Bilkent University

Department of Computer Engineering

Senior Design Project

Final Report

Project: Signify

Team Members: Ali Taha Dinçer, Çağlar Çankaya, İrem Ecem Yelkanat, Muhammed Naci Dalkıran, Sena Korkut

Supervisor: Ayşegül Dündar

Jury Members: Shervin Arashloo & Hamdi Dibeklioglu

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

1. Introduction

In society, people who are hearing impaired and/or speech impaired have difficulty expressing themselves and communicating with other people because most people lack knowledge of sign language. Even though the improvements in technology have changed the way people live and made the lives of people easier by, for example, transforming mobile phones from sound devices into multi-functional devices, communicating with people who are hearing impaired and/or speech impaired continues to be a problem in many areas including social and technical contexts. These communication activities include social life, healthcare, career development, and education. Furthermore, as a result of the Covid19 pandemic that started in late 2019, obligations regulated by most countries, including wearing face masks and social and physical distancing, have increased the communication and social challenges for hearing impaired people. For example, wearing face masks has led to some negative impacts in communication with other people as it eliminates speech perception by visual features through lipreading. Additionally, a considerable amount of face-to-face communications have turned into virtual communications, which results in more hardship for the hearing impaired and/or speech impaired people as some of the most used virtual communication services like Zoom, Microsoft Teams or Skype do not support sign language.

According to the World Health Organization (WHO), 5% of the people on the earth are hearing impaired, which is more than 350 million people [1] and will exceed 700 million by 2050 [2]. Considering that, sustainability of the social lives of hearing-impaired and/or speech impaired people will be an essential issue in the future. Therefore, we propose a solution to this problem named Signify. Signify is a mobile application with the main aim of helping hearing and/or speech impaired people in their social lives by translating sign language into text along with speech and text-speech to sign language translation in real-time.

This report consists of the final requirements details, architecture and design, development and implementation details, testing details, maintenance plan details, and other project elements.

2. Requirements Details

2.1. Overview

Signify is a mobile application that aims to solve social problems regarding communication, understanding, and expression for the hearing impaired and/or speech impaired people. By

this means, Signify helps these people to improve their quality of life. In that manner, the application can be seen as a communication tool.

There are already existing applications such as “Hand Talk Tradutor para Libras” [3], “ASL Translator” [4], and “S.L.A.I.T.” [5] for the hearing impaired and/or speech impaired people. However, “Hand Talk Tradutor para Libras” does not contain real-time translation over the video, which creates a one-way channel between communicators. “ASL Translator” is a paid application and works on pre-given text and video translations, in which translations are pretty limited, and for every new word or phrase, the app needs to be updated. Finally, “S.L.A.I.T.” can only be used during video calls, and it is still in the beta phase.

Signify combines these apps in terms of their translational and conversational capabilities with tools like bidirectional translation. With Signify, users can easily communicate through video calls and use their mobile phones to real-time translate to both text and sign language in a day-to-day conversation. With this, hearing impaired and/or speech impaired people can join conversations even if the people in the conversation do not know sign language.

2.2. Functional Requirements

2.2.1. Bidirectional Language Translation

The main purpose of the application is to provide a channel among hearing-impaired or speech-impaired people and society. For this intention, two-way language transformation is ensured. The application is able to translate an English speech or text into sign language. On the reverse side, the application also converts sign language from video to text for the users who do not know sign language. For this, the user should be able to use the microphone and camera of their phones.

2.2.2. Real-time Language Translation

The application works close to the real-time speed to achieve practical and effective communication, especially for online meetings and conferences. Users should be able to reach the translation almost synchronically with the initial communicator.

2.2.3. Additional Information and Tutorial

The application provides extra information about ASL. The main page contains two different sections that shows the recent news about ASL and a tutorial for learning word-level ASL.

2.3. Non-functional Requirements

2.3.1. Usability

The application has a user-friendly interface for users to navigate and appreciate any feature within two seconds. Also, because the target user is impaired people, universal icons and modern designs will be used. In this way, the application gains higher usability.

2.3.2. Extensibility

The application architecture is easily extensible for new learning models and additional features.

2.3.3. Performance

Machine learning related computation is computed and results are sent from a host, therefore, the response time is expected to be 100 ms.

2.3.4. Security

The application does not leak any non-encrypted personal information in case of a data breach.

2.3.5. Compatibility

The application is available for all operating systems, including Android and iOS, to deliver the service to all users.

2.3.6. Privacy

Any personal information and video and audio records are not used or shared with any user, a third-party company, or application.

2.4. Pseudo Requirements

- Version Control Git/Github is used as a version controller throughout the project.
- In order to implement the project, the Dart programming language is used.
- The application is available on both Android and iOS platforms.
- Firebase is used for the backend side of the project and Flutter will be used for the frontend side of the project.

- PyTorch is used in order to implement the GAN, OpenPose, and other prediction models.
- StanfordNLP and HappyTextToText libraries are used to convert text to gloss and gloss to text.
- WLASL Dataset is used in order to train the models.
- In order to implement online communication, the stream WebRTC library is used.

3. Final Architecture and Design Details

3.1. Overview

Signify mainly works through video, text, and audio inputs followed by generated video and text outputs. It is a phone-based application adaptable to different operating systems. Signify is based on a simple client-server architecture. The application uses Firebase in order to store user-related data and room data that have been generated by creating calls. All the requests by the user require changes in real-time, therefore Firebase Real-Time Database is used. The application requires all the translations handled on the localhost, so, API systems are optimized to send and receive requests near real-time. Each subsystem in the implementation respects the 4 layer MVC architecture. The detailed software architecture and more are shown and discussed in the next sections.

