SIGNFLOW TRANSLATOR APP

TOHEED UL HAQ
01-134221-082
RANA AMANULLAH
01-134221-067

Supervisor: Dr. Erum Ashraf

Bachelor of Science in Computer Science

Department of Computer Science
Bahria University, Islamabad

December 2025

Certificate

We accept the work contained in the report titled "Sign Flow Translator App", written by Mr. Toheed Ul Haq AND Mr. Rana Amanullah as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Computer Science.

Approved by . . . :

Supervisor: Dr. Erum Ashraf

_____

Internal Examiner: Ms. Nadia Kalsoom

_____

External Examiner: Ms. Madiha Khalid

_____

Project Coordinator: Dr. Asim Ali

_____

Head of the Department: Dr. Khurram Ehsan

_____

Abstract

The Sign Flow Translator App is designed to bridge the communication gap between the deaf and hearing communities by translating Pakistani Sign Language (PSL) gestures into text in real time. The SignFlow Translator App aims to bridge this gap by providing a real-time, AI-powered sign language translation system. Leveraging Computer Vision, the application translates sign language gestures into grammatically correct text. Built using modern frameworks such as Flutter for mobile application development and Django for backend, the app supports multi-modal inputs through live webcam feeds. The project adopts an Agile development methodology, ensuring iterative improvements across data collection, model training, and UI/UX design. Though limited in initial scope to text-only and one-way communication. SignFlow sets a strong foundation for scalable and accessible sign language solutions across healthcare, education, and public service sectors.

Acknowledgments

All gratitude is due to Almighty Allah, who gave us a little fraction of his vast wisdom, as well as for the good health and wellbeing required to finish this report. We wish to express our sincere thanks to Dr. Erum Ashraf, for guiding us at various stages of the project and providing us with all the necessary facilities for the research. We are extremely thankful and indebted to her for sharing expertise, sincere and valuable guidance and encouragement. This project would not have been completed without their supervision, advice, and invaluable guidance. We are grateful to them for their support and encouragement throughout this project. We would also like to extend our deepest gratitude to Dr. Neelma Naz, whose generosity in providing the necessary sign language dataset was foundational to the success of our research. We would want to take this opportunity to thank everyone of the department's faculty for their assistance and support. We also like to thank our parents for their constant love, care, and support.

TOHEED UL HAQ AND RANA AMANULLAH
Bahria University
E-8, Islamabad, Pakistan

December 2025

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AI	Artificial Intelligence
ASL	American Sign Language
CNN	Convolutional Neural Network
CSRF	Cross-Site Request Forgery
CV	Computer Vision
DFD	Data Flow Diagram
DRF	Django REST Framework
EGCN	Efficient Graph Convolutional Network
GCN	Graph Convolutional Network
GUI	Graphical User Interface
IDE	Integrated Development Environment
LSTM	Long Short-Term Memory
ML	Machine Learning
NFR	Nonfunctional Requirement
NLP	Natural Language Processing
PSL	Pakistani Sign Language
SDLC	Software Development Life Cycle
SL	Sign Language
UI	User Interface
UML	Unified Modeling Language
UX	User Experience
WHO	World Health Organization

Chapter 1

Introduction

The basis of human interrelation is communication, Sign languages (SLs) are communication systems based on non-verbal language, which are employed by people with hearing and speech disabilities. The world health organization estimates that more than 430 million people in the world have hearing and speech impairments with 70 million of these people being the primary users of sign languages as their communication channel, according to the report of the world federation of the deaf [1]. SL is a life line, as it has facilitated a smooth communication between the hearing and speech impaired. Nevertheless, the distance between the users of SL and the rest of the vocal community is still enormous, which restricts its accessibility in education, healthcare, workplace, and social life. SL is a vital equivalent to all spoken languages. In recent years, the role that Artificial Intelligence (AI) plays in everyday life has been growing more prominent. The applications of this technological revolution are promising to the hearing-impaired community because of the growing use of this technology. The translation systems of SL are a realistic and practical concept that can be applied by the researchers. SL is an essential means of communication that is used by individuals with hearing and speech deficits. SignFlow would help close that gap by creating a real-time SL Translator application based on deep learning and natural language processing (NLP). The SignFlow Translator App will convert SL gestures to texts through NLP thus facilitating smooth interaction among SL users.

1.1 Project Background/Overview

The SignFlow Translator App is aimed at enhancing accessibility, and applicability of sign language contents in different fields, such as business, academia, and media. It is devoted to identifying and translating the Pakistani Sign Language (PSL) based on the video input, which helps to ensure the successful communication with people, who are deaf. The product is guided by addressing the gap between the PSL users and the non-users by incorporating the machine learning models, computer vision, and natural language processing.

1.2 Problem Description

Sign languages are pictorial and they are based on a blend of hand shapes, palm orientations, movement patterns, and facial expressions to communicate a meaning. Nonetheless, these signs only make sense to them based on specific training thus posing communication barriers to the deaf and hard of hearing individuals [2]. In this regard, a number of sign language recognition systems have been created with time. The previous techniques involved wearable sensors, which were frequently restrictive to natural movement and had rigorous requirements, so they were not always applicable in practice. Along with the development of computer vision and deep learning, the current direction of study is now on the vision-based methods that can identify signs more accurately and naturally. Our SignFlow Translator software is based on this new methodology as it creates an AI-driven model to translate PSL to text in real time to make the communication process more open and approachable.

1.3 Project Objectives

The objectives of proposed SignFlow Translator App are as follows:

- To support AI model input for SL translation through live camera feeds.
- To support PSL translation.
- To convert recognized gestures into meaningful words.
- To design an intuitive and accessible UI/UX for users with hearing or speech impairments.

1.4 Project Scope

The project will be aimed at creating a real-time SL translation system based on AI with outstanding features. They are characterized by the support of AI models with the help of TensorFlow, PyTorch, and OpenCV to ensure the best accuracy and reduced computational costs. MediaPipe improves gestures and pose detection. The system will combine Django to process on the back-end and Flutter to provide a unified cross-platform mobile experience, which will be responsive. To the project limitations, SignFlow Translator App does not yet have full Pakistani sign language as its vocabulary. The reason is that there is no formal and standardized sign language system in place, instituted by a governing body in Pakistan. Our project is at the moment founded on PakSign subset of data because of this constraint.

1.5 Application Areas

The application developed can be applied in many areas where communication is a requisite especially to people who have hearing and communication problems like in the healthcare sector, the educational sector as well as in the day to day business dealings where the application will enable easy communication.

1.6 Summary

One of the real-life issues that are impeding accessibility across most of the sectors is the communication gap that sign language users face. SignFlow Translator App is one of the solutions to this issue as it utilizes modern AI and deep learning to translate PSL gestures into text in real-time. The proposed project is based on the creation of an accessible, mobile-first solution to create more inclusive communication to the deaf community.

Chapter 2

Literature Review

The high rate of technology development has introduced new opportunities on the way towards creating sign language recognition systems. Scientists all over the world have been diligently involved in developing systems that can turn the sign language in texts or speech and most of the above systems fail at real-time accuracy and contextual interpretation. The previous models were based on two-dimensional picture or recorded videos and hence were not very effective in natural conversation. Most of the translators who are available do not appreciate continuous gestures, facial expressions, lighting, clothing and movement of the signers, hence translating them incomplete or misconstrued. This literature review will discuss previous studies, limitation existing in various studies, and how SignFlow solved these challenges.

2.1 Comparative Study

An exhaustive introspection of the new technologies in the field indicates that there are a number of methods of identifying sign language but each has its limitations, which inform our solution. They are largely based on deep learning architecture such as LSTMs or CNNs, but commonly they have difficulties in dependency on sensors, or computational costs.

- **SignVista:** SignVista is a video-based ASL project where the main emphasis is put on fingerspelling recognition with the help of video-based data and models such as RFS Model to perform real-time, two-way communication. It is restricted to the ASL language [3].

- **SignChatter:** SignChatter is a real-time, one-way communication project that utilizes an LSTM Model to handle both ASL and PSL based on video input for word-level translation. Its reliance on an LSTM model often makes it computationally heavier and potentially slower for deployment [4].
- **Sign Language Translator Application using AI:** The ”Sign Language Translator Application using AI” is a real-time, one-way system that performs word-level translation for PSL primarily using a conventional LSTM Model based on video input. It uses LSTM on raw video frames can be highly demanding on processing power, leading to potential latency issues during real-time use [5].
- **Urdu Sign Language Recognition:** The ”Urdu Sign Language Recognition” project focuses on real-time, one-way recognition of PSL alphabets using an Image-based dataset and a standard CNN Model. It is limited to the alphabet level [6].
- **Pakistan sign language to Urdu translator using Kinect:** The ”Pakistan sign language to Urdu translator using Kinect” is a real-time, one-way system that uses an LSTM Model and a Kinect sensor to recognize PSL alphabets from an image-based dataset[6]. The key limitation here is the reliance on the expensive and non-mobile Kinect sensor, restricting the application’s portability and widespread use [7].

