of the whole application, uses the engine to get proper translation. All sent messages and media are saved in the Firebase. The Firebase stores the chat between end user in a text format, meaning that no signs or avatar's animation are stored but normal-encrypted text. Other media formats like images, sound voices and documents are saved in it's original format. WRITE THAT YOU USED C4 MODEL

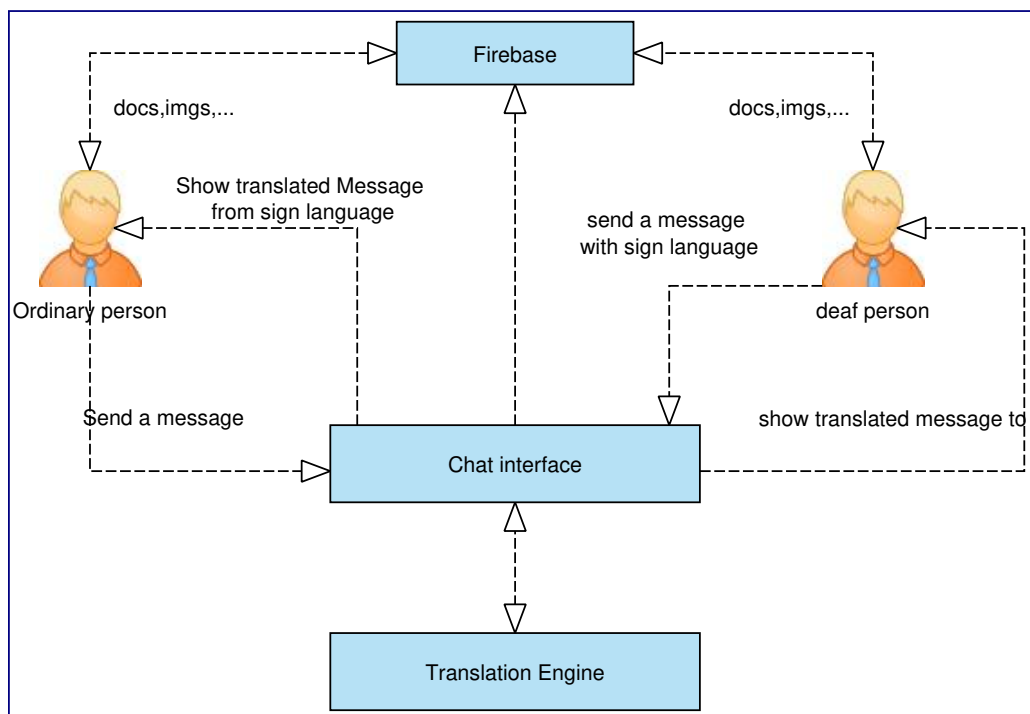


Figure 4. Use case diagram.

4 Prototype description

Our major prototype is the chat application that enables the deaf to chat with other deaf or normal people. The chat app also contains a live translation feature to enable two users to translate a sign into text or vice versa when the users are communicating face to face. A secondary prototype is a desktop application that runs translation engine only. This prototype is used for evaluating the performance and the accuracy of the engine whether translating sign into English text or English text into signs.

4.1 Implementation Platform

For the chat app, it is implemented in Android studio and Java language. The app is provided for the any mobile platform that runs Android, IOS or Windows phone. The released version of the application is for Andorid phones with Android version 4.4 and higher. Meanwhile the desktop application is deployed for Windows 7, 8, 8.1 and 10 and is implemented in visual studio and c# language with WPF GUI design scheme. The main requirements used to running the engine properly is that the platform have at least medium-quality camera and is capable of running 3D graphics models.

4.2 Mapping between requirements and implemented functions

4.3 Implementation details

4.3.1 Neural Network Model

The embedding of the Neural Network model, which it is original format is H5, requires transforming this model format into another format suitable for mobile application; which is so called tensorflow lite. Running tensorflow lite models in mobile application requires implementing The *Classifier* interface. *TensorFlowImageClassifier* is a concrete class of *Classifier* interface and we summarize the main methods below : **create()** : This method is used to load the model from application assets and returns a classifier object.

```
1 public static Classifier create(AssetManager assetManager,
2                               String modelPath,
3                               String labelPath,
4                               int inputSize,
5                               boolean quant) throws IOException {
6
7     TensorFlowImageClassifier classifier = new TensorFlowImageClassifier();
8     classifier.interpreter = new Interpreter(classifier.loadModelFile(assetManager,
9 modelPath), new Interpreter.Options());
10    classifier.labelList = classifier.loadLabelList(assetManager, labelPath);
11    classifier.inputSize = inputSize;
12    classifier.quant = quant;
13
14    return classifier;
15 }
```

recognizeImage() :The role of this method is handling the input image coming from the camera, the images are stored in a *ByteBuffer* and then returned in form of *List* object of type *Recognition*. The returned object from this method comprises the sign translation.

```
1 public List<Recognition> recognizeImage(Bitmap bitmap) {
2     ByteBuffer byteBuffer = convertBitmapToByteBuffer(bitmap);
3     if(quant){
4         float[][] result = new float[1][labelList.size()];
5         interpreter.run(byteBuffer, result);
6         return getSortedResultFloat(result);
7     } else {
8         float [][] result = new float[1][labelList.size()];
9         interpreter.run(byteBuffer, result);
10        return getSortedResultFloat(result);
11    }
12
13 }
```

4.3.2 Authorization

Considering the user registration process, the user is required to input his name, email, password, and username. Each of these fields is implemented in a separate class and each class extends *ernestoyaquello.com.verticalstepperform.Step* class. The same methods and the same implementation pattern is applied in *EmailStep* class for email verification, *NameStep* class for getting full name of the use, *PasswordStep* for ensuring a strong and valid password is entered by user, and finally *UserNameStep* class for creating a unique username for each user. The code for each class is listed below.

EmailStep

```
1 @Override
2     protected IsDataValid isStepDataValid(String stepData) {
3         // The step's data (i.e., the user name) will be considered valid only if it is longer
4         // than
5         // three characters. In case it is not, we will display an error message for feedback.
6         // In an optional step, you should implement this method to always return a valid value.
7         boolean isEmailValid = !TextUtils.isEmpty(stepData) && android.util.Patterns.
8             EMAIL_ADDRESS.matcher(stepData).matches();
9         String errorMessage = !isEmailValid ? "Email not valid" : "";
10
11         return new IsDataValid(isEmailValid, errorMessage);
12     }
13
14     @Override
15     public String getStepData() {
16         // We get the step's data from the value that the user has typed in the EditText view.
17         Editable email = EmailView.getText();
18         return email != null ? email.toString() : "";
```

NameStep

```
1 @Override
2     protected IsDataValid isStepDataValid(String stepData) {
3         // The step's data (i.e., the user name) will be considered valid only if it is longer
4         // than
5         // three characters. In case it is not, we will display an error message for feedback.
6         // In an optional step, you should implement this method to always return a valid value.
7         boolean isNameValid = stepData.length() >= 12;
8         String errorMessage = !isNameValid ? "12 characters minimum" : "";
9
10         return new IsDataValid(isNameValid, errorMessage);
11     }
12
13     @Override
14     public String getStepData() {
15         // We get the step's data from the value that the user has typed in the EditText view.
16         Editable name = NameView.getText();
17         return name != null ? name.toString() : "";
```

PasswordStep

```
1 @Override
2     protected IsDataValid isStepDataValid(String stepData) {
3         // The step's data (i.e., the user name) will be considered valid only if it is longer
4         // than
5         // three characters. In case it is not, we will display an error message for feedback.
6         // In an optional step, you should implement this method to always return a valid value.
7         String pattern = "(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=])(?=\S+$).{8,}";
```

```

7      boolean isPasswordValid = stepData.matches(pattern);
8      String errorMessage = !isPasswordValid ? "Passord EX: aaZa44@" : "";
9
10     return new IsPasswordValid(isPasswordValid, errorMessage);
11 }
12
13 @Override
14 public String getStepData() {
15     // We get the step's data from the value that the user has typed in the EditText view.
16     Editable password = PasswordView.getText();
17     return password != null ? password.toString() : "";
18 }

```

UserNameStep

```

1 @Override
2 protected IsDataValid isStepDataValid(String stepData) {
3     // The step's data (i.e., the user name) will be considered valid only if it is longer
4     // than
5     // three characters. In case it is not, we will display an error message for feedback.
6     // In an optional step, you should implement this method to always return a valid value.
7     boolean isNameValid = !TextUtils.isEmpty(stepData) && android.util.Patterns.
8     EMAIL_ADDRESS.matcher(stepData).matches();
9     String errorMessage = !isNameValid ? "name@exm.com" : "";
10
11     return new IsDataValid(isNameValid, errorMessage);
12 }
13
14 @Override
15 public String getStepData() {
16     // We get the step's data from the value that the user has typed in the EditText view.
17     Editable userName = userNameView.getText();
18     return userName != null ? userName.toString() : "";
19 }

```

Verifying mobile phone number is more different and complex than verifying the name, username, password and email of the user. As a result phone number verification and confirmation is handled in a separate package. The class *MobileNumber* comprises the mobile number object's attributes in the application, meanwhile the *Confirmation* class is responsible for confirming that the user uses a valid phone number by sending a verification code to the user. The sending of verification below is implemented as below :

```

1 private void sendVerificationCode(String number){
2     Toast.makeText(getContext(), getString(R.string.sending_code)+"" ,Toast.LENGTH_SHORT).
3     show();
4     PhoneAuthProvider.getInstance().verifyPhoneNumber(
5         number,
6         60,
7         TimeUnit.SECONDS,
8         TaskExecutors.MAIN_THREAD,
9         mCallBack
10    );
11 }
12
13 private PhoneAuthProvider.OnVerificationStateChangedCallbacks
14     mCallBack = new PhoneAuthProvider.OnVerificationStateChangedCallbacks() {
15
16     @Override
17     public void onCodeSent(String s, PhoneAuthProvider.ForceResendingToken
18     forceResendingToken) {
19         super.onCodeSent(s, forceResendingToken);
20         verificationid = s;
21     }
22 }