3.2. Subsystem Decomposition

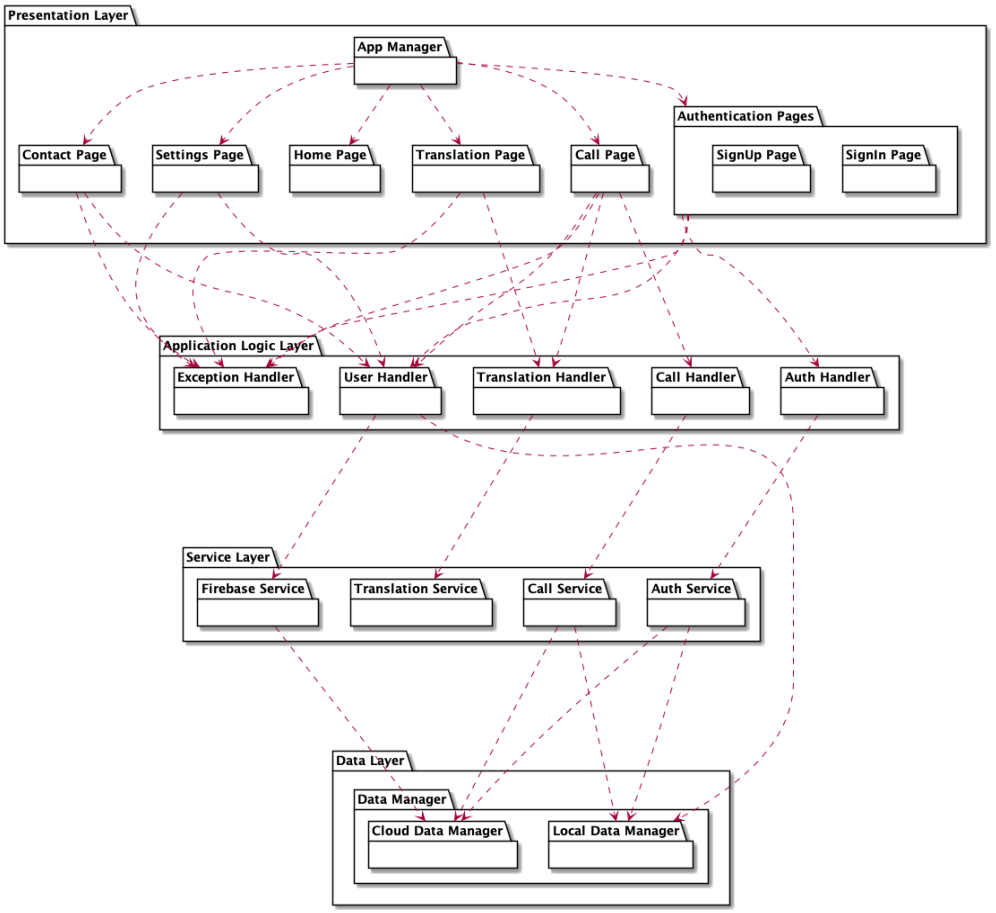

Figure 1: *Subsystem Decomposition Diagram*

3.3. Hardware/software Mapping

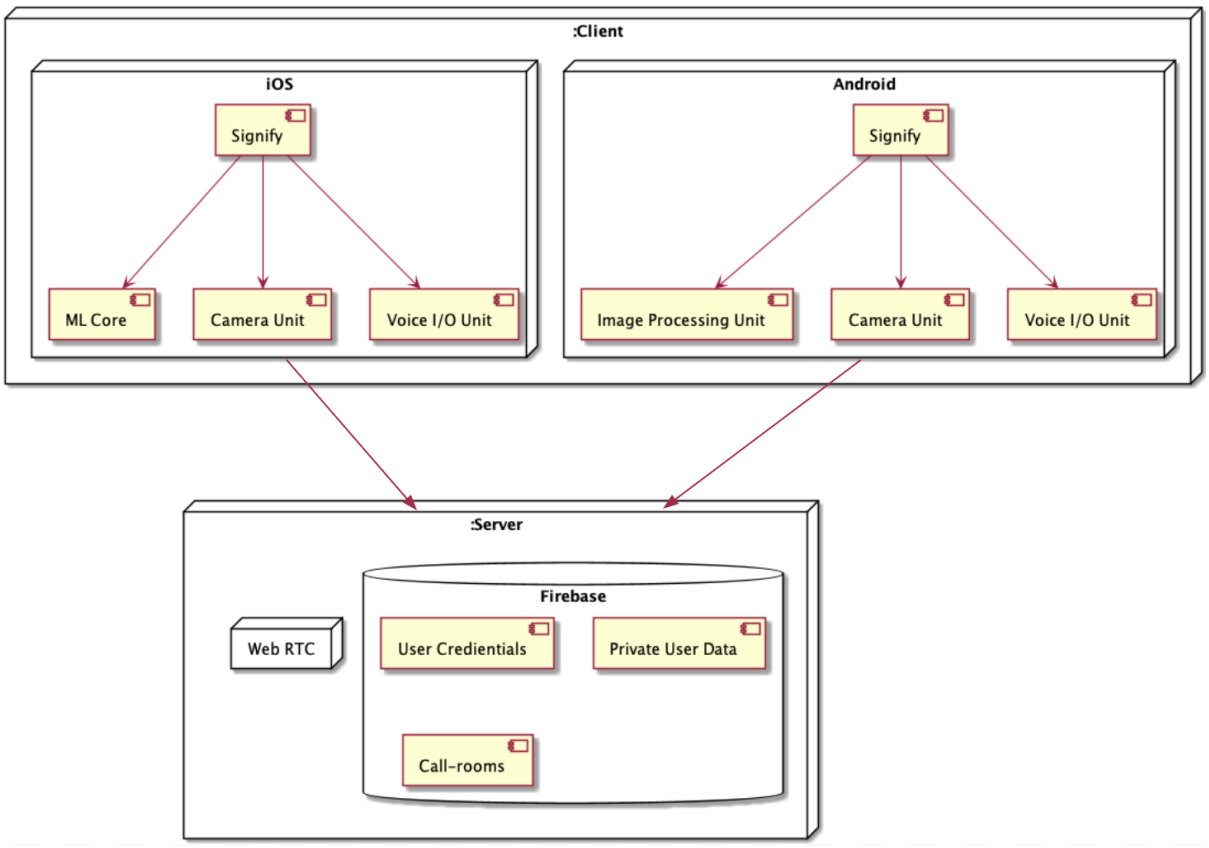

Figure 2: *Deployment Diagram*

The architecture of hardware/software mapping is shown in Figure 2. Signify architecture consists of two main parts: Client and Server. The server holds Firebase, which is the database that will be used to store the information of user credentials, private user data, and call-rooms for video conferencing. Additionally, Web RTC is used for real-time communication during video conferences. It is connected to the server along with Firebase and each process related to video calls is operated based on the information stored on Firebase (i.e. call rooms). Signify stores its machine learning models in localhost and handles the translation using FastAPI services. Every other operation is mapped on the Client side. Translation will be done by processing keyboard input, audio, and video that are input by the users; thus, the application's main features require the usage of hardware systems such as a camera unit and voice I/O unit for getting input data and generating a translated output. Signify uses those units on the Client-side through an operating system. In addition, Signify is adaptable to both iOS and Android.

3.4. Persistent Data Management

User-related data which are the user's name, password, and id are stored in the Firebase database to use during signing in, joining meetings, inviting meetings, etc. The users can have friends and they can invite them; therefore, user's friends data such as their ids, contact information for meeting, names etc. are also stored in the Firebase database. However, the meetings' history, logs, and outputs generated by ML models are not stored in the database because these are personal information, and storing them might cause privacy violations. Meeting-related data mentioned in the previous sentence is stored in the local memory of the user.

3.5. Access Control and Security

Users have an account in Signify. They register the application with their name, password, and email. These data are stored in the database. In the database, users have unique userIds generated automatically by the database. When the user signs in to the application, they should enter her/his password and email. To control the sign-in, email and password combination should be checked. The passwords of the users are encrypted in the database with an authenticator key and hash map to avoid adversarial attacks and account thefts. Even if the adversarials access the database server, they cannot access passwords without an authentication key. The length of the authentication key is n which cannot be broken in real-time (polynomial-time boundary). For the real-time meeting, the microphone for users having no disability and video permission should be taken from the user. In the meeting, users can share personal information with other users; therefore, the chat, logs, and personal data cannot be stored. As a database, we use Firebase. Firebase has its own defense system against adversarial attacks. It can be said to be trusted third-party software. ML models process the meeting data like sent and received messages, voice, and video by sending them through APIs. After a while, the data sent is deleted from the server. Finally, during the meeting, user communication is provided via Web-RTC API. This API is a trusted API and has its own defense protocol against adversaries.

3.6. Global Software Control

In the application, the same data might be requested for meeting the users' needs. For instance, a huge number of users want to add their friends. In this case, the database and application might encounter race conditions. To handle this issue, the server is event-driven.

Event-driven provides the server with opportunities to respond to client's request quickly and accurately. Event-driven programming is generally used in communication between server and client.

3.7. Boundary conditions

- **Initialization**

The application is a Flutter project that is shipped via downloading from App Store for iOS devices and Google Play Store for Android devices. After downloading and installing the application to the phone, the application can start by clicking the application.