Ref	System	Approach	Lang	Scope	Precision	Mode
1	SignVista	RFS Model	ASL	Video-based finger spellings	Real Time	Two-way
2	SignChatte	LSTM Model	ASL + PSL	Video-based word level	Real Time	One-way
3	Sign Language Translator App	LSTM Model	PSL	Video-based word level	Real Time	One-way
4	Urdu Sign Language Recognition	CNN Model	PSL	Image-based alphabets	Real Time	One-way
5	PSL to Urdu using Kinect	LSTM Model	PSL	Image-based alphabets	Real Time	One-way

Table 2.1: Comparison and Performance Metrics of Sign Language Systems

2.2 Summary

The SignFlow Translator is a three-tier mobile application designed for real-time translation of 20 distinct Pakistani Sign Language (PSL) signs into Urdu text. Unlike comparable projects like SignVista or Kinect-based systems, SignFlow is specifically focused on PSL and relies on standard mobile cameras. It avoids the computational heaviness of raw video processing by employing MediaPipe for efficient 65-keypoint extraction from video frames. The core technology is the EfficientGCN-B0 model, a specialized Graph Convolutional Network trained on the PakSign dataset, ensuring rapid and accurate classification of the resulting skeletal data. The system is built with a Flutter/Dart frontend and a Django Python backend that manages the API, user authentication, and executes the deep learning model.

Chapter 3

Requirement Specifications

System requirements are configurations that a system needs in order to function properly and effectively. Failure to comply with these requirements may cause installation issues or performance issues. System requirements are documents or set of documents that outline the features and behaviour of a system or software application.

3.1 Existing Systems

A review of existing sign language solutions reveals three significant gaps that the SignFlow Translator addresses. First, many projects are impractical for widespread adoption due to their reliance on specialized hardware like gloves or Kinect sensors. Second, most computer vision (CV) applications are limited to static image classification (like alphabets), which fundamentally fails to interpret the dynamic and sequential nature of word-level signs. This inability to analyze motion means they cannot distinguish context-dependent actions. Finally, a critical technical gap in current systems like SignChatter is their reliance on computationally heavy LSTM Models for processing video, which leads to high latency and makes them impractical for real-time mobile deployment.

3.2 Proposed System

The proposed SignFlow Translator is a dedicated PSL-to-Urdu communication tool designed to overcome the limitations of prevailing image-based recognition systems that are often restricted to static alphabets or rely on specialized hardware. It utilizes a novel, data-efficient approach based on the EfficientGCN-B0 architecture for superior skeleton-based spatiotemporal analysis of dynamic sign language motion. This decision directly resolves the issue of computational load and latency of older LSTM-based systems which process

small, high-value MediaPipe keypoints instead of video frames. The system is fitted with the complete PSL PakSign skeleton dataset [8], which allows the correct classification at the word level, and is deployed through a robust Django/DRF backend to ensure safe model classification, as well as a responsive Flutter/Dart front-end to ensure mobile accessibility.

3.3 Feasibility Analysis

The following are some of the feasibility considerations during development:

3.3.1 Technical Feasibility

Backed up by such frameworks as TensorFlow, PyTorch, MediaPipe, and Django.

3.3.2 Operational Feasibility

Addresses practical demands of the hearing-impaired communities by supporting them in multiple languages.

3.3.3 Economic Feasibility

The tools that come at no cost are open-source; could generate revenue through sponsorships and premium features.

3.3.4 Legal Feasibility

Adherence to the accessibility laws of Pakistan Association of the Deaf.

3.4 Requirement Specifications

In this section, the unique aspects of the SignFlow application will be outlined in terms of its behavior and restrictions.

3.4.1 Functional Requirements

Functional requirements state what the SignFlow Translator App would need to accomplish.

- **FR1**
  Gesture Recognition: Estimate skeleton keypoints of live sign language gestures with MediaPipe.

- **FR2**
  Real-Time Translation: Estimate skeleton keypoints of live sign language gestures with MediaPipe.
- **FR3**
  User Authentication: To include secure log-in and sign-up options with the use of SQLite authentication.
- **FR4**
  Cross-Platform App: Testing Do not make it difficile on the Android and the iOS devices.

3.4.2 Non-Functional Requirements

Non- Functional Requirements (NFRs) elaborate on the key attributes of a system including its performance, reliability, security, usability, maintainability and portability.

- **NFR 1:**
  The application should have real-time gesture recognition and translation response time should not exceed 2 seconds.
- **NFR 2:**
  To ensure high performance on different phones, the system needs to be optimized on mobile devices with the help of TensorFlow Lite.
- **NFR 3:**
  The APIs of server-side should offer 95 percent uptime.
- **NFR 4:**
  The system should be able to preserve integrity of data of user accounts, preferences and history of translation.
- **NFR 5:**
  The system should adopt end to end encryption in order to protect user data.
- **NFR 6:**
  The application should have the capability to perform role-based authentication to avoid abuse of the sensitive features and data.
- **NFR 7:**
  The application has to offer an easy and user-friendly.
- **NFR 8:**
  The system must include accessibility features designed for hearing-impaired users.

- **NFR 9:**
  The app must follow a modular design to allow quick updates of ML models and UI components.
- **NFR 10:**
  The app must run smoothly on both Android and iOS.
- **NFR 11:**
  The app must remain compatible with various camera hardware commonly used in smartphones.

3.4.3 Software Quality Attributes

The important software quality attributes described in this section make the system usable, secure, and efficiently performing.

- **Usability:** The user interface shall be user-friendly and easy to venture and making the effort and training required to initiate gestures and display translations as minimal as possible.
- **Performance:** The response time in dynamic sign video processing and the response time in giving the Urdu translation should be low so as to guarantee real time responsiveness.
- **Security:** SQLite must be used to encrypt all user data and authenticate them through the security features of the SQLite and strong authentication with tokens.

3.4.4 Resource Requirements

In this section, the key hardware and software needed to train the models and deploy its applications are outlined.

3.4.5 System Requirements

The following section is a description of the hardware and software resources needed to develop and implement the SignFlow application.

3.4.5.1 Hardware Requirements

- Model training needs high-performance CPU/GPU.
- Apple devices should have 4GB or more RAM to run applications.

3.4.5.2 Software Requirements

- **Machine Learning/Vision:** TensorFlow, PyTorch, OpenCV, MediaPipe.
- **Backend/Frontend:** Django, Flutter.
- **Database:** SQLite.

3.4.5.3 Development Team Resources

- **Mobile Developers (Flutter):** Develop the cross-platform interface.
- **Machine Learning Engineers:** Train CNN.
- **Backend Developers:** Manage APIs and databases.
- **UI/UX Designers:** Be usable and accessible do not forget.

3.5 Use Cases

Use cases explains the external behavior of the system in the sense that it illustrates the sequence of activities between the actor and the system. Primarily, they introduce step by step what the system must accomplish to put forward a beneficial output to the user.

The system is made of three major modules.

- User Authentication Module: This module is associated with registration, log in/sign up and successful deletion of personal settings and history of translation.
- Gesture-to-Label Module: CNN and MediaPipe convert sign languages to words.
- Text Generation Module: Interactively provides text generation with the NLP Transformers by constructing meaningful and grammatically correct sentences and by virtually offering next-word prediction.

This model makes sure that users easily authenticate, feed on gestures, as well as get naturally sounding real-time translations.

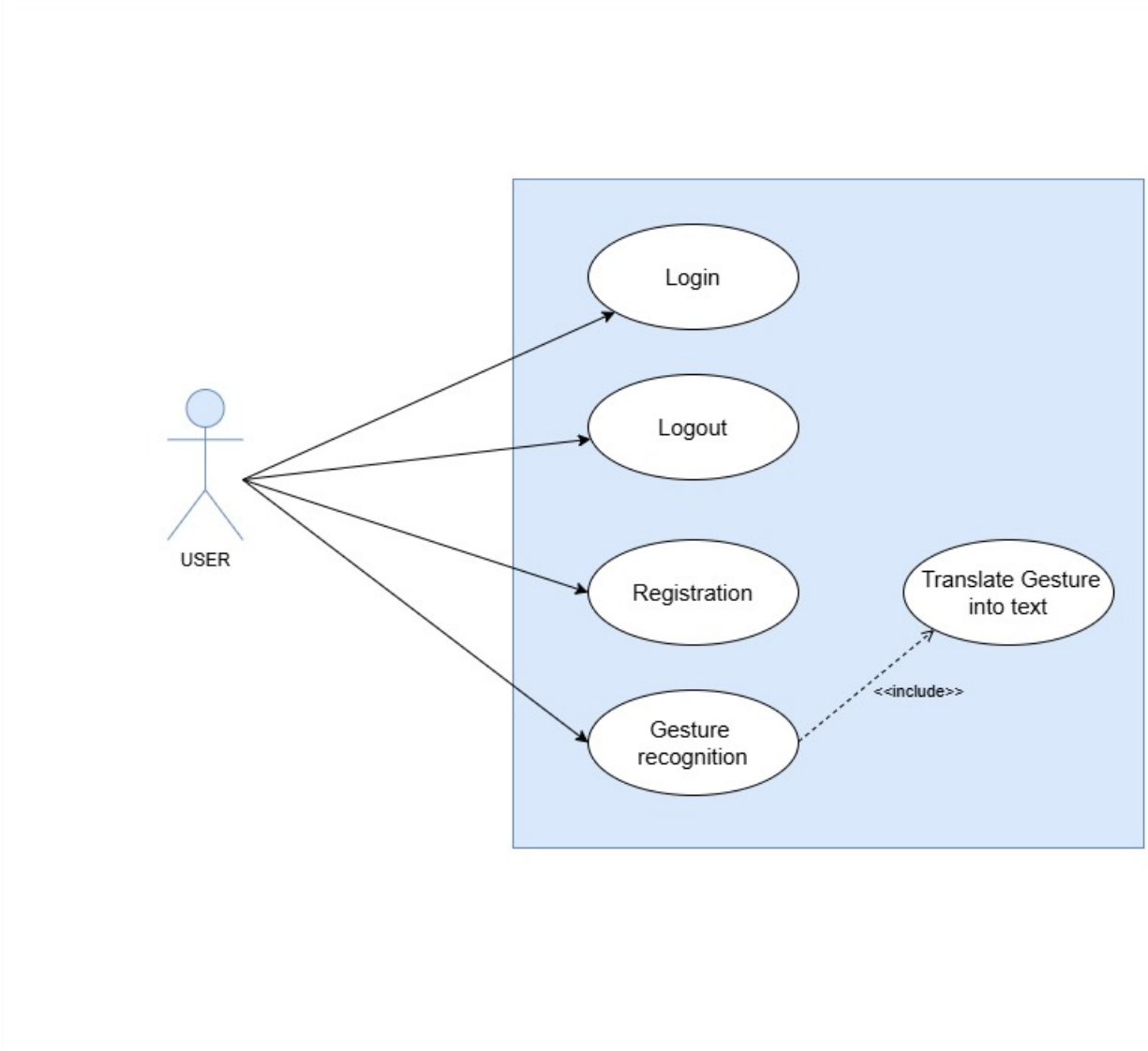

Figure 3.1: Use Case Diagram

The use case diagram (Figure 3.1) highlights key functions.