```

```

19     }
20
21     @Override
22     public void onVerificationCompleted(PhoneAuthCredential phoneAuthCredential) {
23         String code = phoneAuthCredential.getSmsCode();
24         if (code != null){
25             binding.progress.setVisibility(View.VISIBLE);
26             verifyCode(code);
27         }
28     }
29
30     @Override
31     public void onVerificationFailed(FirebaseException e) {
32
33         Toast.makeText(getContext(), "there is " + e.getMessage(), Toast.LENGTH_LONG).show();
34         Log.e(TAG, "onVerificationFailed: " + e.getMessage() );
35
36     }
37 };

```

The confirmation requires that the user enter his own number, then the Firebase creates a temporal record for the user and creates a verification code that user must input to authorize him. The authorization is implemented by creating a credential with the verification code as listed below :

```

1 private void verifyCode(String code){
2     PhoneAuthCredential credential = PhoneAuthProvider.getCredential(verificationid, code);
3     signInWithCredential(credential);
4 }

```

The authorization process is limited for 2 minutes only, if time is elapsed then the generated credential and verification code is deleted and the user must choose to resend the code again. The timer function is implemented as follows

```

1 private void timer() {
2     binding.resendTx.setVisibility(View.GONE);
3     countdownTimer = new CountdownTimer(120000, 1000) {
4         public void onTick(long millisUntilFinished) {
5
6             long timer = (millisUntilFinished / 1000);
7             long min = timer / 60;
8
9             long sec = timer - (min * 60);
10
11             binding.timerTx.setText(getString(R.string.wait_we_will_resend_code_agian_after)
12 + "\n " + String.format(min + " ", "00") + ":" + String.format(sec + " ", "00"));
13         }
14
15         public void onFinish() {
16             binding.resendTx.setVisibility(View.VISIBLE);
17             binding.progress.setVisibility(View.GONE);
18         }
19     };
20
21     countdownTimer.start();
22 }

```

After a successful registration step, the user can log in the application in any time. The log in process involves ensuring that the user is already registered to the Firebase and ensuring a valid username and password are entered. The following method of class *LogIn* do the main

work as follows :

```
1 public void ObserveLogin() {
2     String username = binding.usernameEt.getText().toString();
3     String password = binding.passwordEt.getText().toString();
4     String pattern = "(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=])(?=\S+$).{8,}";
5
6     if (TextUtils.isEmpty(username) && android.util.Patterns.EMAIL_ADDRESS.matcher(username).
7 matches()) {
8         binding.usernameEt.setError("Invalid User name");
9         return;
10    }
11
12    if (!password.matches(pattern)) {
13        binding.passwordEt.setError("Invalid Password");
14        return;
15    }
16    binding.progressBar.setVisibility(View.VISIBLE);
17
18    mViewModel.Login(username, password).observe(this, new Observer<AuthResource<User>>() {
19        @Override
20        public void onChanged(AuthResource<User> userAuthResource) {
21            if (userAuthResource != null) {
22                switch (userAuthResource.status) {
23                    case NOT_AUTHENTICATED: {
24                        binding.progressBar.setVisibility(View.GONE);
25                    }
26
27                    case AUTHENTICATED: {
28                        binding.progressBar.setVisibility(View.GONE);
29                        Intent intent = new Intent(getContext(), MainActivity.class);
30                        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.
31 FLAG_ACTIVITY_NEW_TASK);
32                        startActivity(intent);
33                        getActivity().finish();
34                    }
35
36                    case LOADING: {
37                        binding.progressBar.setVisibility(View.VISIBLE);
38                    }
39
40                    case ERROR: {
41                        binding.progressBar.setVisibility(View.GONE);
42                        Toast.makeText(getContext(), userAuthResource.message, Toast.
43 LENGTH_SHORT).show();
44                    }
45                }
46            }
47        }
48    });
49 }
```

4.3.3 Chatting

The core of the application is chatting between a deaf person and the an ordinary person. The chat functionality is modeled in three classes. The first class is *Chat* which describes the essence of the chat functionality and how it is bined to the user interface. Second class is *ChatAdapter* which is a concrete class of *RecyclerView.Adapter<ChatAdapter.MyViewHolder>*; this class describes the inflating of the view objects on the current chat activity. The third and most critical class of the chat functionality is *ChatViewModel* this class is the backbone that the chat process relies on; it contains the methods which describes how message and other media is transferred from one user to another, how the translation is

handled at both sides. In addition, this class also controls the synchronicity between Firebase data and the data displayed at each user. The following method retrieves the current user's data from Firebase :

```
1 public MutableLiveData<DataResource<User>> getCurrentUserDataResourceMutableLiveData() {
2     currentUserDataResourceMutableLiveData.setValue(DataResource.loading((User) null));
3     murmuroRepositoryImp.getUserById(firebaseAuth.getCurrentUser().getUid())
4         .subscribeOn(Schedulers.io())
5         .observeOn(AndroidSchedulers.mainThread())
6         .onErrorReturn(new Function<Throwable, User>() {
7             @Override
8             public User apply(Throwable throwable) throws Exception {
9                 User user = new User();
10                user.setId("-1");
11                return user;
12            }
13        }).map(new Function<User, Object>() {
14            @Override
15            public Object apply(User user) throws Exception {
16                if (user.getId().equals("-1")) {
17                    currentUserDataResourceMutableLiveData.setValue(DataResource.error("can not
load user", (User) null));
18                    return null;
19                }
20                currentUserDataResourceMutableLiveData.setValue(DataResource.success(user));
21                return user;
22            }
23        }).subscribe();
24     return currentUserDataResourceMutableLiveData;
25 }
```

When the user select a contact to view or begin a chat, the conversation is retrieved from the Firebase and displayed in the current chat activity, the method below do the job. A database reference is created to point to the proper conversation, then the conversation is downloaded. Note that Internet connection is required to accomplish this process.

```
1 public MutableLiveData<DataResource<Conversation>> getConversationDataResourceMutableLiveData(
2     String conversationId) {
3     conversationDataResourceMutableLiveData.setValue(DataResource.loading((Conversation)
4     null));
5     if (isInternetAvailable()) {
6         DatabaseReference databaseReference = firebaseDatabase.getReference().child("
7         conversations").child(conversationId);
8         databaseReference.addValueEventListener(new ValueEventListener() {
9             @Override
10            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
11                Log.e(TAG, "onDataChange: " + dataSnapshot.getValue().toString());
12                Conversation conversation = dataSnapshot.getValue(Conversation.class);
13                murmuroRepositoryImp.updateConversation(conversation);
14                conversationDataResourceMutableLiveData.setValue(DataResource.success(
15                conversation));
16            }
17            @Override
18            public void onCancelled(@NonNull DatabaseError databaseError) {
19                conversationDataResourceMutableLiveData.setValue(DataResource.error(
20                databaseError.getMessage(), (Conversation) null));
21            }
22        });
23    } else if (!isInternetAvailable()) {
24        murmuroRepositoryImp.getConversationById(conversationId)
25            .subscribeOn(Schedulers.io())
26            .observeOn(AndroidSchedulers.mainThread())
27    }
```

```

23         .onErrorReturn(new Function<Throwable, Conversation>() {
24             @Override
25             public Conversation apply(Throwable throwable) throws Exception {
26                 Conversation conversation = new Conversation();
27                 conversation.setId("-1");
28                 return null;
29             }
30         })
31         .map(new Function<Conversation, Object>() {
32             @Override
33             public Object apply(Conversation conversation) throws Exception {
34                 if (conversation.getId().equals("-1")) {
35                     conversationDataResourceMutableLiveData.setValue(DataResource.
error("Can not load conversation", (Conversation) null));
36                     return null;
37                 }
38                 conversationDataResourceMutableLiveData.setValue(DataResource.
success(conversation));
39                 return conversation;
40             }
41         }).subscribe();
42     }
43     return conversationDataResourceMutableLiveData;
44 }

```

Different media ,like voice, images, videos, and other files, are handled differently than normal text messages. This involves locating/recording the media; then a set of database references are created to synchronize the media states(seen, not seen, sent, failed,...) between users. This is achieved by the following method :

```

1 public void sendStorageMessage(final Message message, final Person friendUser, final String
    conversatId, final long messagesSize) {
2     StorageReference ref = null;
3
4     if (message.getMessageType().equals("File")) {
5         ref = firebaseStorage.getReference().child("files/" + conversatId + "/" + message.
getDateTime() + message.getText());
6
7     } else if (message.getMessageType().equals("Audio")) {
8         ref = firebaseStorage.getReference().child("audios/" + conversatId + "/" + message.
getDateTime() + message.getText());
9
10    } else if (message.getMessageType().equals("Video")) {
11        ref = firebaseStorage.getReference().child("videos/" + conversatId + "/" + message.
getDateTime() + message.getText());
12
13    } else if (message.getMessageType().equals("Gif")) {
14        ref = firebaseStorage.getReference().child("gifs/" + conversatId + "/" + message.
getDateTime() + message.getText());
15
16    } else if (message.getMessageType().equals("Photo")) {
17        ref = firebaseStorage.getReference().child("images/" + conversatId + "/" + message.
getDateTime() + message.getText());
18    }
19
20    ref.putFile(Uri.parse(message.getPhoto()))
21        .addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
22            @Override
23            public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
24                Log.e(TAG, "onSuccess: Uploaded");
25                Toast.makeText(context, "Start Uploading", Toast.LENGTH_SHORT).show();
26
27                DatabaseReference databaseReference1 = firebaseDatabase.getReference()