When the user opens the application for the first time, the user is navigated to the sign-in page where they can log in to the application or create a new account. After the authentication process is successfully completed and the user logs in to the application, they can navigate to the main functionality screens including the home page, translation page, video call page, contacts page, and profile page.

The user must have a proper Internet connection in order to use login/register functionalities, change profile-related data, manage the contacts and participate in video calls because these operations need server connection establishment.

- **Termination**

The users can log out from the application by clicking on the logout button. When the user logs out from the application, all the data that is stored locally on the user's mobile device will be deleted permanently. When the user closes the application without logging out, when they open the application again, they will be logged in to the application automatically.

- **Failure**

While using the application, it is possible to face some errors, but it is significant to eliminate these problems before the users encounter them. In the implementation of Signify, we provided users with an extremely reliable system by utilizing exception handling mechanisms as efficiently as possible. However, the users still could encounter some failures that can happen due to the quality of Internet connection, mobile device's hardware systems, and server errors. When the user faces a network error, the system disconnects the user from the video call if the network error could not be solved in 1 minute. In case of server errors, the

applications cannot connect to the database which may result in some failures in the application such as not being able to log in or not being able to create/join video calls.

4. Development/Implementation Details

4.1. Object Design Trade-offs

4.1.1. Security vs. Portability

Computation for machine learning models requires high computational power. Using third-party computational power (computation in the cloud/server) leads to a security concern since the user's audio, and the video should be sent to the cloud/server. For now, we use our local APIs, therefore, there is no security concern. However, if we move the translation operations to a third-party server in the future, we should handle this security concern.

4.1.2. Rapid Development vs. Functionality

This project is a one-year project; therefore, we developed the application rapidly. During the project implementation, we aimed to implement our core functionalities as soon as possible. However, because of the short time we have, some of the planned functionalities might be ignored. For example, for sign language generation, the model can generate sign language using the users' image; but it remained an extra option as its performance is not enough to be published yet.

4.1.3. Efficiency vs. Portability

The application can be used efficiently on the targeted devices which provide high computational power. But if the device cannot provide the required computational power, the efficiency of the application might be decreased. For example, some computations might take a long time, and the accuracy of the ML models might be decreased. Because we want that all users most efficiently use the application, we restricted the target devices.

4.2. Interface Documentation Guidelines

We used the following convention for class descriptions.

Class Name	Explanation for the class	
Attributes	attributeName: type	Explanation for the attribute
Methods	methodName(args): return type	Explanation for the method

“Class Name” is the name of the class described, “Attributes” are listed as names and types followed by their explanations, “Methods” are listed as signatures and return types followed by their explanations.

4.3. Engineering Standards

This report follows the Unified Modeling Language (UML) [3] standards to visualize the design of the system and represent class interfaces. Additionally, IEEE referencing style standards [4] for citations are used throughout the report for all of the citations.

4.4. Packages

4.4.1. Screens

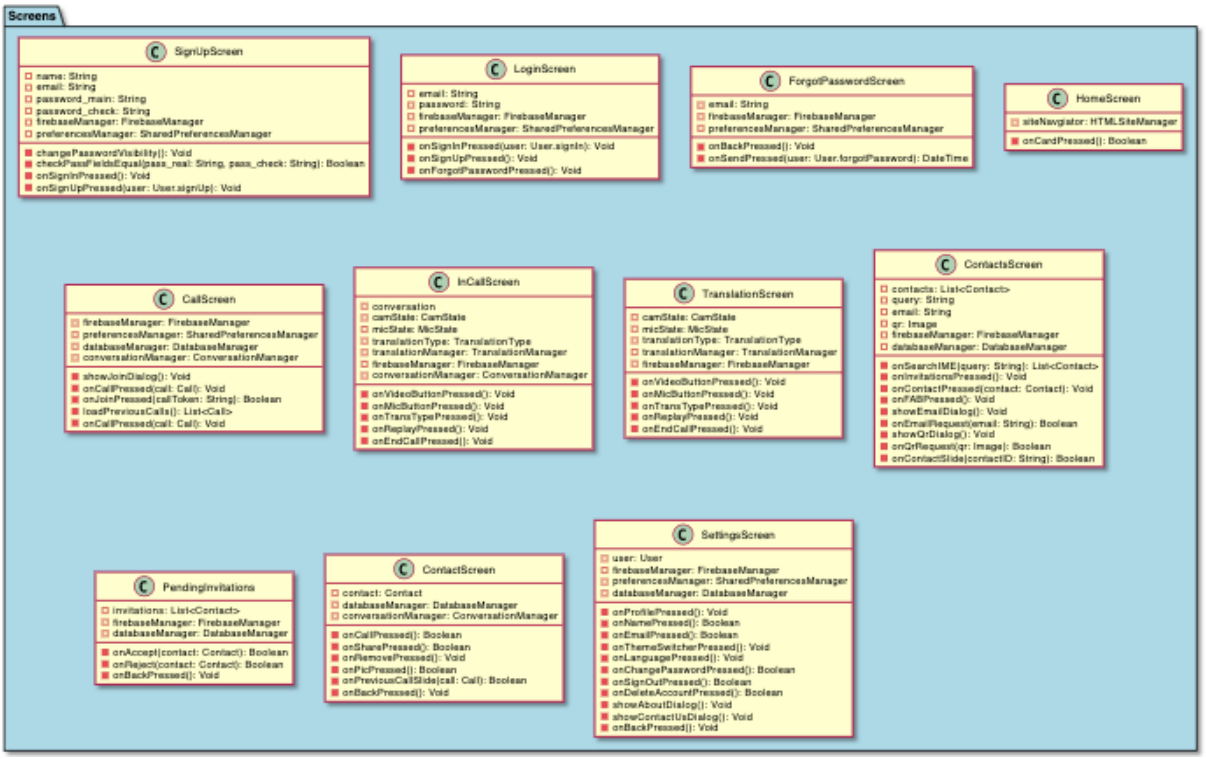

Figure 3: *Screens Package*

SignupScreen: Displays the signup screen before entering the application. The app falls back to this screen if the user has not signed up or logged in.

LoginScreen: Displays the login screen for the users who already have an account. The app falls back to this screen if the user logged off from the app.

ForgotPasswordScreen: Displays the screen to receive mail regarding resetting the account if the user already has an account.

HomeScreen: The main screen after logging or signing in to the application. This screen welcomes the user unless the user decides to log out.

CallScreen: Displays the call screen with call requests and history.

InCallScreen: Displays the call screen in the online conference joined or created.

TranslationScreen: Displays the translation screen with camera and microphone options for the translation in real-time.

ContactsScreen: Displays the contacts of the user.

PendingInvitations: Displays pending contact invitations.

ContactScreen: Displays a specific contact with the profile information and call history.

SettingsScreen: Displays the settings for the users to customize the interface, or change the profile information according to their preferences.