- Login/Sign Up/Logout: User authentication.
- Gesture Recognition: Capture and classify gestures with EfficientGC-b0 + MediaPipe.

3.5.1 Use Case – Login

This use case describes how a registered user signs into the system using valid credentials.

Use Case ID	UC-1
Use Case Name	Login
Actor(s)	User
Priority	High
Pre-Conditions	User has already registered an account.
Basic Flow	1. User opens the mobile app. 2. User enters email and password. 3. User clicks Login.
Actor Actions / System Response	User enters login details, then System verifies credentials. If valid, then User is authenticated and redirected to Home screen.
Alternative Flow	If login fails, system displays an error message.

Table 3.1: Login Use Case

3.5.2 Use Case – Logout

This use case explains how a logged-in user securely ends their session and returns to the login screen.

Use Case ID	UC-2
Use Case Name	Logout
Actor(s)	User
Priority	Medium
Pre-Conditions	User is logged in.
Basic Flow	1. User clicks Logout button. 2. System ends session. 3. Login screen is displayed.
Actor Actions / System Response	User clicks Logout, then System logs out and displays login screen.
Alternative Flow	None.

Table 3.2: Logout Use Case

3.5.3 Use Case – Registration

This use case describes how a new user creates an account in the SignFlow Translator App.

Use Case ID	UC-5
Use Case Name	Registration
Actor(s)	User
Priority	High
Pre-Conditions	User is on the Sign Up Screen.
Basic Flow	1. User clicks the "Sign up" link on the Login screen. 2. User enters required details 3. User clicks the "Sign Up" button. 4. System validates the input 5. System creates and stores the new user account in the database. 6. System displays a success message and redirects the user to the Login Screen.
Actor Actions / System Response	User enters details, then System verifies input and creates account then System redirects to Login Screen.
Alternative Flow	If input validation fails, the system displays an appropriate error message and prompts the user to retry.

Table 3.3: Registration Use Case

3.5.4 Use Case – Gesture Recognition

This use case explains how the system captures and interprets user gestures for real-time recognition.

Use Case ID	UC-3
Use Case Name	Gesture Recognition
Actor(s)	User
Priority	High
Pre-Conditions	User is logged in and camera access is allowed.
Basic Flow	1. User performs a hand gesture. 2. Camera captures the gesture. 3. System processes gesture using CNN + MediaPipe.
Actor Actions / System Response	User performs gesture, then System analyzes and recognizes it.
Alternative Flow	If gesture is unclear, the system prompts the user to retry.

Table 3.4: Gesture Recognition Use Case

3.5.5 Use Case – Translate Gesture to Text

This use case describes how recognized gestures are converted into readable text for the user.

Use Case ID	UC-4
Use Case Name	Translate Gesture to Text
Actor(s)	User
Priority	High
Pre-Conditions	Gesture has been recognized successfully.
Basic Flow	1. System maps gesture to word. 2. Word appears as text on screen.
Actor Actions / System Response	User performs gesture, then System displays corresponding text.
Alternative Flow	If mapping fails, system shows “Unrecognized gesture.”

Table 3.5: Translate Gesture to Text Use Case

3.6 Summary

The chapter stipulated the SignFlow Translator App requirements. It is created to support real-time translation by modularly integrating CNN-based gesture recognition and text generation. The interface and functional requirements guarantee a successful functioning between Android and iOS platforms, whereas non-functional requirements can be seen in performance, reliability, and security. The feasibility study verifies that the system is viable, scaled and effective, creating a good basis to work on.

Chapter 4

Design

The system design entails identification of how our application SignFlow Translator will be developed in order to satisfy all our requirements. This chapter states the design method that would be applied in the construction of user needs into an effective application structure. We shall explain the system architecture in general, highlight all the main components and modules, and demonstrate the data flow among them. This structural step is important towards improving the integrity, effectiveness, and having the final project effectively address the communication issue.

4.1 System Architecture

This section is an analysis of the SignFlow Translator application into its core operating modules, how they are interconnected with each other, and how they work. We will begin by giving a general summary and then move to the details of every component. The general architecture is a classic client server with the mobile front-end, responsible to interface with the user, and the backend is used to process the main translation code.

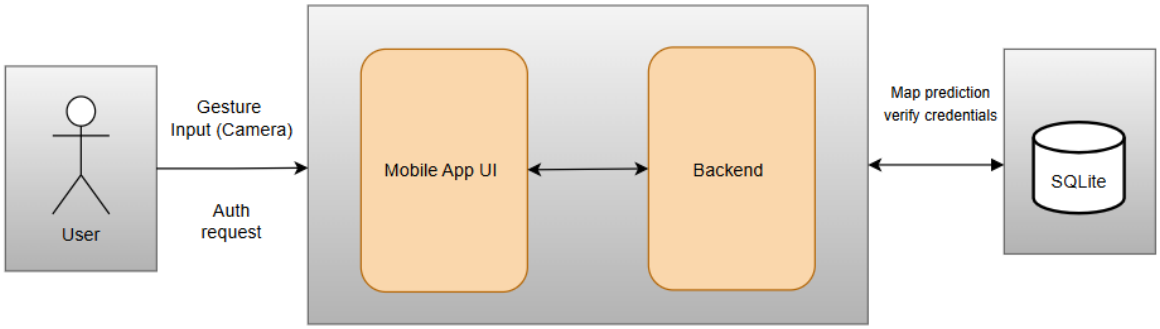

Figure 4.1: High level architecture diagram

User triggers interactions and gives Auth Requests to the Mobile App. UI to the AI/Recognition Logic - login/log out and Gesture Input (using the camera feed). The Interface layer which houses both the UI and the AI/Recognition Logic (which does the core translation) is connected to the BackEnd/Storage layer. The layer is based on local SQLite DB, in order to store essential, non-historical information persistently. UI read/written data / Settings, and the Model and Config Data. (reference dictionary) read by the AI logic to accomplish the final Recognized Text Output that is displayed again on the UI. There are three major tiers in system diagram :