```

```

28         .child("conversations")
29         .child(conversatId)
30         .child("lastMessageId");
31
32     databaseReference1.setValue(messagesSize + "");
33
34     DatabaseReference databaseReference2 = firebaseDatabase.getReference()
35         .child("conversations")
36         .child(conversatId)
37         .child("messages")
38         .child(((messagesSize) + ""));
39
40     databaseReference2.setValue(message);
41
42     final DatabaseReference databaseReference3 = firebaseDatabase.
getReference()
43         .child("conversations")
44         .child(conversatId)
45         .child("unreadMessages")
46         .child(friendUser.getId());
47
48
49     databaseReference3.addValueEventListener(new ValueEventListener() {
50         @Override
51         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
52             unreadMessages = dataSnapshot.getValue(Integer.class);
53             unreadMessages++;
54         }
55
56         @Override
57         public void onCancelled(@NonNull DatabaseError databaseError) {
58             Log.e(TAG, "onCancelled: " + "can not load unread messages");
59         }
60     });
61
62     databaseReference3.setValue(unreadMessages);
63
64     DatabaseReference databaseReference4 = firebaseDatabase.getReference()
65         .child("users")
66         .child(friendUser.getId())
67         .child("conversations")
68         .child(firebaseAuth.getCurrentUser().getUid())
69         .child("unreadMessages")
70         .child(friendUser.getId());
71
72     databaseReference4.setValue(unreadMessages);
73
74     DatabaseReference databaseReference5 = firebaseDatabase.getReference()
75         .child("users")
76         .child(friendUser.getId())
77         .child("conversations")
78         .child(firebaseAuth.getCurrentUser().getUid())
79         .child("displayMessage");
80
81     databaseReference5.setValue(message);
82
83     DatabaseReference databaseReference6 = firebaseDatabase.getReference()
84         .child("users")
85         .child(firebaseAuth.getCurrentUser().getUid())
86         .child("conversations")
87         .child(friendUser.getId())
88         .child("displayMessage");
89

```

```

90         databaseReference6.setValue(message);
91         Toast.makeText(context, "Uploaded Success", Toast.LENGTH_SHORT).show();
92     }
93 }
94 .addOnFailureListener(new OnFailureListener() {
95     @Override
96     public void onFailure(@NonNull Exception e) {
97         Log.e(TAG, "onFailure: Failed");
98         Toast.makeText(context, "Uploaded : Failed", Toast.LENGTH_SHORT).show();
99     }
100 })
101 .addOnProgressListener(new OnProgressListener<UploadTask.TaskSnapshot>() {
102     @Override
103     public void onProgress(UploadTask.TaskSnapshot taskSnapshot) {
104         double progress = (100.0 * taskSnapshot.getBytesTransferred() /
taskSnapshot
105             .getTotalByteCount());
106         Log.e(TAG, "onProgress: Uploaded" + (int) progress + "%");
107     }
108 });
109 }

```

Sending a text message is same as sending media in establishing the proper database references. However, it's simpler as it gets the input directly from the user from the keyboard and does not require locating external files or recording any data. The method below summarizes this process.

```

1 public void sendTextMessage(final Message message, final Person friendUser, final String
    conversatId, long messagesSize) {
2     DatabaseReference databaseReference1 = firebaseDatabase.getReference()
3         .child("conversations")
4         .child(conversatId)
5         .child("lastMessageId");
6
7     databaseReference1.setValue(messagesSize + "");
8
9     DatabaseReference databaseReference2 = firebaseDatabase.getReference()
10        .child("conversations")
11        .child(conversatId)
12        .child("messages")
13        .child(((messagesSize) + ""));
14
15
16    databaseReference2.setValue(message);
17
18    final DatabaseReference databaseReference3 = firebaseDatabase.getReference()
19        .child("conversations")
20        .child(conversatId)
21        .child("unreadMessages")
22        .child(friendUser.getId());
23
24
25    databaseReference3.addValueEventListener(new ValueEventListener() {
26        @Override
27        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
28            unreadMessages = dataSnapshot.getValue(Integer.class);
29            unreadMessages++;
30        }
31
32        @Override
33        public void onCancelled(@NonNull DatabaseError databaseError) {
34            Log.e(TAG, "onCancelled: " + "can not load unread messages");
35        }
36    });
37 }

```

```

36     });
37
38     databaseReference3.setValue(unreadMessages);
39
40     DatabaseReference databaseReference4 = firebaseDatabase.getReference()
41         .child("users")
42         .child(friendUser.getId())
43         .child("conversations")
44         .child(firebaseAuth.getCurrentUser().getUid())
45         .child("unreadMessages")
46         .child(friendUser.getId());
47
48     databaseReference4.setValue(unreadMessages);
49
50     DatabaseReference databaseReference5 = firebaseDatabase.getReference()
51         .child("users")
52         .child(friendUser.getId())
53         .child("conversations")
54         .child(firebaseAuth.getCurrentUser().getUid())
55         .child("displayMessage");
56
57     databaseReference5.setValue(message);
58
59     DatabaseReference databaseReference6 = firebaseDatabase.getReference()
60         .child("users")
61         .child(firebaseAuth.getCurrentUser().getUid())
62         .child("conversations")
63         .child(friendUser.getId())
64         .child("displayMessage");
65
66     databaseReference6.setValue(message);
67
68 }

```

Interpolating the various functions of text and media exchanging, translating a message into a sign using avatar, capturing user's sign are done using a very long adapter method. This method is listed below as individual code snippets :

In this part, the database references are created to point to a target messages, and a page of the conversation is prepared to be displayed on the screen.

```

1  public MutableLiveData<DataResource<FirebaseRecyclerPagingAdapter<Message, ChatAdapter.
2  MyViewHolder>>> getMessagesAdapter(final String conversationId, final Person friendUser,
3  final LifecycleOwner lifecycleOwner) {
4
5      DatabaseReference messagesReference = firebaseDatabase.getReference()
6          .child("conversations")
7          .child(conversationId)
8          .child("messages");
9
10     PagedList.Config config = new PagedList.Config.Builder()
11         .setEnablePlaceholders(false)
12         .setPrefetchDistance(5)
13         .setPageSize(10)
14         .build();
15
16     DatabasePagingOptions<Message> options = new DatabasePagingOptions.Builder<Message>()
17         .setLifecycleOwner(lifecycleOwner)
18         .setQuery(messagesReference, config, Message.class)
19         .build();
20
21     FirebaseRecyclerPagingAdapter<Message, ChatAdapter.MyViewHolder>

```

```

21 firebaseRecyclerPagingAdapter = new FirebaseRecyclerPagingAdapter<Message, ChatAdapter.
    MyViewHolder>(options) {...}
    messagesAdapterDataResourceMutableLiveData.setValue(DataResource.success(
    firebaseRecyclerPagingAdapter));
22
23     return messagesAdapterDataResourceMutableLiveData;
24 }

```

The code of *FirebaseRecyclerPagingAdapter* is collapsed in the previous code snippet and is detailed in the next code fragments. *FirebaseRecyclerPagingAdapter* Library binds Firebase Realtime Database Query to a *RecyclerView* by loading Data in pages. This is enable real-time data exchange and reliable message translation. Instantiating an object of *FirebaseRecyclerPagingAdapter* requires overriding the following-collapsed methods.

```

1 FirebaseRecyclerPagingAdapter<Message, ChatAdapter.MyViewHolder> firebaseRecyclerPagingAdapter
  = new FirebaseRecyclerPagingAdapter<Message, ChatAdapter.MyViewHolder>(options) {
2     @Override
3     protected void onBindViewHolder(@NonNull final ChatAdapter.MyViewHolder myViewHolder
4                                     ,
5                                     int i, @NonNull final Message message) {...}
6     @Override
7     protected void onLoadingStateChanged(@NonNull LoadingState state){...}
8     @NonNull
9     @Override
10    public ChatAdapter.MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
    viewType){...}
11
12    @Override
13    protected void onError(@NonNull DatabaseError databaseError){...}
14 }

```

Note You can see the full implementation of *getMessagesAdapter* method in the appendix

4.3.4 Live Translation

A second important feature of the application is Live Translation. This feature is used when two users meet up and a translation from one language to the other is required. Live Translation is modeled in the application in the *LiveTranslation* class, which is responsible for inflating layout on the screen and allow the user to switch between sign language translation and English translation. As a normal user can talk directly instead of writing, a method is implemented to handle user's speech :

```

1 private void startVoiceInput() {
2     Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
3     intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.
    LANGUAGE_MODEL_FREE_FORM);
4     intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, "en-US");
5     intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Hello, Say your message?");
6     try {
7         startActivityForResult(intent, REQ_CODE_SPEECH_INPUT);
8     } catch (ActivityNotFoundException a) {
9
10    }
11 }

```

Also, as the deaf user will use sign to communicate with his partner, a method is required for loading Neural Network model to be executable :

```

1 private void initTensorFlowAndLoadModel() {

```

```

2      executor.execute(new Runnable() {
3          @Override
4          public void run() {
5              try {
6                  classifier = TensorFlowImageClassifier.create(
7                      getActivity().getAssets(),
8                      MODEL_PATH,
9                      LABEL_PATH,
10                     INPUT_SIZE,
11                     QUANT);
12              } catch (final Exception e) {
13                  throw new RuntimeException("Error initializing TensorFlow!", e);
14              }
15          }
16      });
17  }