4.4.2. Managers

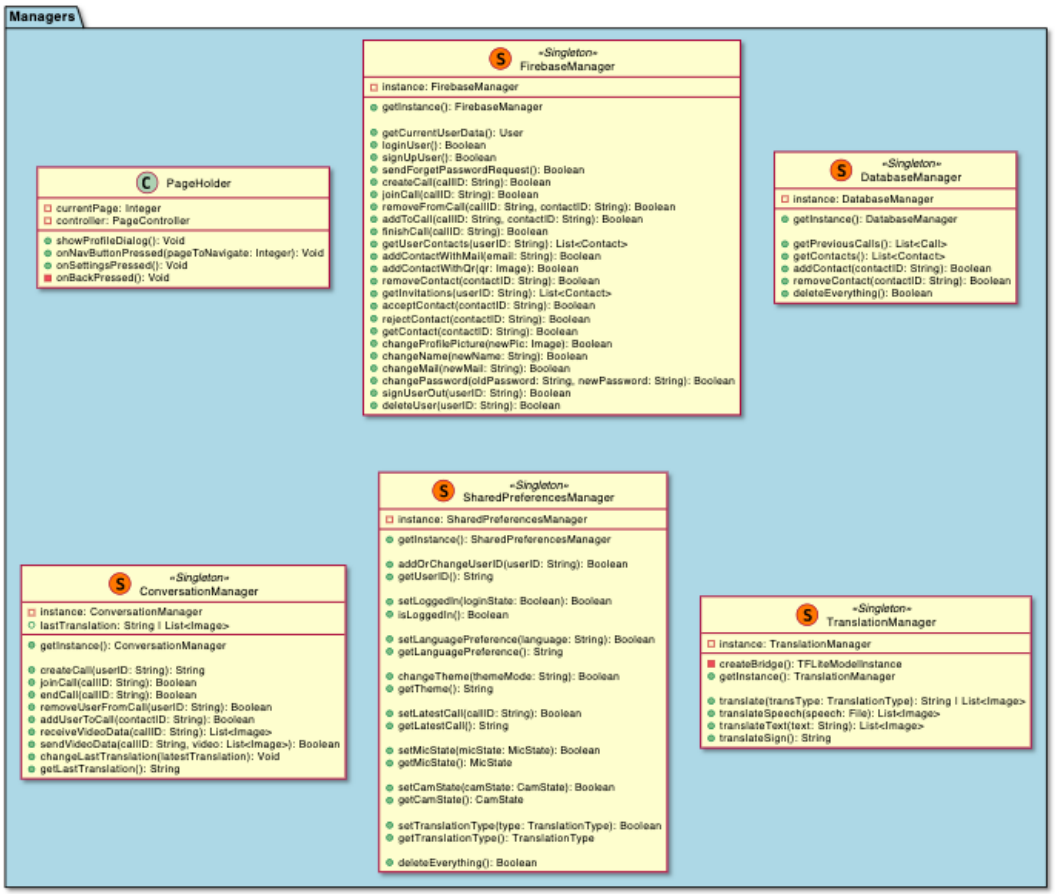

Figure 4: *Managers Package*

PageHolder: Main manager to handle screen translation throughout the app. Basically, a super-class that contains each screen and handles the navigation.

FirebaseManager: Updates Firebase and retrieves information according to the user.

DatabaseManager: Updates the database and retrieves information about the contacts according to the user.

ConversationManager: Manages the requirements of the online conference.

SharedPreferencesManager: Manages the interface settings according to user preferences.

TranslationManager: Manages the translation among text, speech and video. Work as a bridge to main machine learning mechanics.

4.4.3. Model

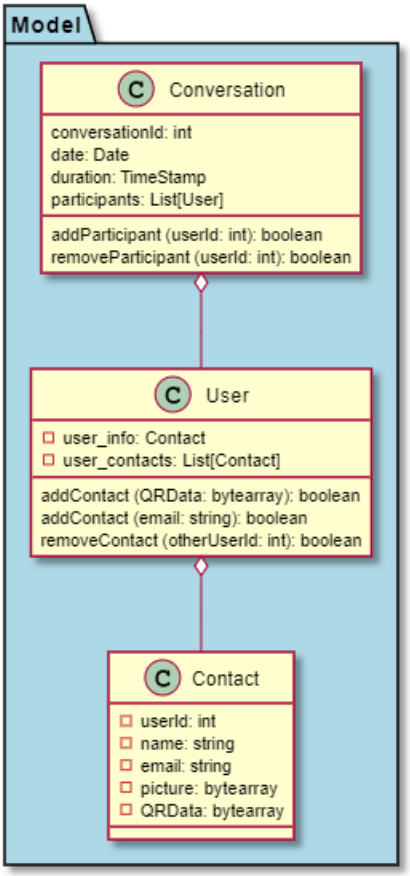

Figure 5: *Model Package*

Conversation: Holds the information about the online conference.

User: Holds the information about the user.

Contact: Holds the information about a contact of the user.

4.5. Class Interfaces

4.5.1. Screens

SignUpScreen	Displays the sign up screen before entering the application.	
Attributes	name: String	Name input by the user.
	email: String	E-mail input by the user.
	password_main: String password_check: String	Password and password check input by the user.
	firebaseManager:FirebaseManager	Instance of a Firebase Manager class.
	preferencesManager:SharedPreferencesManager	Instance of a Shared Preferences Manager class.
Methods	changePasswordVisibility(): void	The option to make the password visible or invisible during user input.
	checkPasswordEquals(): boolean	Checks if the password main and password check matches.
	onSignInPressed():void	Calls Firebase instance and SharedPreferences instance and invokes sign-in operations including saving user token to preferences instance.
	onSignUpPressed(user: User.signUp): void	Navigates to LoginScreen without losing the already entered user info by using the User<SignUpType> constructor.

LogInScreen	Displays the login screen for the users who already have an account.	
Attributes	email: String	E-mail input by the user.
	password: String	Password input by the user.

	firebaseManager:FirebaseManager	Instance of a Firebase Manager class.
	preferencesManager:SharedPreferencesManager	Instance of a Shared Preferences Manager class.
Methods	onSignInPressed(user: User.signIn): void	Navigates to SignInScreen without losing the already entered user info by using the User<SignInType> constructor.
	onSignUpPressed(): void	Calls Firebase instance and SharedPreferences instance and invokes sign-up operations including saving already existing user token to preferences instance.
	onForgotPasswordPressed(): void	Navigates to ForgetPasswordScreen.

ForgotPasswordScreen	Displays the screen to receive mail regarding resetting the account if the user already has an account.	
Attributes	email: String	E-mail input by the user.
	firebaseManager:FirebaseManager	Instance of a Firebase Manager class.
	preferencesManager:SharedPreferencesManager	Instance of a Shared Preferences Manager class.
Methods	onBackPressed(): void	Goes back to the previous page when it is called.
	onSendPressed(user:User.forgotPassword): DateTime	Sends e-mail to the user when it is called. Returns a DateTime instance of the epoch time that the request sent in order to calculate the time to show the “resend” option.

HomeScreen	The main screen after logging in to the application.	
Attributes	siteNavigator: HTMLSiteManager	Site navigator attribute.

Methods	onCardPressed():boolean	Open the URL by using the HTMLSiteManager instance with respect to the clicked card content.

CallScreen	Displays the call screen with call requests and history.	
Attributes	firebaseManager: FirebaseManager	Instance of a Firebase Manager class.
	preferencesManager: SharedPreferencesManager	Instance of a Shared Preferences Manager class.
	databaseManager: DatabaseManager	Instance of a Database Manager class.
	conversationManager: ConversationManager	Instance of a Conversation Manager class.
Methods	showJoinDialog():void	Shows the joining dialog when it is called.
	onCallPressed(call:Call):void	Creates a call when it is called by using a Call instance containing the users' ID and created call instance ID.
	onJoinPressed(callToken:String): boolean	Joins the user to an ongoing call when it is called by using the callToken. Returns true if joining is successful, false otherwise.
	loadPreviousCalls():List<Call>	Retrieves the list of previous calls of the user.