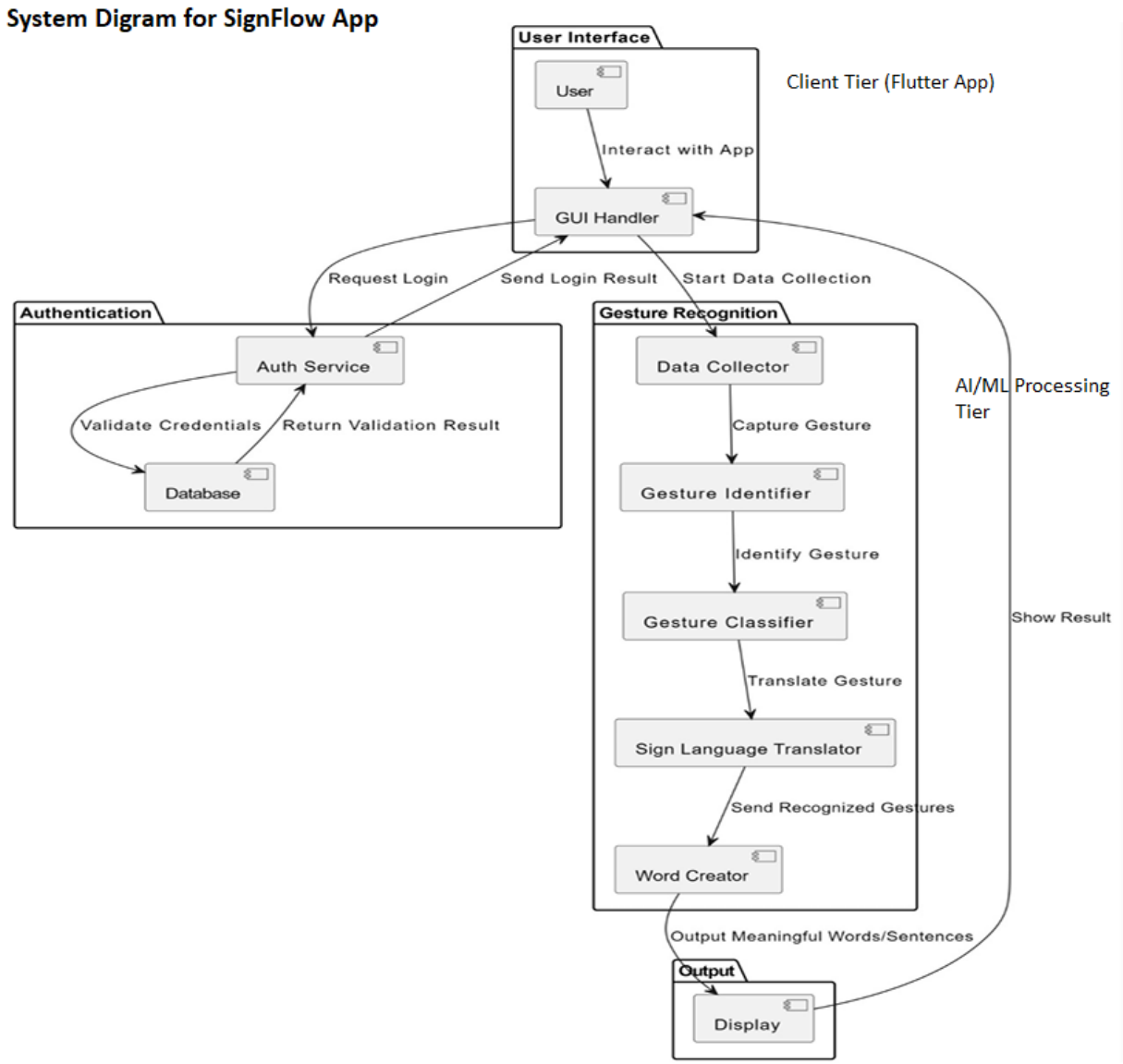

Figure 4.2: Low level architecture diagram

4.1.1 Client Tier

It is the interface of the user and it is based on Flutter/Dart. It deals with capturing of the live sign input with the help of the mobile camera. It handles any additional authentication needed by users and initiates the connection with the use of https to transmit the video data captured by it to the Django backend. When the sign prediction is prepared by using EfficientGCN-B0 model, it takes the text that has been translated and transforms it to display the result.

4.1.2 Application Tier

This is the heart of the system, which is written in Python in Django and Django Rest Framework. It serves as the focal check point router and security. It processes all user sessions and records the history of translation in the database and directs the data on the special sequence of keypoints to the central EfficientGCN-B0 model of processing the sign. It guarantees that the client supplied data, in this case, 65 keypoints per frame generated by MediaPipe, is verified and safe prior to hitting the deep learning model which is very expensive to implement.

4.1.3 Data Tier

This level will store all the application information in a permanent and secured fashion and will mostly use SQlite database with Django layer. Major information held is the User Profiles that are used on authentication and the 20-sign Vocabulary Mapping. The database is also guaranteed and secured in dataintegrity, speed and is accessed via the secure Backend Tier.

4.2 Design Constraints

SignFlow Translator software design process was scheduled in a sequence of the clearly defined steps. The application consists of two significant components, the mobile end, and the API which were developed in isolation taking into consideration the ease of use to the user. To ensure quality, we made important Design Constraints as indicated in Table 4.1.

Constraint	Description
Security	All user data, especially login credentials stored in SQLite, must be protected. Connection between the backend and the mobile application should be secured.
Criticality of The Application	The application should respond and communicate the data immediately.
Camera Quality	The accuracy of sign recognition is inherently constrained by the user's camera quality and ambient lighting conditions.
Internet Connection	There must be a good internet connection for successful data transfer.

Table 4.1: Additional System Constraints

4.3 Design Methodology

The SignFlow translator project will be based on the Agile methodology. Agile model is a project management strategy that is particularly deployed in the development of

sophisticated software. This structure enables development cycles to be done befriend, thereby playing a critical role in reducing the number of errors and mistakes that usually face the development process. Figure represents the model and the project is broken down into a number of short-time boxes which are allocated to the professionals involved in the project team. It is the team approach that enables a quick response to the rapid changes and is intricate to indicate adjustment to change in requirements during the lifetime of the mobile application development.

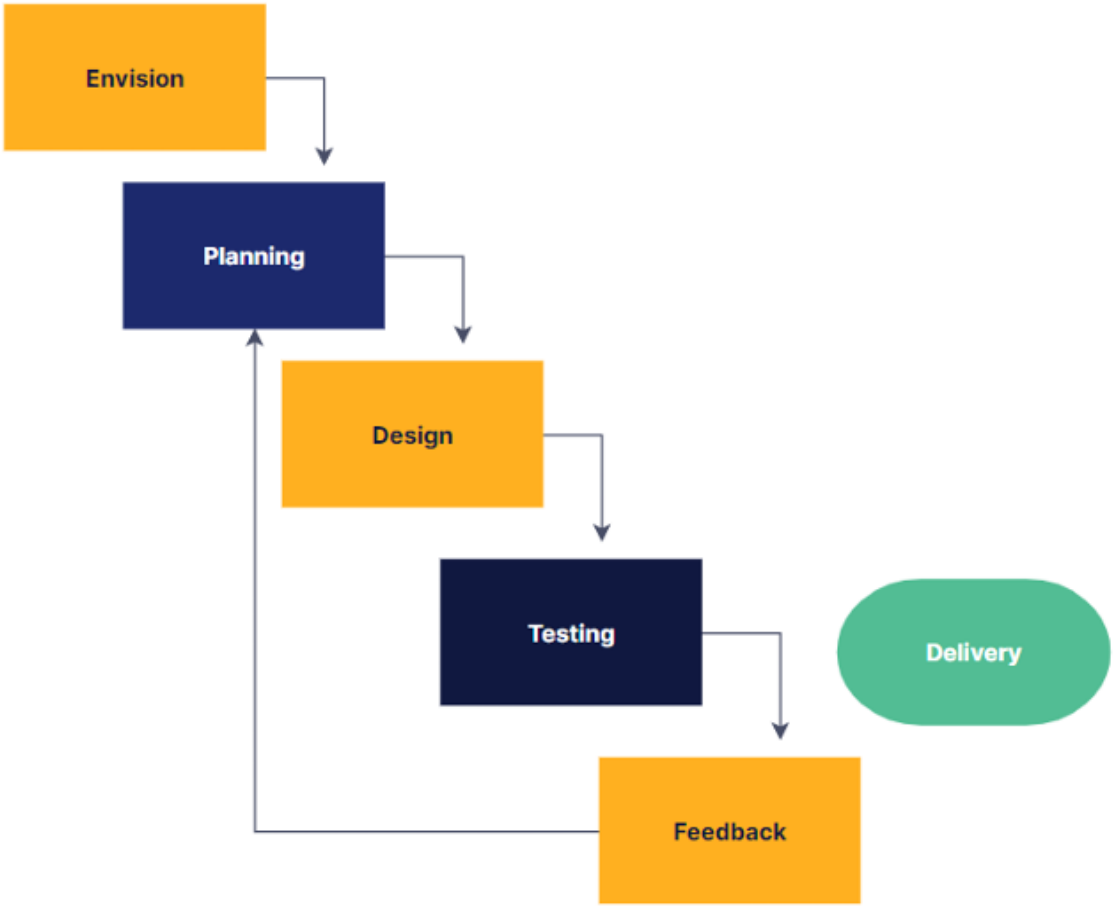

Figure 4.3: Agile process diagram

4.4 High Level Design

This chapter makes the fundamental design and dynamics of the. SignFlow Translator App. We are incorporating renowned diagrams to depict the manner in which the components associate and the movement of data.

4.4.1 Component Diagram

Some of the fundamental aspects of our application will include User Authentication modules, video capture and detector as well as prediction and connection to the vocabulary of PSL signs which will be handled and stored in the SQLite database in the Django backend.

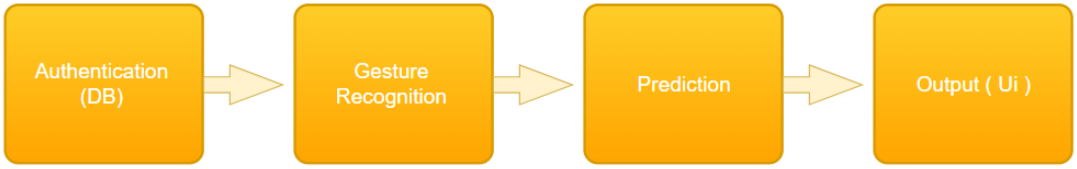

Figure 4.4: Component diagram

4.4.2 Data Flow Diagram (DFD)

The Data Flow Diagram provides a graphic representation of the flow of data in and out of the application components of SignFlow Translator. This starts when the user enters a sign. Such video is uploaded to backend. MediaPipe processes this video with the Key point Extraction Component that extracts 65 key points per frame. This Key point Sequence is then forwarded to the Translation Model Component, which applies EfficientGCN-B0 model and Vocabulary Data to come up with the Predicted Sign Text. Lastly, the translated work is saved as the initial user and registered in the Translation History through the Django backend.