```

Putting engine components to work together and enabling user to switch between Avatar and model is accomplished by the overridden method below. The essence of this method is to bind application's activity with appropriate *Listeners* for each task, for example : running avatar, opening camera to capture sign, record audio from mic.

```

1  @Override
2  public void onActivityCreated(@Nullable Bundle savedInstanceState){
3      super.onActivityCreated(savedInstanceState);
4      mViewModel = ViewModelProviders.of(this,providerFactory).get(LiveTranslationViewModel.
5      class);
6      if(savedInstanceState != null){
7          Navigation.findNavController(getActivity(), R.id.host_fragment).restoreState(
8          savedInstanceState.getBundle("nav_state"));
9          Log.e(TAG, "onRestoreInstanceState: " + savedInstanceState.getBundle("nav_state").
10         describeContents());
11      }
12
13      binding.camera.setLifecycleOwner(this);
14
15      handler = new Handler();
16      runnable = new Runnable() {...};
17      if(handler!= null && runnable != null){...}
18
19      binding.camera.close();
20
21      binding.camera.addCameraListener(new CameraListener(){...});
22
23      initTensorFlowAndLoadModel();
24
25      binding.toolbar.setNavigationOnClickListener(new View.OnClickListener(){...});
26
27      binding.messageEditText.addTextChangedListener(new TextWatcher(){...});
28
29      binding.sendImage.setOnClickListener(new View.OnClickListener(){...});
30      binding.signTranslationButton.setOnClickListener(new View.OnClickListener(){...});
31  }

```

Note You can see the full implementation of *onActivityCreated* method in the appendix

4.3.5 Application main

The entry point of the application is defined in *MainActivity* class which is a child class of *BaseActivity* class. Application initialization and layout inflating is defined in the following method :

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4
5      binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
6
7      bottomNavigationView = binding.bottomNavigation;
8      floatingActionButton_LiveTranslation = binding.liveTranslationFAB;
9
10     bottomNavigationView.setOnNavigationItemSelectedListener(new BottomNavigationView.
11         OnNavigationItemSelectedListener() {
12         @Override
13         public boolean onNavigationItemSelected(@NonNull MenuItem item) {
14             switch (item.getItemId()) {
15                 case R.id.conversations_item:
16                     if(Navigation.findNavController(MainActivity.this, R.id.host_fragment).
17                         getCurrentDestination() != null){
18                         navigate(0);
19                     }
20                     break;
21                 case R.id.people_item:
22                     if(Navigation.findNavController(MainActivity.this, R.id.host_fragment).
23                         getCurrentDestination() != null){
24                         navigate(1);
25                     }
26                     break;
27                 return true;
28             }
29         });
30
31         MainActivity.floatingActionButton_LiveTranslation.setOnClickListener(new View.
32             OnClickListener() {
33             @Override
34             public void onClick(View v) {
35                 if( Navigation.findNavController(MainActivity.this, R.id.host_fragment).
36                     getCurrentDestination().getId() == R.id.conversations){
37                     Navigation.findNavController(MainActivity.this,R.id.host_fragment).navigate(
38                         ConversationsDirections.actionConversationsToLiveTranslation());
39                 }else if(Navigation.findNavController(MainActivity.this, R.id.host_fragment).
40                     getCurrentDestination().getId() == R.id.poeple)
41                 { Navigation.findNavController(MainActivity.this, R.id.host_fragment).navigate(
42                     PeopleDirections.actionPoepleToLiveTranslation());
43                 }
44
45                 MainActivity.bottomNavigationView.setVisibility(View.GONE);
46                 MainActivity.floatingActionButton_LiveTranslation.setVisibility(View.GONE);
47             }
48         });
49     }
50 }
```

4.4 Actual database schema

5 Testing

5.1 Expected test scenarios

5.2 Unit test

5.3 Functional test

5.4 Usability test

6 Deployment of the system

7 Limitation of the system

8 Conclusion and further work

9 Appendix

9.1 *getMessagesAdapter* method

```
1 public MutableLiveData<DataResource<FirebaseRecyclerPagingAdapter<Message, ChatAdapter.  
MyViewHolder>>> getMessagesAdapter(final String conversationId, final Person friendUser,  
final LifecycleOwner lifecycleOwner) {  
2  
3     DatabaseReference messagesReference = firebaseDatabase.getReference()  
4         .child("conversations")  
5         .child(conversationId)  
6         .child("messages");  
7  
8  
9     PagedList.Config config = new PagedList.Config.Builder()  
10         .setEnablePlaceholders(false)  
11         .setPrefetchDistance(5)  
12         .setPageSize(10)  
13         .build();  
14  
15     DatabasePagingOptions<Message> options = new DatabasePagingOptions.Builder<Message>()  
16         .setLifecycleOwner(lifecycleOwner)  
17         .setQuery(messagesReference, config, Message.class)  
18         .build();  
19  
20  
21     FirebaseRecyclerPagingAdapter<Message, ChatAdapter.MyViewHolder>  
firebaseRecyclerPagingAdapter =  
22         new FirebaseRecyclerPagingAdapter<Message, ChatAdapter.MyViewHolder>(options) {  
23         @Override  
24         protected void onBindViewHolder(@NonNull final ChatAdapter.MyViewHolder myViewHolder  
25         ,  
26                                     int i, @NonNull final Message message) {  
27             myViewHolder.bind(message);  
28  
29             myViewHolder.itemView.findViewById(R.id.message_layout).setOnClickListener(new  
View.OnClickListener() {  
30                 @Override  
31                 public void onClick(View v) {  
if (myViewHolder.itemView.findViewById(R.id.message_options).  
getVisibility() != GONE) {
```

```

32         myViewHolder.itemView.findViewById(R.id.message_options).
setVisibility(GONE);
33     }
34 }
35 }
36 });
37
38     myViewHolder.itemView.findViewById(R.id.message_layout).setOnLongClickListener(
new View.OnLongClickListener() {
39         @Override
40         public boolean onLongClick(View v) {
41             if (myViewHolder.itemView.findViewById(R.id.message_options).
getVisibility() == GONE) {
42                 myViewHolder.itemView.findViewById(R.id.message_options).
setVisibility(View.VISIBLE);
43
44                 if (message.getMessageType().equals("Text") || message.
getMessageType().equals("Welcome")) {
45                     myViewHolder.itemView.findViewById(R.id.download_file_option).
setVisibility(GONE);
46                 } else if (message.getMessageType().equals("Photo")) {
47                     myViewHolder.itemView.findViewById(R.id.translate_to_sign_option
).setVisibility(GONE);
48                     myViewHolder.itemView.findViewById(R.id.
convert_to_voice_message_option).setVisibility(GONE);
49                 } else if (message.getMessageType().equals("File")) {
50                     myViewHolder.itemView.findViewById(R.id.translate_to_sign_option
).setVisibility(GONE);
51                     myViewHolder.itemView.findViewById(R.id.
convert_to_voice_message_option).setVisibility(GONE);
52                 } else if (message.getMessageType().equals("Gif")) {
53                     myViewHolder.itemView.findViewById(R.id.translate_to_sign_option
).setVisibility(GONE);
54                     myViewHolder.itemView.findViewById(R.id.
convert_to_voice_message_option).setVisibility(GONE);
55                 } else if (message.getMessageType().equals("Audio")) {
56                     myViewHolder.itemView.findViewById(R.id.translate_to_sign_option
).setVisibility(GONE);
57                     myViewHolder.itemView.findViewById(R.id.
convert_to_voice_message_option).setVisibility(GONE);
58                 } else if (message.getMessageType().equals("Video")) {
59                     myViewHolder.itemView.findViewById(R.id.translate_to_sign_option
).setVisibility(GONE);
60                     myViewHolder.itemView.findViewById(R.id.
convert_to_voice_message_option).setVisibility(GONE);
61                 }
62
63                 if (!message.getSentBy().getId().equals(firebaseAuth.getCurrentUser
().getUid())) {
64                     myViewHolder.itemView.findViewById(R.id.delete_message_option).
setVisibility(GONE);
65                 } else {
66                     myViewHolder.itemView.findViewById(R.id.delete_message_option).
setVisibility(VISIBLE);
67                 }
68             }
69         }
70     } else {
71         myViewHolder.itemView.findViewById(R.id.message_options).
setVisibility(GONE);
72     }
73     return true;
74 }

```

```

75     });
76     if (!message.getStatus().equals("Seen") && !message.getSentBy().getId().equals(
firebaseAuth.getCurrentUser().getUid())) {
77         DatabaseReference messageSatatusDatabaseReference = firebaseDatabase
78             .getReference()
79             .child("conversations")
80             .child(conversationId)
81             .child("messages")
82             .child(message.getId())
83             .child("status");
84         messageSatatusDatabaseReference.setValue("Seen");
85     }
86
87
88     DatabaseReference usersUnreadMessagesDatabaseReference = firebaseDatabase
89         .getReference()
90         .child("users")
91         .child(firebaseAuth.getCurrentUser().getUid())
92         .child("conversations")
93         .child(friendUser.getId())
94         .child("unreadMessages")
95         .child(firebaseAuth.getCurrentUser().getUid());
96
97     usersUnreadMessagesDatabaseReference.setValue(0);
98
99     LinearLayout send_messageLinearLayout = myViewHolder.itemView.findViewById(R.
id.send_message);
100     LinearLayout recieved_messageLinearLayout = myViewHolder.itemView.findViewById(R.
id.recieved_message);
101     TextView converstaion_time = myViewHolder.itemView.findViewById(R.id.
converstaion_time);
102     TextView converstaion_date = myViewHolder.itemView.findViewById(R.id.
converstaion_date);
103
104
105     if (!message.getId().equals("-1")) {
106         if (!c_date.equals(message.getDateTime().toString().substring(6, 8))) {
107             converstaion_date.setText(message.getDateTime().toString().substring(6,
8) + " - " + message.getDateTime().toString().substring(4, 6));
108             c_date = message.getDateTime().toString().substring(6, 8);
109         } else {
110             converstaion_date.setVisibility(View.GONE);
111         }
112
113         if (!c_time.equals(message.getDateTime().toString().substring(8, 10))) {
114             c_time = message.getDateTime().toString().substring(8, 10);
115
116             converstaion_time.setText(message.getDateTime().toString().substring(8,
10) + " : " + message.getDateTime().toString().substring(10, 12));
117         } else {
118             converstaion_time.setVisibility(View.GONE);
119         }
120     }
121
122
123
124     if (message.getSentBy().getId().equals(firebaseAuth.getCurrentUser().getUid()))
{
125         recieved_messageLinearLayout.setVisibility(View.GONE);
126         send_messageLinearLayout.setVisibility(View.VISIBLE);
127
128         ImageView message_statusImageView = myViewHolder.itemView.findViewById(R.id.
message_statusImageView);
129

```