InCallScreen	Displays the call screen in the online conference joined or created.	
Attributes	camState:CamState	State of the camera: CamState.ON: Camera is On CamState.OFF: Camera is Off
	micState: MicState	State of the microphone: MicState.ON: Microphone is

		On MicState.OFF: Microphone is Off
	translationType: TranslationType	Type of the translation: TranslationType.SETT : speech to text TranslationType.SITT : sign to text TranslationType.TTS: text to sign
	translationManager:TranslationManager	Instance of Translation Manager class.
	firebaseManager:FirebaseManager	Instance of Firebase Manager class.
	conversationManager:ConversationManager	Instance of Conversation Manager class.
Methods	onVideoButtonPressed():void	Changes the state of the camera to CamState.ON if the camera is off and vice-versa.
	onMicButtonPressed():void	Changes the state of the microphone to MicState.ON if the microphone is off and vice-versa.
	onTransTypePressed():void	Changes the state of the translation type to other types sequentially.
	onReplayPressed():void	Replays the last translation when it is called.
	onEndCallPressed():void	Ends the call when it is called. Will invoke Firebase and Conversation Manager instances to handle the finishing operations.

TranslationScreen	Displays the translation screen with camera and microphone options for the translation in real life.	
Attributes	camState:CamState	State of the camera: CamState.ON: Camera is On CamState.OFF: Camera is Off

	micState: MicState	State of the microphone: MicState.ON: Microphone is On MicState.OFF: Microphone is Off
	translationType: TranslationType	Type of the translation: TranslationType.SETT : speech to text TranslationType.SITT : sign to text TranslationType.TTS: text to sign
	translationManager:Translation Manager	Instance of Translation Manager class.
	firebaseManager:FirebaseManag er	Instance of Firebase Manager class.
Methods	onVideoButtonPressed():void	Changes the state of the camera to CamState.ON if the camera is off and vice-versa.
	onMicButtonPressed():void	Changes the state of the microphone to MicState.ON if the microphone is off and vice-versa.
	onTransTypePressed():void	Changes the state of the translation type to other types sequentially.
	onReplayPressed():void	Replays the last translation when it is called.

ContactsScreen	Displays the contacts of the user.	
Attributes	contacts:List<Contact>	Contact list of the user.
	query:String	Search query input string.
	email:String	E-mail of the user.
	qr:Image	Qr code of the user.
	firebaseManager:FirebaseManag er	Instance of Firebase Manager class.
	databaseManager:DatabaseMana	Instance of Database Manager

	ger	class.
Methods	onSearchIME(query: String): List<Contact>	Search a contact from the list according to the given query.
	onFABPressed():void	Opens the modal sheet containing options to add a new contact.
	onContactPressed(contact:Contact):void	Goes to the contact screen for the specific contact chosen.
	showEmailDialog():void	Opens a dialog in order to get the user input containing email of the contact to add.
	onEmailRequest(email:String):boolean	Sends an invitation request to the new added contact through Firebase channels by using the given email.
	showQrDialog():void	Opens a dialog in order to get the user input containing Qr of the contact to add..
	onQrRequest(qr:Image):boolean	Sends an invitation request to the new added contact through Firebase channels by using the given Qr.
	onContactSlide(contactID:String):boolean	Delets the slided contact instance from the users' contacts.

PendingInvitations	Displays pending contact invitations.	
Attributes	invitations:List<Contact>	List of the contact requests.
	firebaseManager:FirebaseManager	Instance of a Firebase Manager class.
	databaseManager:DatabaseManager	Instance of a Database Manager class.
Methods	onAccept(contact:Contact):boolean	Accepts the contact request when it is called.
	onReject(contact:Contact):boole	Rejects the contact request

	an	when it is called.
	onBackPressed():void	Goes back to the previous page when it is called.

ContactScreen	Displays a specific contact with the profile information and call history.	
Attributes	contact:Contact	The current contact
	databaseManager:DatabaseManager	Instance of a Database Manager class.
	conversationManager:ConversationManager	Instance of a Conversation Manager class.
Methods	onCallPressed():boolean	Creates a call with the contact when it is called.
	onSharePressed():boolean	Opens sharing options when it is called.
	onRemovePressed():void	Removes the contact when it is called.
	onPicPressed():boolean	Shows the profile picture of the contact in full-screen mode.
	onPreviousCallSlide(call:Call):boolean	Deletes the slided previous call instance from all the saved places.
	onBackPressed():void	Goes back to the previous page when it is called.

SettingsScreen	Displays the settings for the users to customize the interface, or change the profile information according to their preferences.	
Attributes	user:User	Current user.
	databaseManager:DatabaseManager	Instance of a Database Manager class.

	firebaseManager:FirebaseManager	Instance of a Firebase Manager class.
	preferenceManager:SharedPreferencesManager	Instance of a Shared Preferences Manager class.
Methods	onProfilePressed():void	Shows a modal sheet in order to select a new profile picture that is taken from the camera or the gallery. The sheet also contains a selection for deleting the profile picture of the user.
	onNamePressed():boolean	Handles the input to change the name when it is called.
	onEmailPressed():boolean	Handles the input to change the email when it is called.
	onThemeSwitcherPressed():void	Handles the input to change the theme when it is called.
	onLanguagePressed():void	Handles the input to change the language when it is called.
	onChangePasswordPressed():boolean	Navigates to the change password options.
	onSignOutPressed():boolean	Signs the user out when it is called.
	onDeleteAccountPressed():boolean	Deletes the account of the user when it is called.
	showAboutDialog():void	Shows the about dialog when it is called.
	onBackPressed():void	Goes back to the previous page when it is called.

4.5.2. Managers

PageHolder	Main manager to handle screen translation throughout the app. Basically, a super-class that contains each screen and handles the navigation.	
Attributes	currentPage: Integer	Holds the id of the current page.

	controller:PageController	Instance of PageController class.
Methods	showProfileDialog():void	Shows the dialog containing user information and Qr Code of the user.
	onNavButtonPressed(pageToNavigate: Integer):void	Handles the navigation from currentPage to given page.
	onSettingsPressed():void	Navigates to the settings screen when it is called.
	onBackPressed():void	Closes the application.

FirabaseManager	Manages the information updates and retrieval from Firebase. All methods of this class use Firebase to operate requests.	
Attributes	instance:FirebaseManager	Instance of the class itself
Methods	getInstance():FirebaseManager	Retrieves the instance of the class.
	getCurrentUserData():User	Retrieves the current data of the user.
	loginUser():boolean	Logs in the user.
	signUpUser():boolean	Signs up the user and updates Firebase.
	sendForgetPasswordRequest():boolean	Sends a request to Firebase in order to send a reset password mail.
	createCall(callID:String):boolean	Saves the shareable callID to the Firebase.
	joinCall(callID:String):boolean	Saves userID to the given shareable callID which exists in the Firebase.
	removeFromCall(callID:String, contactID:String):boolean	Removes the userID from the given shareable callID which exists in Firebase.
	addToCall(callID:String, contactID:String):boolean	Add a contact to the call with respect to the given callID which exists in Firebase.

	finishCall(callID:String):boolean	Deletes the callID from the Firebase.
	getUserContacts(userID:String): List<Contact>	Retrieves the contact list of the user from the Firebase.
	addContactWithMail(email:String):boolean	Sends a request to the contact that exists in Firebase with the given e-mail.
	addContactWithQr(qr:Image):boolean	Sends a request to the contact that exists in Firebase with the given qr code.
	removeContact(contactID:String):boolean	Removes a contact from the user's contact list stored in Firebase.
	getInvitations(userID:String):List<Contact>	Retrieves contact requests waiting for approval in the Firebase for the user.
	acceptContact(contactID:String):boolean	Accepts the contact request and adds the contact to the user's contacts stored in Firebase.
	rejectContact(contactID:String):boolean	Rejects the contact request and deletes the invitation from the list stored in Firebase.
	getContact(contactID:String):boolean	Retrieves specific contact information by using the given contactID from the Firebase.
	changeProfilePicture(newPic:Image):boolean	Changes the profile picture of the user stored in Firebase.
	changeName(newName:String):boolean	Changes the name of the user stored in Firebase.
	changeMail(newMail:String):boolean	Changes the email of the user stored in Firebase.
	changePassword(old:String, new:String):boolean	Changes the password of the user stored in Firebase.
	signUserOut(userID:String):boolean	Signs the user out and invokes Firebase for the logout event.
	deleteUser(userID:String):boolean	Deletes the user and all the information about it from the Firebase.