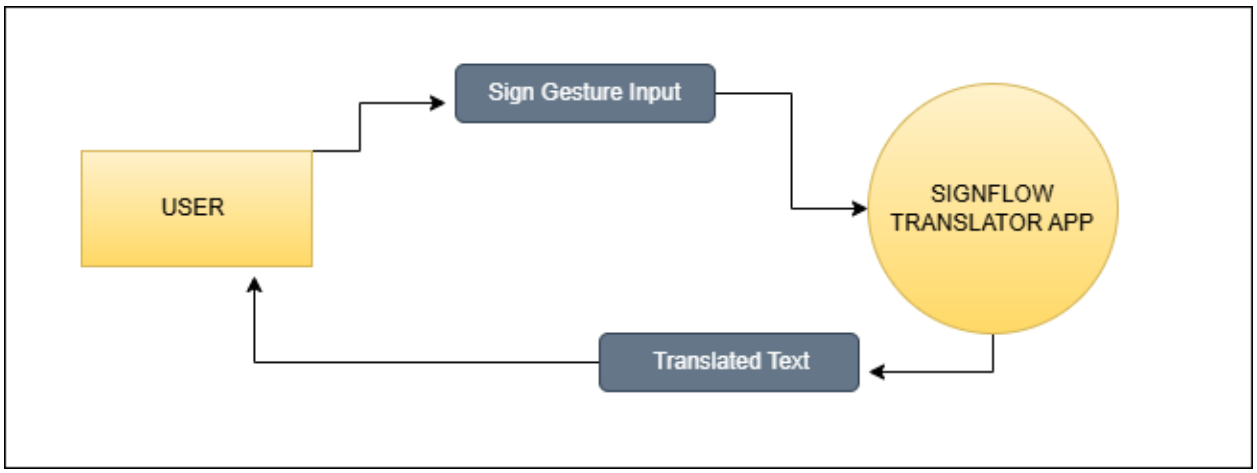

Figure 4.5: DFD Level 0

The Level 0 DFD 4.5 defines the boundary between the User and the SignFlow Translator System. The primary inputs are Gestures (live feed) and Authentication Details. The primary outputs are Translated Text/Speech.

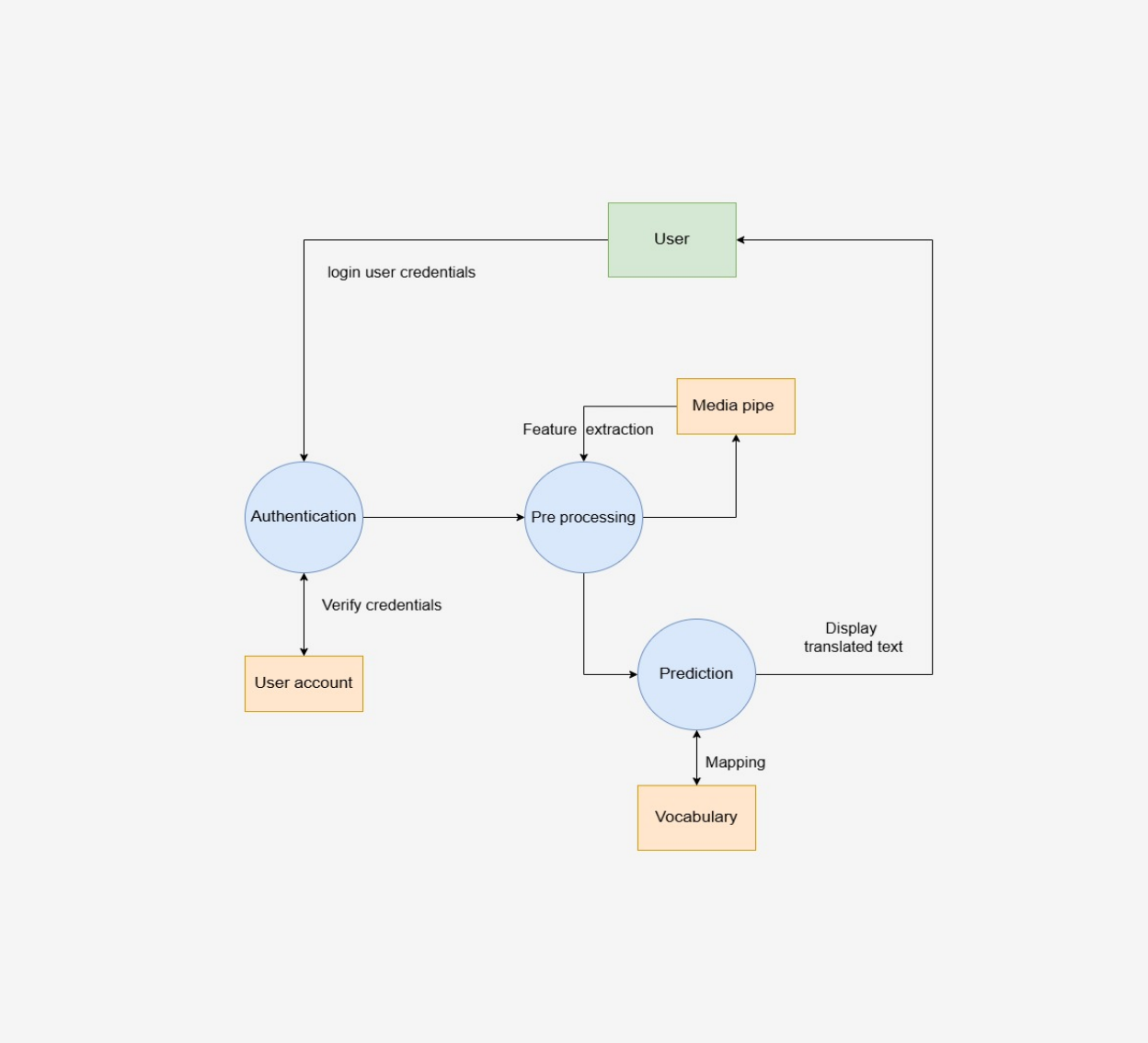

Figure 4.6: DFD Level 1

The Level 1 DFD 4.6 breaks the system down into its main functional processes, showing how data flows between them:

1. Input & Authentication Process: Management of the video input of a user and its capture.

2. AI Recognition Process:Data input in the form of a video, landmark identification algorithm, executes the Graph Convolutional Network to classify signs and sends the raw signs along.

3. Meaningful Words: The model output is mapped, and the meaningful words get transferred to frontend.

4.4.3 Sequence Diagram

A sequence diagram just shows how the application components interact with one another in a chronological manner; that is, the chronology of the interaction. There are various kinds of interactions that would be conducted through SignFlow translator application. The PSL Signer will be interpreting the system to carry out the essence of translation. This will give a close up view of the way the user will use the application and what will be returned by our system.

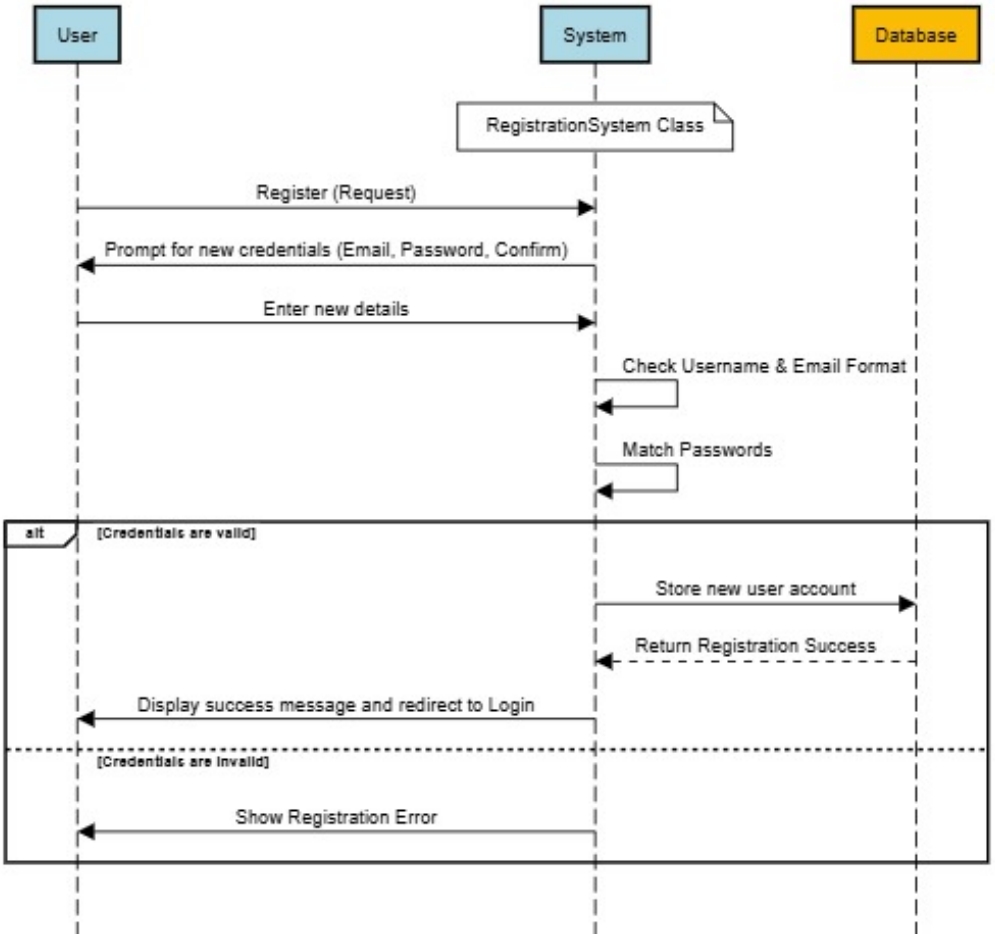

Figure 4.7: Sequence Diagram for User Registration

Figure 4.7 shows that new users enter their details, which the system verifies before saving the account and redirecting them to the login screen.