```

message_status);
130         final MaterialTextView message_textMaterialTextView = myViewHolder.itemView.
            findViewById(R.id.s_message_text);
131         final ImageView message_photoImageView = myViewHolder.itemView.findViewById(
            R.id.s_message_photo);
132         final VideoView message_videoVideoView = myViewHolder.itemView.findViewById(
            R.id.s_message_video);
133         LinearLayout message_audioLinearLayout = myViewHolder.itemView.findViewById(
            R.id.s_message_audio);
134         final ImageView message_audio_playImageView = myViewHolder.itemView.
            findViewById(R.id.s_message_audio_play);
135         final SeekBar message_audio_seekBarSeekBar = myViewHolder.itemView.
            findViewById(R.id.s_message_audio_seekBar);
136         final MaterialTextView message_audio_timeMaterialTextView = myViewHolder.
            itemView.findViewById(R.id.s_message_audio_time);
137         MaterialTextView message_timeMaterialTextView = myViewHolder.itemView.
            findViewById(R.id.s_message_time);
138         LinearLayout file_layout = myViewHolder.itemView.findViewById(R.id.
            s_file_layout);
139         final ImageView downloadFile = myViewHolder.itemView.findViewById(R.id.
            s_downloadFile);
140         TextView fileName = myViewHolder.itemView.findViewById(R.id.s_fileName);
141
142         if (message.getMessageType().equals("Text") || message.getMessageType().
            equals("Welcome")) {
143             message_textMaterialTextView.setText(message.getText());
144         } else if (message.getMessageType().equals("Photo")) {
145             message_textMaterialTextView.setVisibility(GONE);
146             message_photoImageView.setVisibility(VISIBLE);
147             firebaseStorage.getReference().child("images/" + conversationId + "/" +
            message.getDateTime() + message.getText())
148                 .getDownloadUrl()
149                 .addOnSuccessListener(new OnSuccessListener<Uri>() {
150                     @Override
151                     public void onSuccess(Uri uri) {
152                         requestManager.load(uri.toString()).into(
            message_photoImageView);
153                     }
154                 });
155         } else if (message.getMessageType().equals("File")) {
156             message_textMaterialTextView.setVisibility(GONE);
157             file_layout.setVisibility(VISIBLE);
158             fileName.setText(message.getText());
159
160         } else if (message.getMessageType().equals("Video")) {
161             message_videoVideoView.setVisibility(VISIBLE);
162             message_textMaterialTextView.setVisibility(GONE);
163
164             firebaseStorage.getReference().child("videos/" + conversationId + "/" +
            message.getDateTime() + message.getText())
165                 .getDownloadUrl()
166                 .addOnSuccessListener(new OnSuccessListener<Uri>() {
167                     @Override
168                     public void onSuccess(Uri uri) {
169
170                         message_videoVideoView.setVideoURI(uri);
171                         message_textMaterialTextView.setVisibility(VISIBLE);
172                         message_textMaterialTextView.setText("Loading.");
173                         message_videoVideoView.requestFocus();
174                         message_videoVideoView.start();
175
176                         message_videoVideoView.setOnPreparedListener(new
            MediaPlayer.OnPreparedListener() {

```

```

177         @Override
178         public void onPrepared(MediaPlayer mp) {
179             mp.setLooping(true);
180         }
181     });
182
183     message_videoVideoView.setOnInfoListener(new MediaPlayer.
OnInfoListener() {
184
185         @Override
186         public boolean onInfo(MediaPlayer mp, int what, int
extra) {
187
188             MediaController mediaController = null;
189             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.
LOLLIPOP) {
190                 mediaController = new MediaController(
context);
191             }
192             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.
LOLLIPOP) {
193                 message_videoVideoView.setMediaController(
mediaController);
194             }
195             message_videoVideoView.setAnchorView(
mediaController);
196
197             if (what == MediaPlayer.MEDIA_INFO_BUFFERING_END
) {
198                 message_textMaterialTextView.setVisibility(
GONE);
199
200                 return true;
201             } else if (what == MediaPlayer.
MEDIA_INFO_BUFFERING_START) {
202                 message_textMaterialTextView.setVisibility(
VISIBLE);
203                 message_textMaterialTextView.setText("
Loading..");
204                 return true;
205             }
206             return false;
207         }
208     });
209
210     } else if (message.getMessageType().equals("Audio")) {
211         message_textMaterialTextView.setVisibility(GONE);
212         message_audioLinearLayout.setVisibility(VISIBLE);
213
214         firebaseStorage.getReference().child("audios/" + conversationId + "/" +
message.getDateTime() + message.getText())
215             .getDownloadUrl()
216             .addOnSuccessListener(new OnSuccessListener<Uri>() {
217                 @Override
218                 public void onSuccess(Uri uri) {
219                     final MediaPlayer mp = new MediaPlayer();
220                     try {
221                         mp.setDataSource(context, uri);
222                         mp.prepare();
223                     } catch (IOException e) {
224                         e.printStackTrace();
225                     }
226                     message_audio_seekBarSeekBar.setMax(mp.getDuration());

```

```

227
228         final Handler mSeekBarUpdateHandler = new Handler();
229         final Runnable mUpdateSeekBar = new Runnable() {
230             @Override
231             public void run() {
232                 message_audio_seekBarSeekBar.setProgress(mp.
getCurrentPosition());
233                 mSeekBarUpdateHandler.postDelayed(this, 50);
234             }
235         };
236
237         message_audio_timeMaterialTextView.setText(getTimeString
(mp.getDuration()));
238         message_audio_playImageView.setOnClickListener(new View.
OnClickListener() {
239             @Override
240             public void onClick(View v) {
241                 if (mp.isPlaying()) {
242                     mp.pause();
243                     message_audio_seekBarSeekBar.removeCallbacks
(mUpdateSeekBar);
244                     message_audio_playImageView.setImageResource
(R.drawable.ic_play);
245                 } else {
246                     message_audio_playImageView.setImageResource
(R.drawable.ic_pause);
247                     mp.start();
248                     mSeekBarUpdateHandler.postDelayed(
mUpdateSeekBar, 0);
249                     Log.e(TAG, "onClick: " + mp.
getCurrentPosition());
250                     Log.e(TAG, "onClick: " + mp.getDuration());
251                 }
252             }
253         });
254
255         message_audio_seekBarSeekBar.setOnSeekBarChangeListener(
new SeekBar.OnSeekBarChangeListener() {
256             @Override
257             public void onProgressChanged(SeekBar seekBar, int i
, boolean b) {
258                 if (b)
259                     mp.seekTo(i);
260                 Log.e(TAG, "onProgressChanged: " + i);
261             }
262
263             @Override
264             public void onStartTrackingTouch(SeekBar seekBar) {
265             }
266
267             @Override
268             public void onStopTrackingTouch(SeekBar seekBar) {
269             }
270         });
271
272     }
273 }
274
275     }
276 }
277
278     } else if (message.getMessageType().equals("Gif")) {
279

```

```

280         message_textMaterialTextView.setVisibility(GONE);
281         message_photoImageView.setVisibility(VISIBLE);
282         firebaseStorage.getReference().child("gifs/" + conversationId + "/" +
message.getDateTime() + message.getText())
283             .getDownloadUrl()
284             .addOnSuccessListener(new OnSuccessListener<Uri>() {
285                 @Override
286                 public void onSuccess(Uri uri) {
287                     requestManager.asGif().load(uri.toString()).into(
message_photoImageView);
288                 }
289             });
290     }
291
292     if (message.getId().equals("-1")) {
293         conversaion_time.setText(message.getDateTime().toString().substring(8,
10) + " : " + message.getDateTime().toString().substring(10, 12));
294         conversaion_date.setText(message.getDateTime().toString().substring(6,
8) + " - " + message.getDateTime().toString().substring(4, 6));
295         c_date = message.getDateTime().toString().substring(8, 10);
296         c_time = message.getDateTime().toString().substring(4, 6);
297     }
298
299     // Message options
300     myViewHolder.itemView.findViewById(R.id.translate_to_sign_option).
setOnClickListener(new View.OnClickListener() {
301         @Override
302         public void onClick(View v) {
303             if (message.getMessageType().equals("Text") || message.
getMessageType().equals("Welcome")) {
304                 String text = message.getText();
305                 message_photoImageView.setVisibility(VISIBLE);
306
307                 final List<String> words = new ArrayList<>();
308
309                 String word = "";
310
311                 for (int i = 0; i < text.length(); i++) {
312                     if (text.charAt(i) == ' ') {
313                         Log.e(TAG, "onClick: add " + word);
314                         words.add(word);
315                         word = "";
316                     } else {
317                         word += text.charAt(i);
318                     }
319                 }
320
321                 if (!word.equals("")) {
322                     words.add(word);
323                     Log.e(TAG, "onClick: add " + word);
324                 }
325
326                 wordsHandler = new Handler();
327                 wordsRunnable = new Runnable() {
328                     @Override
329                     public void run() {
330                         if (wordsIndex < words.size()) {
331                             firebaseStorage.getReference().child("Signs/" +
words.get(wordsIndex) + ".gif").getDownloadUrl().addOnSuccessListener(new OnSuccessListener<
Uri>() {
332
333                                 @Override
334                                 public void onSuccess(Uri uri) {

```