DatabaseManager	Updates the database and retrieves information about the contacts according to the user.	
Attributes	instance: DatabaseManager	Instance of the class itself.
Methods	getInstance(): DatabaseManager	Retrieves the instance of the class.
	getPreviousCalls():List<Call>	Retrieves the list of previous calls.
	getContacts():List<Contact>	Retrieves the list of contacts from the database.
	addContact(contactID:String):boolean	Saves the given contactID to the database.
	deleteEverything():boolean	Clears the database including all saved instances of previous calls and saved contacts.

ConversationManager	Manages the online conference calls.	
Attributes	instance:ConversationManager	Instance of the class itself.
	lastTranslation: String | List<Image>	Holds the last translation information. It can be a text translated from video data or a video generated from text.
Methods	getInstance():ConversationManager	Retrieves the instance of the class.
	createCall(userID:String):String	Creates a call on behalf of the user.
	joinCall(callID:String):boolean	Joins the user to the call with the given callID which was created before.
	endCall(callID:String):boolean	Ends the call.
	removeUserFromCall(userID:String):boolean	Removes the user from the call.
	addUserToCall(contactID:String)	Adds a user to the call.

	:boolean	
	receiveVideoData(callID:String): List<Image>	Receives the video data coming from the server and feeds to the UI.
	sendVideoData(callID:String, video:List<Image>):boolean	Sends the video output coming from the camera to share the video to the server that the call is going on.
	changeLastTranslation(latestTranslation):void	Updates the last translated data.
	getLastTranslation():String	Retrieves the last translation information.

SharedPreferences Manager	Manages the interface settings according to user preferences.	
Attributes	instance: SharedPreferencesManager	Instance of the class itself.
Methods	getInstance(): SharedPreferencesManager	Retrieves the instance of the class.
	addOrChangeUserID(userID:String):boolean	Saves the UserID created by a Firebase instance to preferences store. If an existing value is found, an update operation will be done.
	getUserID():String	Retrieves the user ID.
	setLoggedIn(loginState:boolean) :boolean	Sets the logged in state of the user.
	isLoggedIn():boolean	Checks if the user is logged in.
	setLanguagePreference(language :String):boolean getLanguagePreference():boolean	Retrieves and updates the language preferences of the user according to the input.
	changeTheme(themeMode:String):boolean getTheme():String	Retrieves and updates the application's theme according to the input.
	setLatestCall(callID:String):bool	Retrieves and updates the latest

	ean getLatestCall():String	call of the user.
	setMicState(micState:MicState): boolean getMicState():MicState	Retrieves and updates the latest state of the microphone.
	setCamState(camState:CamState):boolean getCamState():CamState	Retrieves and updates the latest state of the camera.
	setTranslationType(type:Translat ionType):boolean getTranslationType():boolean	Retrieves and updates the latest translation type of the user.
	deleteEverything():boolean	Clears the shared preferences data store, including deleting every key and their corresponding values.

TranslationManage r	Creates a bridge and handles the I/O operations between Flutter interface and Python interface.	
Attributes	instance: TranslationManager	Instance of the class itself.
Methods	createBridge(): TFLiteModelInstace	Creates a bridge between Python Manager instance and Dart Manager instance and returns it as a TFLite Model
	getInstance(): TranslationManager	Retrieves the instance of the class.
	translate(transType: TranslationType): String | List<Image>	Translates the given input and returns a string if the given translation type is TranslationType.SITT or TranslationType.SETT or a List of images which corresponds to a video if the translation type is TranslationType.TTS
	translateSpeech(speech:File): List<Image>	Translates given speech and returns a video output.
	translateText(text:String): List<Image>	Translates given test and returns a video output.
	translateSign(List<Image>):	Translate given video in sign

	String	language to text.

4.5.3. Model

Conversation	A model class to hold information about online conversation (online conference call).	
Attributes	conversationId:int	Discriminative id for the call.
	date:Date	Date of the conversation created.
	duration:TimeStamp	Duration of the conversation after the creation.
	participants:List<User>	List of users participated in the call.
Methods	addParticipant(userId:int): boolean	Adds a user to the conversation.
	removeParticipant(userId:int):boolean	Removes a participant from the conversation.

User	A model class to represent the user.	
Attributes	user_info: Contact	Holds the information about user's id, name, e-mail, profile picture and qr code.
	user_contacts:List<Contact>	Holds the contact list of the user
Methods	addContact(QRData: bytearray): boolean	Adds the contact by qr code.
	addContact(email:String):boolean	Adds the contact by email
	removeContact(otherUserId:int): boolean	Removes the contact from the contacts list.

Contact	A model class that holds the information of the contact.	
Attributes	userId:int	Discriminative id of the

		contact.
	name:String	Name of the contact.
	email:String	E-mail of the contact.
	picture:bytearray	Profile picture of the contact.
	QRData:bytearray	Qr code assigned to the contact.

4.6. Translation Pipeline

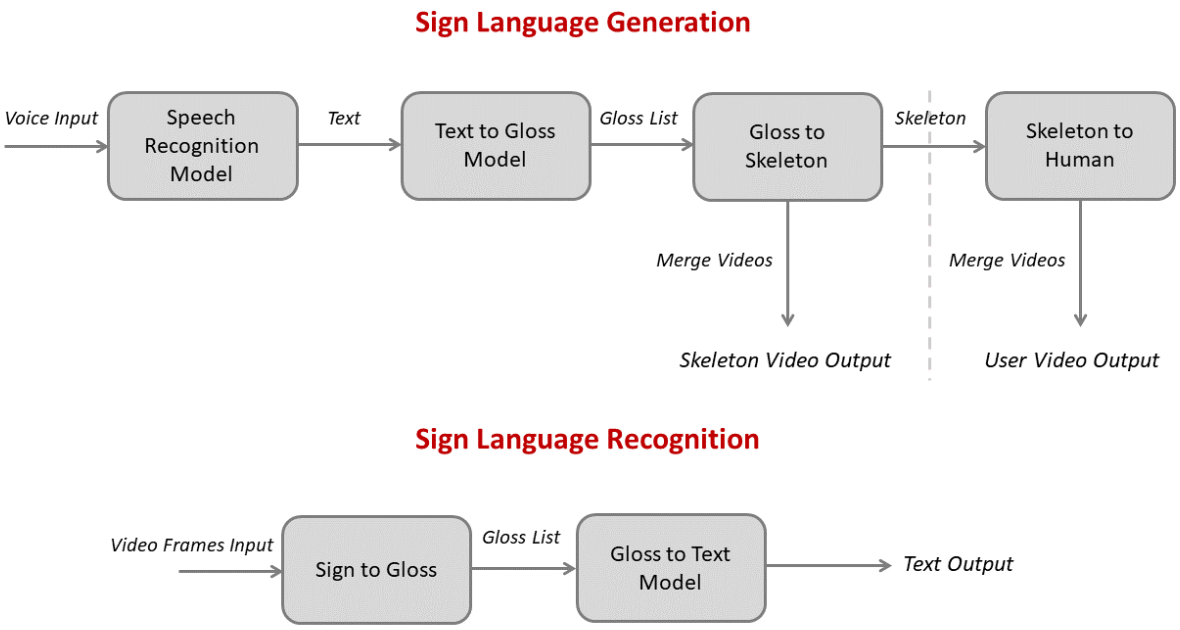

Figure 6: *Bidirectional Translation Pipeline*

The translation consists of two different pipelines. These are Sign Language Generation and Sign Language Recognition as shown in Figure 6.