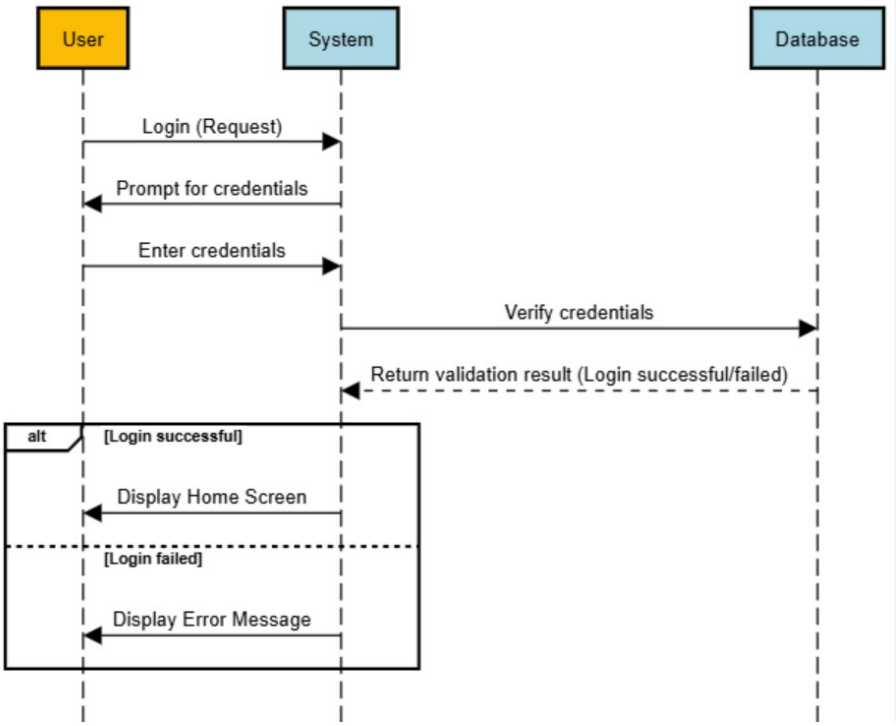

Figure 4.8: Sequence Diagram for User Login

Figure 4.8 illustrates the login process where the system checks the user's credentials against the database to either grant access or show an error.

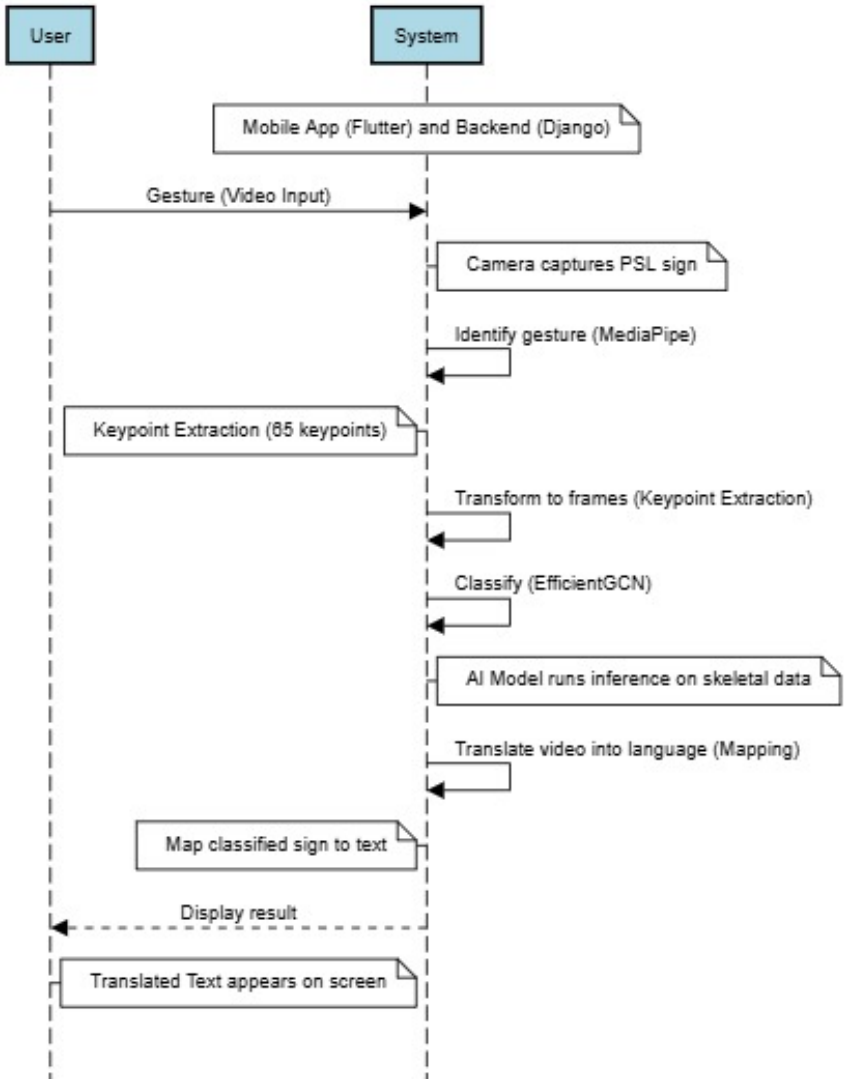

Figure 4.9: Sequence Diagram for PSL Sign Detection

Figure 4.9 demonstrates how the system captures gestures via camera, processes them using AI models, and displays the translated text on the screen.

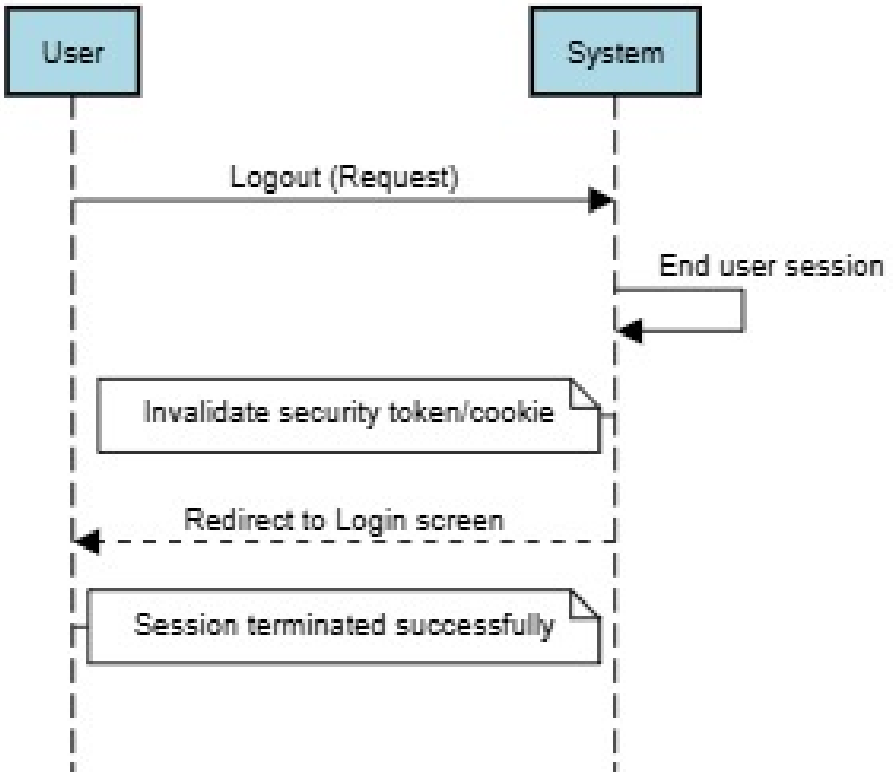

Figure 4.10: Sequence Diagram for User Logout

Figure 4.10 shows that when a user logs out, the system ends the session and sends them back to the Login screen.

4.5 Low Level Design

This part will contain a low level design of our SignFlow Translator application. Here, low level design descriptions that provide a direct creation of underlying software modules are provided. Now we are going to break each of the parts into respective underlying parts or modules. In Figure, one can observe the specifics of the Translation Module. A video stream is inputted by the user and passed through the media pipe Key point Extractor. The resulting data stream is then pumped into EfficientGCN-B0 Model to be predicted. The resulting text is next sent to the Translation History Logger after being predicted before being returned to the user interface to print. Figure will describe User Authentication Module and other classes and relationships in it.

4.5.1 Deployment Diagram

The deployment diagram shows the physical deployment of SignFlow Translator components on the physical nodes. The Client Tier is implemented on mobile devices. EfficientGCN-B0 model and Rebel Tier (Backend Tier) is run on Local host, as well

as the Data Tier. These two environments interact with each other in an authenticated way via API endpoints so as to provide real time sign translation capability.