```

335         requestManager.asGif().load(uri.toString()).
into(message_photoImageView);
336         if (wordsIndex < words.size()) {
337             Log.e(TAG, "onSuccess: loaded a " +
words.get(wordsIndex));
338         }
339     }
340     });
341     wordsIndex++;
342 } else {
343     wordsIndex = 0;
344     wordsHandler = null;
345     wordsRunnable = null;
346     message_photoImageView.setVisibility(GONE);
347     myViewHolder.itemView.findViewById(R.id.
message_options).setVisibility(GONE);
348 }
349
350     if (wordsHandler != null || wordsRunnable != null) {
351         wordsHandler.postDelayed(this, 1000);
352     }
353 }
354 };
355     if (wordsHandler != null || wordsRunnable != null) {
356         wordsHandler.postDelayed(wordsRunnable, 1000);
357     }
358 }
359 }
360 }
361 });
362
363
364     message_timeMaterialTextView.setText(
365         message.getDateTime().toString().substring(8, 10) + " : " + message.
getDateTime().toString().substring(10, 12)
366     );
367
368     t1 = new TextToSpeech(context, new TextToSpeech.OnInitListener() {
369         @Override
370         public void onInit(int status) {
371             if (status != TextToSpeech.ERROR) {
372                 t1.setLanguage(Locale.UK);
373             }
374         }
375     });
376
377     myViewHolder.itemView.findViewById(R.id.convert_to_voice_message_option).
setOnClickListener(new View.OnClickListener() {
378         @Override
379         public void onClick(View v) {
380             t1.speak(message.getText(), TextToSpeech.QUEUE_FLUSH, null);
381         }
382     });
383
384     myViewHolder.itemView.findViewById(R.id.download_file_option).
setOnClickListener(new View.OnClickListener() {
385         @Override
386         public void onClick(View v) {
387             StorageReference ref = null;
388
389             String filePath = "";
390
391             if (message.getMessageType().equals("File")) {

```



```

392         ref = firebaseStorage.getReference().child("files/" +
conversationId + "/" + message.getDateTime() + message.getText());
393         filePath = "Murmuro/Files";
394     } else if (message.getMessageType().equals("Audio")) {
395         ref = firebaseStorage.getReference().child("audios/" +
conversationId + "/" + message.getDateTime() + message.getText());
396         filePath = "Murmuro/Audios";
397     } else if (message.getMessageType().equals("Video")) {
398         ref = firebaseStorage.getReference().child("videos/" +
conversationId + "/" + message.getDateTime() + message.getText());
399         filePath = "Murmuro/Videos";
400     } else if (message.getMessageType().equals("Gif")) {
401         ref = firebaseStorage.getReference().child("gifs/" +
conversationId + "/" + message.getDateTime() + message.getText());
402         filePath = "Murmuro/Gifs";
403     } else if (message.getMessageType().equals("Photo")) {
404         ref = firebaseStorage.getReference().child("images/" +
conversationId + "/" + message.getDateTime() + message.getText());
405         filePath = "Murmuro/Photos";
406     }
407
408     File fileNameOnDevice = null;
409
410
411
412     final File folder = new File(Environment.getExternalStorageDirectory
() +
413         File.separator + filePath);
414     boolean success = true;
415     if (!folder.exists()) {
416         success = folder.mkdirs();
417     }
418     if (success) {
419         // Do something on success
420         fileNameOnDevice = new File(folder + "/" + message.getText());
421     } else {
422         // Do something else on failure
423     }
424
425     Toast.makeText(context, "Start Downloading", Toast.LENGTH_SHORT).
show();
426
427     ref.getFile(fileNameOnDevice).addOnSuccessListener(new
OnSuccessListener<FileDownloadTask.TaskSnapshot>() {
428         @Override
429         public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot
) {
430             Log.e(TAG, "onSuccess: downloaded in " + folder.getName());
431             Toast.makeText(context, "downloaded in " + folder.getName()
+ " folder", Toast.LENGTH_SHORT).show();
432         }
433     });
434 }
435 });
436
437
438 if (message.getStatus().equals("Seen")) {
439     message_statusImageView.setImageResource(R.drawable.ic_online);
440 } else if (message.getStatus().equals("Arrived")) {
441     message_statusImageView.setImageResource(R.drawable.ic_away);
442
443 } else if (message.getStatus().equals("Sended")) {
444     message_statusImageView.setImageResource(R.drawable.ic_busy);

```

```

445     }
446
447
448     } else if (message.getSentBy().getId().equals(friendUser.getId())) {
449         recieved_messageLinearLayout.setVisibility(View.VISIBLE);
450         sended_messageLinearLayout.setVisibility(View.GONE);
451
452         final MaterialTextView message_textMaterialTextView = myViewHolder.itemView.
453         findViewById(R.id.message_text);
454         final ImageView message_photoImageView = myViewHolder.itemView.findViewById(
455         R.id.message_photo);
456         final VideoView message_videoVideoView = myViewHolder.itemView.findViewById(
457         R.id.message_video);
458         LinearLayout message_audioLinearLayout = myViewHolder.itemView.findViewById(
459         R.id.message_audio);
460         final ImageView message_audio_playImageView = myViewHolder.itemView.
461         findViewById(R.id.message_audio_play);
462         final SeekBar message_audio_seekBarSeekBar = myViewHolder.itemView.
463         findViewById(R.id.message_audio_seekBar);
464         final MaterialTextView message_audio_timeMaterialTextView = myViewHolder.
465         itemView.findViewById(R.id.message_audio_time);
466         MaterialTextView message_timeMaterialTextView = myViewHolder.itemView.
467         findViewById(R.id.message_time);
468         LinearLayout file_layout = myViewHolder.itemView.findViewById(R.id.
469         file_layout);
470         ImageView downenloadFile = myViewHolder.itemView.findViewById(R.id.
471         downloadFile);
472         TextView fileName = myViewHolder.itemView.findViewById(R.id.fileName);
473
474         if (message.getMessageType().equals("Text") || message.getMessageType().
475         equals("Welcome")) {
476             message_textMaterialTextView.setText(message.getText());
477         } else if (message.getMessageType().equals("Photo")) {
478             message_textMaterialTextView.setVisibility(GONE);
479             message_photoImageView.setVisibility(VISIBLE);
480             firebaseStorage.getReference().child("images/" + conversationId + "/" +
481             message.getDateTime() + message.getText())
482                 .getDownloadUrl()
483                 .addOnSuccessListener(new OnSuccessListener<Uri>() {
484                     @Override
485                     public void onSuccess(Uri uri) {
486                         requestManager.load(uri.toString()).into(
487                         message_photoImageView);
488                     }
489                 });
490         } else if (message.getMessageType().equals("File")) {
491             message_textMaterialTextView.setVisibility(GONE);
492             file_layout.setVisibility(VISIBLE);
493             fileName.setText(message.getText());
494
495         } else if (message.getMessageType().equals("Video")) {
496             message_videoVideoView.setVisibility(VISIBLE);
497             message_textMaterialTextView.setVisibility(GONE);
498
499             firebaseStorage.getReference().child("videos/" + conversationId + "/" +
500             message.getDateTime() + message.getText())
501                 .getDownloadUrl()
502                 .addOnSuccessListener(new OnSuccessListener<Uri>() {
503                     @Override
504                     public void onSuccess(Uri uri) {
505                         message_videoVideoView.setVideoURI(uri);
506                         message_textMaterialTextView.setVisibility(VISIBLE);
507                     }
508                 });
509         }
510     }

```

```

494         message_textMaterialTextView.setText("Loading.");
495         message_videoVideoView.requestFocus();
496         message_videoVideoView.start();
497
498         message_videoVideoView.setOnPreparedListener(new
MediaPlayer.OnPreparedListener() {
499             @Override
500             public void onPrepared(MediaPlayer mp) {
501                 mp.setLooping(true);
502             }
503         });
504
505         message_videoVideoView.setOnInfoListener(new MediaPlayer.
OnInfoListener() {
506             @Override
507             public boolean onInfo(MediaPlayer mp, int what, int
extra) {
508                 MediaController mediaController = null;
509                 if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.
LOLLIPOP) {
510                     mediaController = new MediaController(
context);
511                 }
512                 if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.
LOLLIPOP) {
513                     message_videoVideoView.setMediaController(
mediaController);
514                 }
515                 mediaController.setAnchorView(
message_videoVideoView);
516
517                 if (what == MediaPlayer.MEDIA_INFO_BUFFERING_END
) {
518                     message_textMaterialTextView.setVisibility(
GONE);
519                     return true;
520                 } else if (what == MediaPlayer.
MEDIA_INFO_BUFFERING_START) {
521                     message_textMaterialTextView.setVisibility(
VISIBLE);
522                     message_textMaterialTextView.setText("
Loading..");
523                     return true;
524                 }
525                 return false;
526             }
527         });
528
529     }
530 }
531
532 } else if (message.getMessageType().equals("Audio")) {
533     message_textMaterialTextView.setVisibility(GONE);
534     message_audioLinearLayout.setVisibility(VISIBLE);
535
536     firebaseStorage.getReference().child("audios/" + conversationId + "/" +
message.getDateTime() + message.getText())
537         .getDownloadUrl()
538         .addOnSuccessListener(new OnSuccessListener<Uri>() {
539             @Override
540             public void onSuccess(Uri uri) {
541                 final MediaPlayer mp = new MediaPlayer();
542                 try {

```