Sign Language Generation:

- Speech Recognition Model: Used Flutter libraries to integrate this model. It gets the voice input from the application and converts the speech to the corresponding text. The text output is then sent to the FastAPI server.
- Text to Gloss Model: It takes text input and tokenizes the sentence using StanfordNLP libraries. Then, it converts the words to their base forms. Post-processing the words, it constructs the glosses according to the ASL grammar. Finally, it outputs the gloss list.

- Gloss to Skeleton Model: Takes gloss list as input and finds corresponding skeletons generated by OpenPose model. Then, by default, it merges the skeleton videos to construct the final sentence and outputs a skeleton video.
- Skeleton to Human Model: This is an optional step as this feature is still under improvement. If the user chooses this option to try, it is then activated. It takes skeletons as inputs and constructs the videos with user images using a GAN model. The output is then sent back to the mobile application.

Sign Language Recognition:

- Sign to Gloss Model: The application takes frames from the camera or WebRTC server and sends the frames to the local server. Then, the sign recognition model predicts the sign and outputs the corresponding glosses.
- Gloss to Text Model: It takes the gloss list and converts these glosses into an English sentence by using an NLP grammar library called HappyTextToText. Then, it outputs the sentence and sends the result to the application through local APIs.

5. Testing Details

Our system is not so complex that required a separate testing environment. That is why we did not implement any tools or test modules independentyl. Instead, each team member is assigned to a different sub-system and we continuously applied manual testing and specifi testing libraries for the completeness and performance of each sub-system. As we were meeting at least once each week, we managed to test the systems every time we added something new. That allowed us to detect the errors in the early steps and continue stacking up to the error-free systems.

5.1. Testing User Interface

The user interface of the application is one of the most important parts of our application. We used manual testing for each screen. Additionally, as we used Flutter for the whole UI system, it was easier to construct tests as Flutter contains testing libraries that can be integrated easily. We checked the system's performance, localization, and user interface response by creating widgets. We also used manual testing to evaluate the performance of each screen on different operations, including user-specific choices on the visual options of UI.

5.2. Testing APIs

For the local server to run our translation models, we used FastAPI. It was important to make our APIs run properly as our main features relied on the requests that are executed on them. That is why we paid attention to handling the errors and warnings appropriately and testing our APIs performance and continuity. To do this, we mainly used test client libraries specific to FastAPI.

5.3. Testing the Performance of the Translation Models

For evaluating the performance of our translation models in each stage of the pipeline, we tested each model with our own videos and sentences. For the text-to-sign pipeline, we generated sample sentences to check text-to-gloss and reviewed the ground truth from ASL grammar sites. According to the results of NLP models, we had post-processing steps to construct the most similar gloss representation of the sentences. Then, we tested the outputs of OpenPose, which was used to generate skeletons. For the sign-to-text pipeline, we tested the trained sign recognition model with our own videos and fine-tuned the hyperparameters for the optimal output.

6. Maintenance Plan and Details

6.1. Server Maintenance

The servers of Signify are mainly used to execute machine learning models. Currently, we use local servers and post requests from APIs, however, our future plans are different. For now, we do not have a maintainable server as our local host cannot be open for forever. As our models require a large amount of memory, it is also not possible to maintain our models on the application. Therefore, with the help of a budget or investment, our future plans include paying extra fees to maintain a space in a new server.

6.2. Database Maintenance

Signify stores the user data in Firebase DB. Maintaining this database updated is a crucial part of the application as all user information is kept here. To keep our database safe and up to date, we should optimize queries and perform regular backup operations. We also need to optimize the changes to keep the database structure simple and efficient.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

To conduct a project, different factors should be considered to produce a product that provides a great user experience, solves people's problems, etc.

The main factor for the developers is to organize and plan the project's progress. Since it is a teamwork project, team members should collaborate and communicate with each other. One part might have a huge impact on other parts; therefore, team members should consider the consequences of their actions. To ensure this communication and collaboration, we followed agile and scrum methodology for developing the project. The team had weekly meetings to determine the road map and discuss the current situation.

Another major consideration is related to user experience as the application targets specific people. The application is designed user-friendly so that no drawback occurs. It has not only a modern user interface but also a simple, target-specific, flawless, and fast design to provide the best user experience, specifically for speech/hearing-impaired users.

Furthermore, the user should have no security concerns. Personal and sensitive data should be protected against any adversarial attack. Nowadays, a lot of applications are exposed to adversarial attacks. The application is designed to be protected against them. All personal and sensitive data is encrypted and stored safely. Also, any non-required data which has no impact on application performance are not stored or taken by the user. To comply with “General Data Protection Regulation” - GDPR and “Kişisel Verilerin Korunumu Kanunu” - KVKK, any personal and sensitive data are not shared with third parties.

7.2. Ethics and Professional Responsibilities

Our target users are hearing and speech impaired people; therefore, some ethical and professional responsibilities are taken. The most significant moral obligation is to provide equal opportunities in the usage of the application.

To ensure the ethical responsibility mentioned above, the dataset used in training is checked not to be biased. The dataset should be representative of all ethnic groups. The training data appeals to all the users to not humiliate the users' identity and to provide a better user experience. Since the project aims to improve the standard of hearing and speech impaired

people, it should be sufficient and representative to make correct and similarly accurate predictions without discriminating against any user. Furthermore, the output of the voice to text and sign language models should not be discriminative. The caption generated by the application should not be biased regarding any race, gender, or social class.

Furthermore, some conversations might include private information about the user; therefore, this information should be protected and should not be shared with other users or third-party companies. As the application grows, user feedback and some recordings of the meetings can be utilized to enhance the performance of the machine learning models. In these cases, any personal information, such as name, address, and the job should not be revealed. Additionally, to serve the machine learning models, we used APIs; however, any information about the user is not sent to APIs to protect the user's data.

In conclusion, in the development of this project, a representative and unbiased dataset is selected so as not to cause any discrimination. Moreover, the machine learning models' output did not include phrases that imply gender inequality, humiliation, and ethnic-based bias. Finally, users' personal information is not shared with other users or third-party companies and is protected.

7.3. Judgments and Impacts on Various Contexts

Signify takes various contexts into account as discussed in the table below.

	Effect Level	Effect
Cultural Factors	9	Change in learning model outputs due to cultural differences in spoken-sign language.
Public Safety	8	General Data Protection Regulation
Public Health	1	Impaired people's health should be considered, however, it is not the main purpose of the application.
Public Welfare	2	Reducing the communication limits among impaired people.
Economic Factors	6	The application should be free.
Environmental Factors	1	It is not in the scope of this project.
Social Factors	5	Language barrier for hearing impaired people in the sign up process.

Global Factors	8	Translation is done only between English and ASL.
Technological Factors	10	Change in design of learning models due to the publication of new technologies.