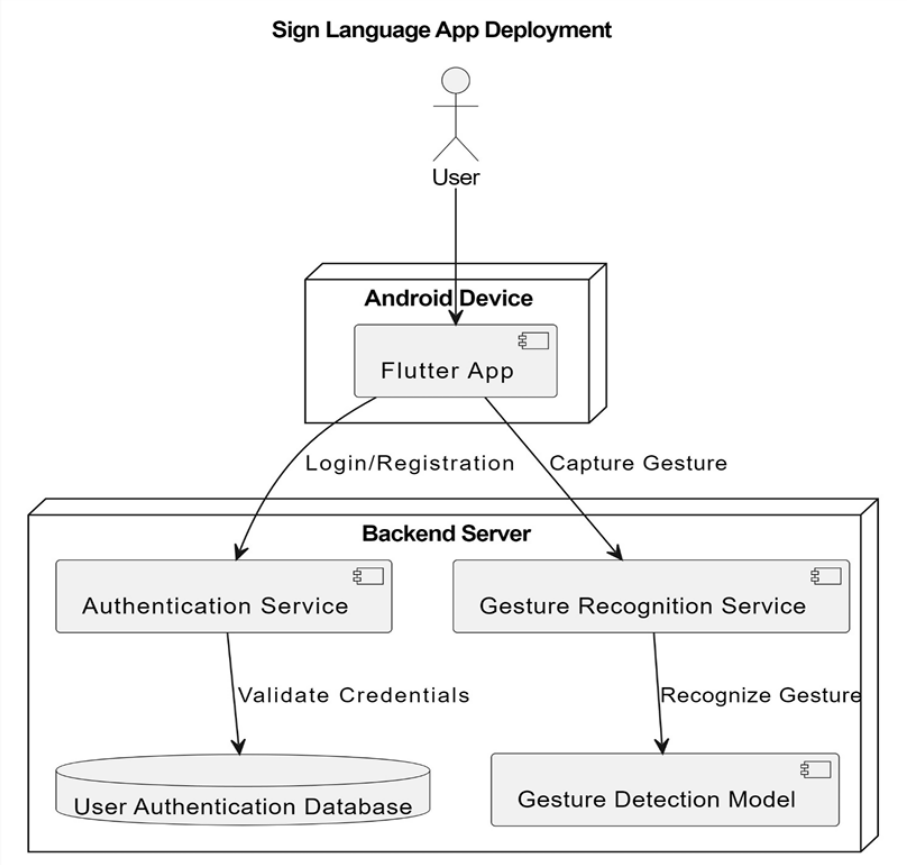

Figure 4.11: Deployment diagram

4.5.2 UML Class Diagram (Logical View)

The UML Class Diagram forms the software blueprint on which the development of SignFlow Translator will be developed. It specifies the key modules of the system with the attributes and the methods they have. This perspective is crucial in the definition of the contracts between the Python classes and those offered by the Dart classes that coordinate the flow of the keypoint information.

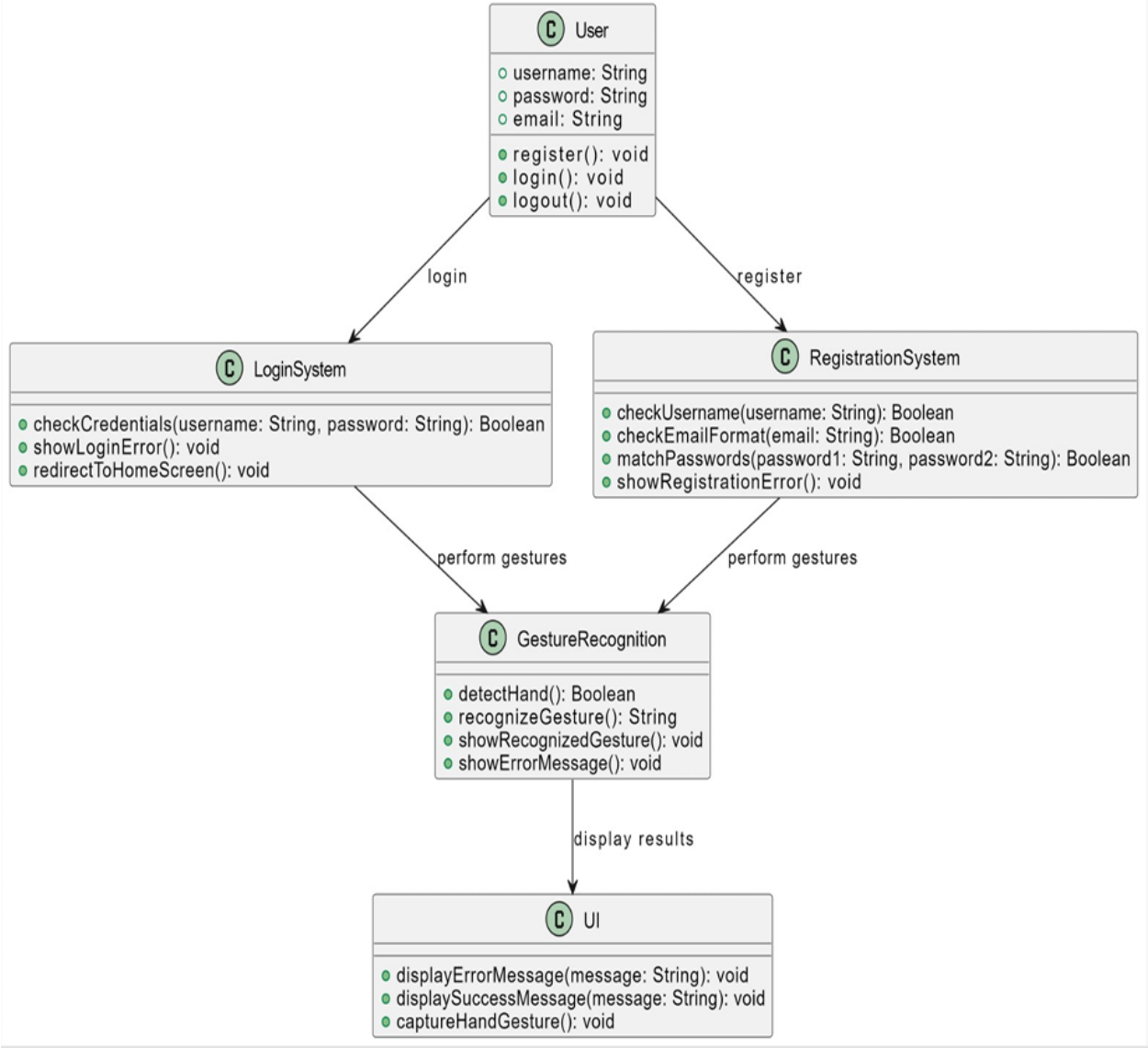

Figure 4.12: UML Class diagram

4.6 Model Architecture and Feature Extraction

This section details the core deep learning components responsible for the actual sign language translation, specifically, the EfficientGCN-B0 model, the feature extraction process, and how they combine to achieve real-time gesture detection.

4.6.1 EfficientGCN-B0

The main machine learning element of the SignFlow Translator is EfficientGCN-B0. It is a variant of Graph Convolutional Networks (GCNs) of neural networks that take non-Euclidean data structures, including the human body pose (skeleton keypoints), as a graph

[9].

The main benefit of EfficientGCN is that it is both efficient and fast, which is why it will be applicable to real-time implementation, such as a mobile translator. The B0 design is the leanest, most computationally-sparse model in the EfficientGCN family, which fits your requirements of processing speed on a typical server setup [10].

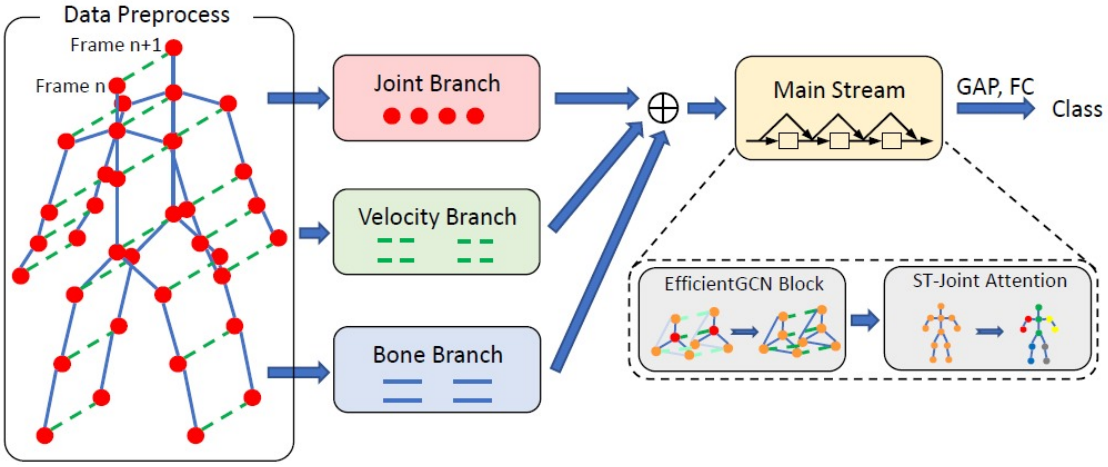

Figure 4.13: Model architecture

The architecture is highly specialized for skeletal data, using a spatio-temporal approach:

- Spatio-Temporal Graph Convolution: Unlike standard convolutional networks that process pixels in a grid, EfficientGCN processes a graph where nodes are body joints and edges represent bone connections. This allows the model to learn spatial relationships and temporal relationships [11].
- Multi-Stream Input: As confirmed by our configuration, the model utilizes a multi-stream approach for robust sign recognition. It processes three distinct data streams simultaneously:
  1. Joint Data: The raw 3D coordinates (x, y, z) of the 65 keypoints.
  2. Bone Data: Derived features representing the directional vectors between connected joints. Using both streams significantly improves accuracy by providing the model with both absolute position and dynamic movement change.
  3. Velocity Data: This tracks the speed of the movements. It measures how much the keypoints shift from one frame to the next, helping the model distinguish between fast and slow signs.
- Architecture Components: The model structure includes optimized blocks with techniques like Channel Fusion at level 2, Graph Attention Mechanisms, and efficient

activation functions to maximize performance while minimizing computational cost. The entire model is designed to accept a sequence of 64 frames containing the 65 joints.