```

543         mp.setDataSource(context, uri);
544         mp.prepare();
545     } catch (IOException e) {
546         e.printStackTrace();
547     }
548     message_audio_seekBarSeekBar.setMax(mp.getDuration());
549
550     final Handler mSeekBarUpdateHandler = new Handler();
551     final Runnable mUpdateSeekBar = new Runnable() {
552         @Override
553         public void run() {
554             message_audio_seekBarSeekBar.setProgress(mp.
getCurrentPosition());
555
556             mSeekBarUpdateHandler.postDelayed(this, 50);
557         }
558     };
559
560     message_audio_timeMaterialTextView.setText(getTimeString
(mp.getDuration()));
561
562     message_audio_playImageView.setOnClickListener(new View.
OnClickListener() {
563
564         @Override
565         public void onClick(View v) {
566             if (mp.isPlaying()) {
567                 mp.pause();
568                 message_audio_seekBarSeekBar.removeCallbacks
(mUpdateSeekBar);
569
570                 message_audio_playImageView.setImageResource
(R.drawable.ic_play);
571
572                 message_audio_playImageView.setImageResource
(R.drawable.ic_pause);
573
574                 mp.start();
575                 mSeekBarUpdateHandler.postDelayed(
mUpdateSeekBar, 0);
576
577                 Log.e(TAG, "onClick: " + mp.
getCurrentPosition());
578
579                 Log.e(TAG, "onClick: " + mp.getDuration());
580             }
581         }
582     });
583
584     message_audio_seekBarSeekBar.setOnSeekBarChangeListener(
new SeekBar.OnSeekBarChangeListener() {
585
586         @Override
587         public void onProgressChanged(SeekBar seekBar, int i
, boolean b) {
588
589             if (b)
590                 mp.seekTo(i);
591             Log.e(TAG, "onProgressChanged: " + i);
592         }
593
594         @Override
595         public void onStartTrackingTouch(SeekBar seekBar) {
596
597         }
598
599         @Override
600         public void onStopTrackingTouch(SeekBar seekBar) {
601
602         }
603     });

```

```

596         }
597     });
598
599
600
601     } else if (message.getMessageType().equals("Gif")) {
602         message_textMaterialTextView.setVisibility(GONE);
603         message_photoImageView.setVisibility(VISIBLE);
604         firebaseStorage.getReference().child("gifs/" + conversationId + "/" +
message.getDateTime() + message.getText())
605             .getDownloadUrl()
606             .addOnSuccessListener(new OnSuccessListener<Uri>() {
607                 @Override
608                 public void onSuccess(Uri uri) {
609                     requestManager.asGif().load(uri.toString()).into(
message_photoImageView);
610                 }
611             });
612     }
613
614
615     if (message.getId().equals("-1")) {
616         conversaion_time.setText(message.getDateTime().toString().substring(8,
10) + " : " + message.getDateTime().toString().substring(10, 12));
617         conversaion_date.setText(message.getDateTime().toString().substring(6,
8) + " - " + message.getDateTime().toString().substring(4, 6));
618         c_date = message.getDateTime().toString().substring(8, 10);
619         c_time = message.getDateTime().toString().substring(4, 6);
620     }
621
622     // Message options
623
624     myViewHolder.itemView.findViewById(R.id.translate_to_sign_option).
setOnClickListener(new View.OnClickListener() {
625         @Override
626         public void onClick(View v) {
627             if (message.getMessageType().equals("Text") || message.
getMessageType().equals("Welcome")) {
628                 String text = message.getText();
629                 message_photoImageView.setVisibility(VISIBLE);
630
631                 final List<String> words = new ArrayList<>();
632
633                 String word = "";
634
635                 for (int i = 0; i < text.length(); i++) {
636                     if (text.charAt(i) == ' ') {
637                         Log.e(TAG, "onClick: add " + word);
638                         words.add(word);
639                         word = "";
640                     } else {
641                         word += text.charAt(i);
642                     }
643                 }
644
645                 if (!word.equals("")) {
646                     words.add(word);
647                     Log.e(TAG, "onClick: add " + word);
648                 }
649
650
651                 wordsHandler = new Handler();
652                 wordsRunnable = new Runnable() {

```

```

653         @Override
654         public void run() {
655             if (wordsIndex < words.size()) {
656                 firebaseStorage.getReference().child("Signs/" +
words.get(wordsIndex) + ".gif").getDownloadUrl().addOnSuccessListener(new OnSuccessListener<
Uri>() {
657                     @Override
658                     public void onSuccess(Uri uri) {
659                         requestManager.asGif().load(uri.toString()).
into(message_photoImageView);
660                         if (wordsIndex < words.size()) {
661                             Log.e(TAG, "onSuccess: loaded a " +
words.get(wordsIndex));
662                         }
663                     }
664                 });
665                 wordsIndex++;
666             } else {
667                 wordsIndex = 0;
668                 wordsHandler = null;
669                 wordsRunnable = null;
670                 message_photoImageView.setVisibility(GONE);
671                 myViewHolder.itemView.findViewById(R.id.
message_options).setVisibility(GONE);
672             }
673             if (wordsHandler != null || wordsRunnable != null) {
674                 wordsHandler.postDelayed(this, 1000);
675             }
676         }
677     };
678     if (wordsHandler != null || wordsRunnable != null) {
679         wordsHandler.postDelayed(wordsRunnable, 1000);
680     }
681 }
682 }
683 }
684 }
685 });
686
687 message_timeMaterialTextView.setText(
688     message.getDateTime().toString().substring(8, 10) + " : " + message.
getDateTime().toString().substring(10, 12)
689 );
690
691
692 t1 = new TextToSpeech(context, new TextToSpeech.OnInitListener() {
693     @Override
694     public void onInit(int status) {
695         if (status != TextToSpeech.ERROR) {
696             t1.setLanguage(Locale.UK);
697         }
698     }
699 });
700
701 t1 = new TextToSpeech(context, new TextToSpeech.OnInitListener() {
702     @Override
703     public void onInit(int status) {
704         if (status != TextToSpeech.ERROR) {
705             t1.setLanguage(Locale.UK);
706         }
707     }
708 });
709

```

```

710         myViewHolder.itemView.findViewById(R.id.convert_to_voice_message_option).
setOnClickListener(new View.OnClickListener() {
711             @Override
712             public void onClick(View v) {
713                 t1.speak(message.getText(), TextToSpeech.QUEUE_FLUSH, null);
714             }
715         });
716
717         message_timeMaterialTextView.setText(
718             message.getDateTime().toString().substring(8, 10) + " : " + message.
getDateTime().toString().substring(10, 12)
719         );
720
721
722         myViewHolder.itemView.findViewById(R.id.download_file_option).
setOnClickListener(new View.OnClickListener() {
723             @Override
724             public void onClick(View v) {
725                 StorageReference ref = null;
726
727                 String filePath = "";
728
729                 if (message.getMessageType().equals("File")) {
730                     ref = firebaseStorage.getReference().child("files/" +
conversationId + "/" + message.getDateTime() + message.getText());
731                     filePath = "Murmuro/Files";
732                 } else if (message.getMessageType().equals("Audio")) {
733                     ref = firebaseStorage.getReference().child("audios/" +
conversationId + "/" + message.getDateTime() + message.getText());
734                     filePath = "Murmuro/Audios";
735                 } else if (message.getMessageType().equals("Video")) {
736                     ref = firebaseStorage.getReference().child("videos/" +
conversationId + "/" + message.getDateTime() + message.getText());
737                     filePath = "Murmuro/Videos";
738                 } else if (message.getMessageType().equals("Gif")) {
739                     ref = firebaseStorage.getReference().child("gifs/" +
conversationId + "/" + message.getDateTime() + message.getText());
740                     filePath = "Murmuro/Gifs";
741                 } else if (message.getMessageType().equals("Photo")) {
742                     ref = firebaseStorage.getReference().child("images/" +
conversationId + "/" + message.getDateTime() + message.getText());
743                     filePath = "Murmuro/Photos";
744                 }
745
746                 File fileNameOnDevice = null;
747
748
749                 final File folder = new File(Environment.getExternalStorageDirectory
750 () +
751                 File.separator + filePath);
752                 boolean success = true;
753                 if (!folder.exists()) {
754                     success = folder.mkdirs();
755                 }
756                 if (success) {
757                     // Do something on success
758                     fileNameOnDevice = new File(folder + "/" + message.getText());
759                 } else {
760                     // Do something else on failure
761                 }
762
763                 Toast.makeText(context, "Start Downloading", Toast.LENGTH_SHORT).

```

```

show();
764
765         ref.getFile(fileNameOnDevice).addOnSuccessListener(new
OnSuccessListener<FileDownloadTask.TaskSnapshot>() {
766             @Override
767             public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot
) {
768                 Log.e(TAG, "onSuccess: downloaded in " + folder.getName());
769                 Toast.makeText(context, "downloaded in " + folder.getName()
+ " folder", Toast.LENGTH_SHORT).show();
770             }
771         });
772     }
773 }
774
775 }
776
777
778     myViewHolder.itemView.findViewById(R.id.delete_message_option).
setOnClickListener(new View.OnClickListener() {
779         @Override
780         public void onClick(View v) {
781             DatabaseReference databaseReference2 = firebaseDatabase.getReference()
782                 .child("conversations")
783                 .child(conversationId)
784                 .child("messages")
785                 .child(message.getId());
786
787             databaseReference2.setValue(null);
788         }
789     });
790
791 }
792
793 @Override
794 protected void onLoadingStateChanged(@NonNull LoadingState state) {
795     switch (state) {
796         case LOADING_INITIAL:
797         case LOADING_MORE:
798             // Do your loading animation
799             messagesAdapterDataResourceMutableLiveData.setValue(DataResource.loading
((FirebaseRecyclerPagingAdapter<Message, ChatAdapter.MyViewHolder>) null));
800             break;
801
802         case LOADED:
803             // Stop Animation
804             messagesAdapterDataResourceMutableLiveData.setValue(DataResource.error("
LOADED", (FirebaseRecyclerPagingAdapter<Message, ChatAdapter.MyViewHolder>) null));
805             break;
806
807         case FINISHED:
808             //Reached end of Data set
809             messagesAdapterDataResourceMutableLiveData.setValue(DataResource.error("
FINISHED", (FirebaseRecyclerPagingAdapter<Message, ChatAdapter.MyViewHolder>) null));
810
811             break;
812
813         case ERROR:
814             retry();
815             break;
816     }
817 }
818

```