Table 1: *Factors that can affect analysis and design*

7.4. Teamwork Details

7.4.1. Contributing and Functioning Effectively on the Team

The most important factor of proper teamwork is communication. As a team, we scheduled a time when everyone is available. During the development, we met at least once and discussed the progress so far and regulate the future plans. New responsibilities were distributed to each team member considering the workload and previous experiences. While distributing the tasks, various factors are taken into account. Each team member has his/her own weaknesses and strengths in different fields, and different interests; thus, assignments of tasks are done accordingly. We have worked together before, so it was easier to decide and communicate based on our previous experiences.

In addition, we have scheduled the works that should be completed by a given deadline that was set either by the department or our team. We gave great importance to the deadlines and finished our tasks accordingly. Deadlines were also crucial for a functioning project and future progress.

7.4.2. Helping Creating a Collaborative and Inclusive Environment

The works are separated among team members; however, that does not mean we worked only on the tasks given. When someone had a problem or needs assistance, other team members became involved in the work to achieve better outcomes. Additionally, after a task is completed, it was reviewed by other members. Collaborative reviewing was one of the significant facts of teamwork. One can overlook mistakes he/she has done; however, others can see and correct those mistakes. By reviewing each work done, we aimed to improve our design and eliminate errors that can influence our future work.

7.4.3. Taking Lead Role and Sharing Leadership on the Team

As part of the planning activity, the project goals are listed and divided into work packages (WPs). As going through each WP, new sub-packages were created to divide the responsibilities. Each work package had its own leader and at least two students worked on a WP. Although the contributions of the members are equally divided, leaders had the extra responsibility to regulate the process of the work packages they were assigned to.

WP#	Work package title	Leader	Members involved
WP1	Project Specifications Report	Çağlar	Everyone
WP2	Analysis Report	Ali Taha	Everyone
WP3	User Interface Implementation	İrem	Ali Taha
WP4	High Level Design Report	Naci	Everyone
WP5	Sign Language Translation (From Video to Text)	Sena	Çağlar
WP6	Sign Language Translation (From Text to Animation)	Çağlar	Naci
WP7	Database Connection	Ali Taha	Sena
WP8	Mid Testing	Naci	Everyone
WP9	Demo-Presentation	Sena	Everyone
WP10	Real Life Communication	İrem	Ali Taha
WP11	Video Conference Communication	Naci	İrem
WP12	Low-Level Design Report	Çağlar	Everyone
WP13	Final Testing	Ali Taha	Everyone
WP14	Final Report	Sena	Everyone
WP15	Final Presentation	İrem	Everyone

Table 2: *List of Work Packages*

7.4.4. Meeting objectives

The objectives mentioned in previous reports were successfully implemented. However, due to various limitations and the ambiguity in state-of-the-art models, changes are done to some

features and backend development. For example, our first design included a text-to-sign translation where the generated video was planned to be the user him/herself. Due to the performance issues in the GAN model and the ambiguity of the results, we have decided to show the generated skeletons only. The generation of the user's ASL video is added as an option until we get to improve the model enough to propose as an application feature. Additionally, our first backend design included the translation models integrated into the application itself, to get the translation results faster. However, due to the size of the models and the complexity of the deployment, we have instead decided to use API's to run the models. We also optimized the communication through API's to provide a real-time experience as we promised in our previous objectives.

Besides the successful implementation of our objectives, we have added extra features and tutorials to Signify. The main page of the application provides additional sections to learn word-level ASL and see the latest news about ASL.

7.5. New Knowledge Acquired and Applied

As Signify, we have acquired various hard and soft skills during the project's design and implementation. The most important factor of our project was to translate ASL properly. To do that, we have learned some basic words and daily conversations along with the ASL grammar in simple sentences. It was necessary for us to learn ASL as we needed to check the correctness and completeness of our models.

Another important factor was to successfully translate the sign language in both directions. We needed to understand the pipeline for both sign-to-text and text-to-sign translation. We conducted a literature review for both translations and constructed our baseline accordingly. As explained detailly in section 4, we have a bidirectional translation pipeline that requires different models for each stage. For text-to-gloss and gloss-to-text models, we have learned about various natural language analysis packages that contain tools to tokenize texts and convert the languages into lists of base-formed words and morphological features. For gloss-to-sign and sign-to-gloss models, we have learned Openpose to generate the skeleton for the corresponding input. We also trained a GAN model to generate user-specific video, where we use the image of the user and generated a skeleton to construct a video of the user doing ASL. We learned the basics of GAN and several different implementations of the network.

For the backend communication in the application, we first decided to use the user's local memory to store the models and have a faster translation. However, we encountered serious limitations in both the memory capacity of the phone and the deployment of the learning models. That is why, we decided to use FastAPI and optimized the communication between the host computer and our mobile application. We learned how to share information through APIs and how to provide a translation near real-time.

8. Conclusion and Future Work

In the end of the project, we successfully created the application we proposed ensuring the completeness of our main objectives. Signify is able to work both on Android and iOS with the help of its Flutter backend. We can now offer a communication application for hearing/speech impaired people by bidirectional translation in both online calls and real-life scenarios.

Although we have completed all our tasks, Signify is open to future extensions and performance improvements, especially for translation models. For the application part, we designed the software architecture in a way that new features can be added and a new server can be hosted easily. For the learning models, we used the latest state-of-the-art models to provide the most accurate translation service. However, bidirectional ASL translation is still a continuing research area that requires significant improvements. It is open to new ideas, more comprehensive datasets, and improved network architectures to reach a satisfactory level of translation.

During this project, we encountered various problems both in theory and in practice due to the challenging features we proposed. However, we continued working as a team and tried our best to accomplish what we want to do. We learned a lot and also improved our soft skills such as teamwork, leadership, and time management. Even though there were times we felt hopeless, we are very proud of ourselves as we successfully overcame all the hardships.

9. Glossary

- Online conference: A feature where people can easily communicate online through our application.
- Real-life communication: A feature where people can easily communicate in real life through our application.
- Bidirectional translation: Translation from text to sign language and from sign language to text.
- ASL: American Sign Language, the chosen sign language for the application.
- UI: User interface of the application.
- DB: Database that holds the user related data.
- ML: Machine learning.
- GAN: Generative adversarial networks that are used to generate an ASL animation from text information.
- NLP: Natural Language Processing
- OpenPose: The model used for the generation of skeletons.
- Model data: Data that is used for the design and improvement of ML models.

10. References

[1] E. McPhillips , “World wide hearing loss: Stats from around the world,” *Audicus*, 14-Sep-2021. [Online]. Available: https://www.audicus.com/world-wide-hearing-loss-stats-from-around-the-world/. [Accessed: 05-Oct-2021].

[2] “Every 4th person to suffer hearing loss by 2050: Who,” *Down To Earth*. [Online]. Available: https://www.downtoearth.org.in/news/health/every-4th-person-to-suffer-hearing-loss-by-2050-who-75718. [Accessed: 05-Oct-2021].

[3] “Hand talk translator - apps on Google Play,” *Google*. [Online]. Available: https://play.google.com/store/apps/details?id=br.com.handtalk. [Accessed: 05-Oct-2021].

[4] “ASL translator - apps on Google Play,” *Google*. [Online]. Available: https://play.google.com/store/apps/details?id=com.asltranslator. [Accessed: 05-Oct-2021].

[5] “S L a I T. real-time sign language translator with AI.,” *S L A I T. Real-time Sign Language Translator with AI.* [Online]. Available: https://slait.ai/. [Accessed: 05-Oct-2021].