```

819         @NonNull
820         @Override
821         public ChatAdapter.MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
822             LayoutInflater inflater = LayoutInflater.from(parent.getContext());
823             ViewDataBinding binding = DataBindingUtil.inflate(inflater, R.layout.
chat_item, parent, false);
824             return new ChatAdapter.MyViewHolder(binding);
825         }
826
827
828         @Override
829         protected void onError(@NonNull DatabaseError databaseError) {
830             super.onError(databaseError);
831             databaseError.toException().printStackTrace();
832         }
833     };
834
835     messagesAdapterDataResourceMutableLiveData.setValue(DataResource.success(
firebaseRecyclerPagingAdapter));
836
837     return messagesAdapterDataResourceMutableLiveData;
838 }

```

9.2 onActivityCreated method

```

1     @Override
2     public void onActivityCreated(@Nullable Bundle savedInstanceState) {
3         super.onActivityCreated(savedInstanceState);
4         mViewModel = ViewModelProviders.of(this, providerFactory).get(LiveTranslationViewModel.
class);
5         if(savedInstanceState != null)
6         {
7             Navigation.findNavController(getActivity(), R.id.host_fragment).restoreState(
savedInstanceState.getBundle("nav_state"));
8             Log.e(TAG, "onRestoreInstanceState: " + savedInstanceState.getBundle("nav_state").
describeContents());
9         }
10
11         binding.camera.setLifecycleOwner(this);
12
13         handler = new Handler();
14         runnable = new Runnable() {
15             @Override
16             public void run() {
17                 // binding.camera.setFilter(Filters.GRAYSCALE.newInstance());
18                 binding.camera.setAudio(Audio.OFF);
19                 binding.camera.takePicture();
20                 if(handler != null && runnable != null)
21                 {
22                     handler.postDelayed(this, 2000);
23                 }
24             }
25         };
26         if(handler != null && runnable != null)
27         {
28             handler.postDelayed(runnable, 2000);
29         }
30
31
32         binding.camera.close();
33
34
35         binding.camera.addCameraListener(new CameraListener() {

```

```

36         @Override
37         public void onPictureTaken(final PictureResult result) {
38             result.toBitmap(40, 40, new BitmapCallback() {
39                 @SuppressWarnings("WrongThread")
40                 @Override
41                 public void onBitmapReady(@Nullable Bitmap bitmap) {
42
43                     bitmap = Bitmap.createScaledBitmap(bitmap, INPUT_SIZE, INPUT_SIZE, false
44 );
45
46                     final List<Classifier.Recognition> results = classifier.recognizeImage(
47 bitmap);
48
49                     Log.d(TAG, "oraby onBitmapReady: " + results.toString());
50                     String message = "";
51
52                     for(int i=0; i<results.size();i++)
53                     {
54                         message += results.get(i).getTitle() + " ";
55                     }
56                     binding.translatedText.setText(binding.translatedText.getText() +
57 message);
58
59                 }
60             });
61
62             initTensorFlowAndLoadModel();
63
64
65             binding.toolbar.setNavigationOnClickListener(new View.OnClickListener() {
66                 @Override
67                 public void onClick(View v) {
68                     Navigation.findNavController(getActivity(), R.id.host_fragment).popBackStack();
69                 }
70             });
71
72
73             binding.avatarButton.setOnClickListener(new View.OnClickListener() {
74                 @Override
75                 public void onClick(View v) {
76                     binding.camera.setVisibility(View.GONE);
77                     binding.camera.close();
78                     handler = null;
79                     runnable = null;
80
81                     binding.avatarLayout.setVisibility(View.VISIBLE);
82                     binding.editTextLayout.setVisibility(View.VISIBLE);
83                     binding.signTranslationButton.setVisibility(View.VISIBLE);
84                     binding.translatedText.setVisibility(View.GONE);
85                     binding.avatarButton.setVisibility(View.GONE);
86
87                 }
88             });
89
90             binding.messageEditText.addTextChangedListener(new TextWatcher() {
91                 @Override
92                 public void beforeTextChanged(CharSequence s, int start, int count, int after) {
93
94                 }
95

```

```

96         @Override
97         public void onTextChanged(CharSequence s, int start, int before, int count) {
98
99
100     }
101
102     @Override
103     public void afterTextChanged(Editable s) {
104         if(s.toString().equals(""))
105         {
106             binding.sendImage.setImageResource(R.drawable.ic_microphone);
107         }else
108         {
109             binding.sendImage.setImageResource(R.drawable.ic_send);
110         }
111     }
112 });
113
114 binding.sendImage.setOnClickListener(new View.OnClickListener() {
115     @Override
116     public void onClick(View v) {
117         if(!binding.messageEditText.getText().toString().trim().equals(""))
118         {
119             String text = binding.messageEditText.getText().toString().trim();
120
121             final List<String> words = new ArrayList<>();
122
123             String word = "";
124
125             for(int i=0; i < text.length();i++)
126             {
127                 if(text.charAt(i) == ' ')
128                 {
129                     Log.e(TAG, "onClick: add " + word );
130                     words.add(word);
131                     word = "";
132                 }else
133                 {
134                     word += text.charAt(i);
135                 }
136             }
137
138             if(!word.equals(""))
139             {
140                 words.add(word);
141                 Log.e(TAG, "onClick: add " + word );
142             }
143
144
145             wordsHandler = new Handler();
146             wordsRunnable = new Runnable() {
147                 @Override
148                 public void run() {
149                     if(wordsIndex < words.size())
150                     {
151                         firebaseStorage.getReference().child("Signs/" + words.get(
wordsIndex) + ".gif").getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
152                             @Override
153                             public void onSuccess(Uri uri) {
154                                 requestManager.asGif().load(uri.toString()).into(binding.
avatarImageView);
155
156                                 if(wordsIndex < words.size())
157                                 {

```

```

157         Log.e(TAG, "onSuccess: loaded a " + words.get(
wordsIndex) );
158     }
159 }
160 });
161     wordsIndex++;
162 }else
163 {
164     wordsIndex = 0;
165     wordsHandler = null;
166     wordsRunnable = null;
167 }
168
169     if(wordsHandler != null || wordsRunnable != null)
170     {
171         wordsHandler.postDelayed(this, 1000);
172     }
173 }
174 };
175 if(wordsHandler != null || wordsRunnable != null)
176 {
177     wordsHandler.postDelayed(wordsRunnable, 1000);
178 }
179
180 }else
181 {
182     startVoiceInput();
183 }
184 }
185 });
186
187 binding.signTranslationButton.setOnClickListener(new View.OnClickListener() {
188     @Override
189     public void onClick(View v) {
190
191         binding.avatarLayout.setVisibility(View.GONE);
192         binding.editTextLayout.setVisibility(View.GONE);
193         binding.signTranslationButton.setVisibility(View.GONE);
194         binding.translatedText.setVisibility(View.VISIBLE);
195         binding.avatarButton.setVisibility(View.VISIBLE);
196
197
198
199         binding.camera.open();
200         binding.camera.setVisibility(View.VISIBLE);
201         binding.camera.setLifecycleOwner(LiveTranslation.this);
202
203         handler = new Handler();
204         runnable = new Runnable() {
205             @Override
206             public void run() {
207                 // binding.camera.setFilter(Filters.GRAYSCALE.newInstance());
208                 binding.camera.setAudio(Audio.OFF);
209                 binding.camera.takePicture();
210                 if(handler!= null && runnable != null) {
211                     handler.postDelayed(this, 2000);
212                 }
213             }
214         };
215         if(handler!= null && runnable != null) {
216             handler.postDelayed(runnable, 2000);
217         }
218

```

```

219         binding.camera.addCameraListener(new CameraListener() {
220             @Override
221             public void onPictureTaken(final PictureResult result) {
222                 result.toBitmap(40, 40, new BitmapCallback() {
223                     @SuppressWarnings("WrongThread")
224                     @Override
225                     public void onBitmapReady(@Nullable Bitmap bitmap) {
226
227                         bitmap = Bitmap.createScaledBitmap(bitmap, INPUT_SIZE,
228 INPUT_SIZE, false);
229                         if(bitmap != null)
230                         {
231                             final List<Classifier.Recognition> results = classifier.
232 recognizeImage(bitmap);
233
234                             Log.d(TAG, "oraby onBitmapReady: " + results.toString());
235
236                             String message = "";
237
238                             for(int i=0; i<results.size();i++)
239                             {
240                                 message += results.get(i).getTitle() + " ";
241                             }
242
243                             binding.messageEditText.setText(binding.messageEditText.
244 getText() + message);
245                         }
246                     }
247                 });
248             }
249         });
250
251         initTensorFlowAndLoadModel();
252     }
253 }
254 });
255
256
257 }

```

Références

- [1] B Dorner. « Hand shape identification and tracking for sign language interpretation ». In : *IJCAI'93 Looking at people Workshop*. 1993.
- [2] Jason Andre Gilbert et Shau-yuh YU. *Sign language translation system*. US Patent App. 12/167978. Jan. 2009.
- [3] Sami M Halawani. « Arabic sign language translation system on mobile devices ». In : *IJCSNS International Journal of Computer Science and Network Security* 8.1 (2008), p. 251-256.
- [4] Tomoichi Takahashi et Fumio Kishino. « Hand gesture coding based on experiments using a hand gesture interface device ». In : *Acm Sigchi Bulletin* 23.2 (1991), p. 67-74.