4.6.2 Video to Text Model Architecture

The video-to-text workflow can be technically described as a very simple, yet, very effective pipeline: Video input is kept to generate MediaPipe Keypoints which are then inputted into the EfficientGCN-B0 model to generate the final Text output.

- The model output is a vector of probabilities corresponding to the 20 sign classes.
- The final output is the text associated with the class having the highest confidence score.

4.6.3 Real-Time Feature Extraction Architecture

Feature extraction is handled by MediaPipe and custom Python preprocessing scripts on the backend:

1. MediaPipe Execution: This library runs on the incoming video frames to locate 65 keypoints on the person signing.
2. Normalization: The raw X, Y, Z coordinates must be normalized to ensure the model is invariant to where the person stands in the frame and how far they are from the camera. This step ensures that a small sign close-up is interpreted the same as a large sign farther away.
3. Bone Feature Calculation: The normalized joint coordinates are used to calculate the directional vectors between connected joints.
4. Multi-Stream Assembly: Both the normalized Joint and calculated Bone data are temporally segmented into 64-frame chunks and assembled into the exact format required by the EfficientGCN-B0 input layer. This final, optimized feature set is what the model consumes for prediction.

4.6.4 Real-Time Gesture Detection Flow

The real-time detection process involves tight integration between the mobile client and the backend:

1. Capture Start: The user presses "Start Recording" on the Flutter Client.

2. Video Stream Processing: The mobile camera captures the PSL sign. While the user is signing, the video is being locally processed by MediaPipe to extract the 65 keypoints for the hands, face, and pose in real-time.

3. Data Transmission: Once the sign is complete, the resulting sequential array of X, Y, Z (z coordinate is not used because its noisy) coordinates and the calculated "Bone" features are compiled. The DetectionController uploads this data package to the Django Backend via the /predict/ endpoint.

4. Backend Inference: The Django Backend Tier receives the data and feeds it directly into the preloaded EfficientGCN-B0 model for prediction.

5. Result Return: The backend sends the highest-scoring sign label and its confidence back to the mobile client as a JSON response.

6. Display: The Flutter client updates the display with the translated sign and the confidence score.

4.7 GUI Design

The objective of the GUI Design is to make it as maximized as possible in terms of usability and access by the hearing-impaired user since it is the primary communication tool. The structure is developed on Flutter. We have established an interface with all the factors taken of usability. We also endeavor to make it attractive and beautiful. The rationale behind coming up with such interface is that, the user should feel good when using our application. Today as much stress is put on appearance and touch of interface. Here we are going to present our GUI.

4.7.1 Login Screen:

A clean interface prompting the user to enter their Email and Password to access the application.

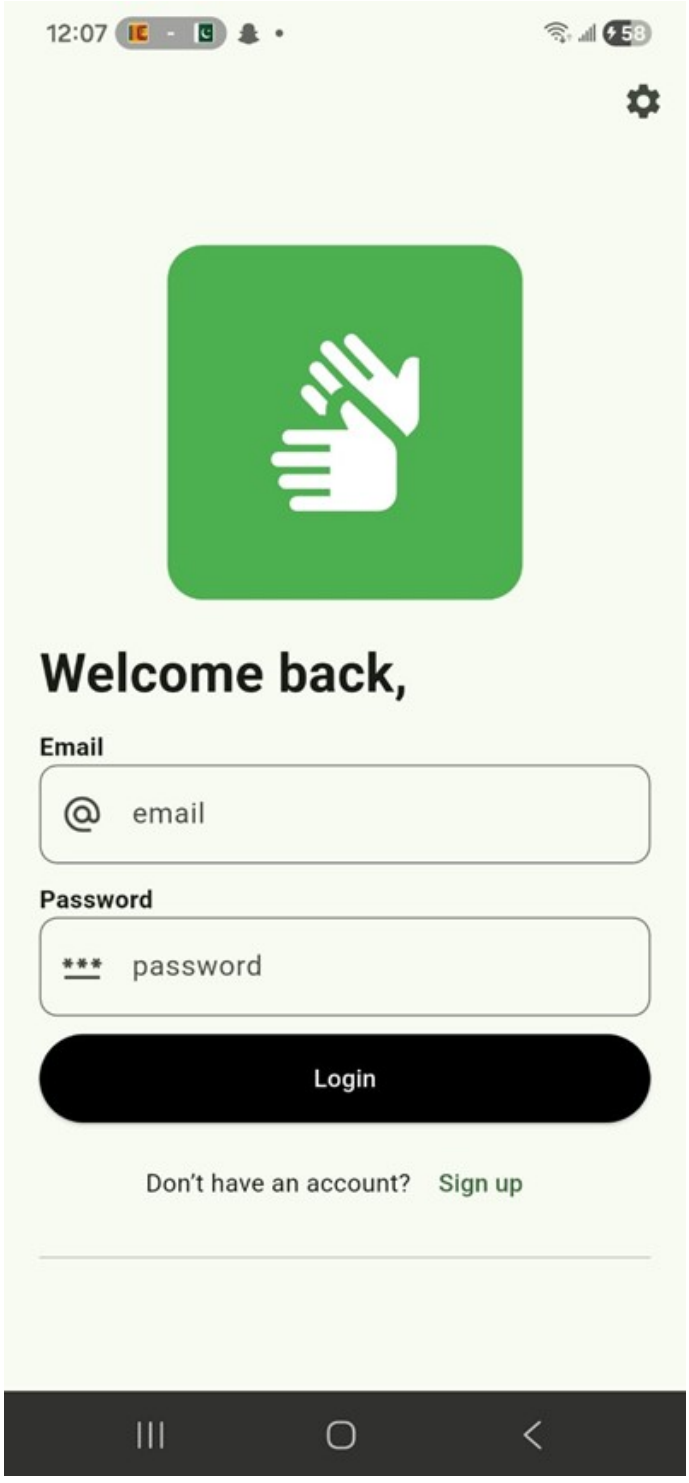

Figure 4.14: Login screen

4.7.2 Sign Up Screen:

The interface where new users can create an account by providing and confirming their email and password.

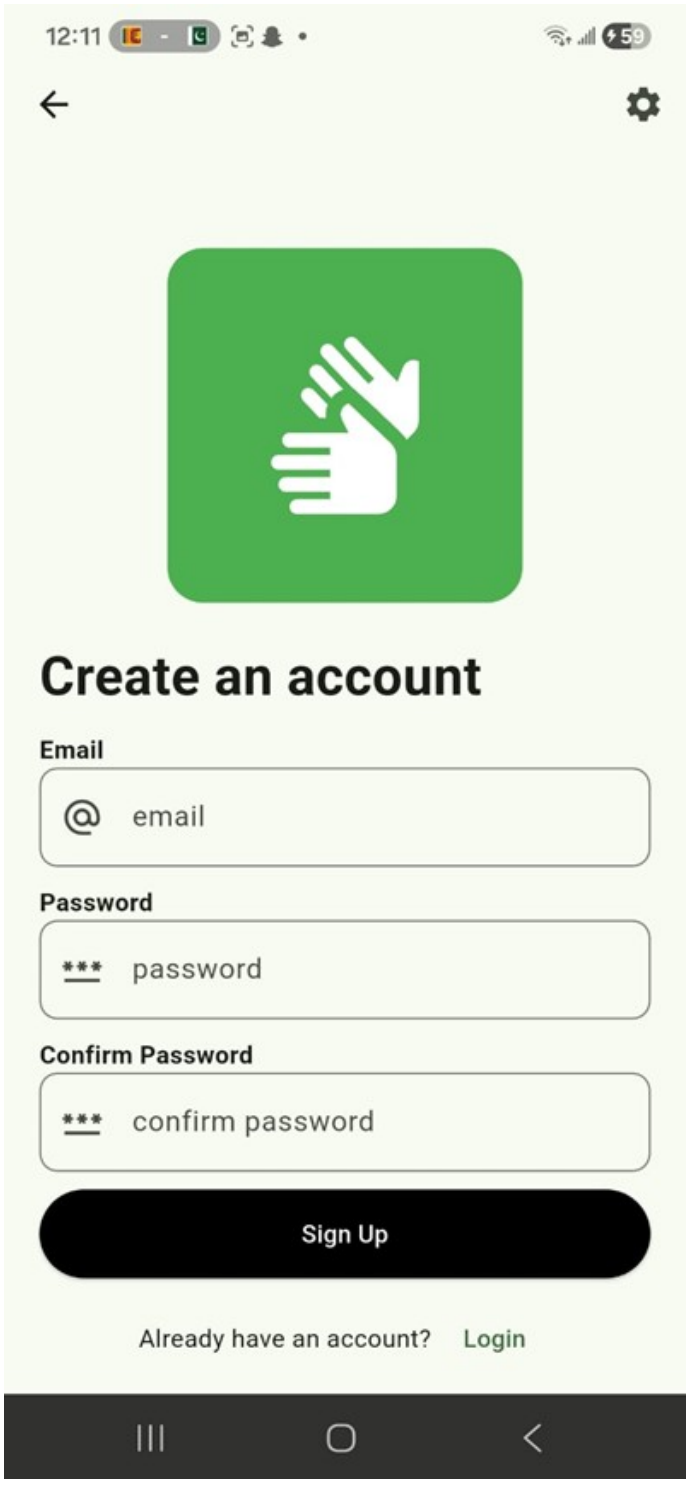

Figure 4.15: Signup screen

4.7.3 Home:

The main landing page displays the SignFlow logo and a large "Start Detection" button to begin translation.

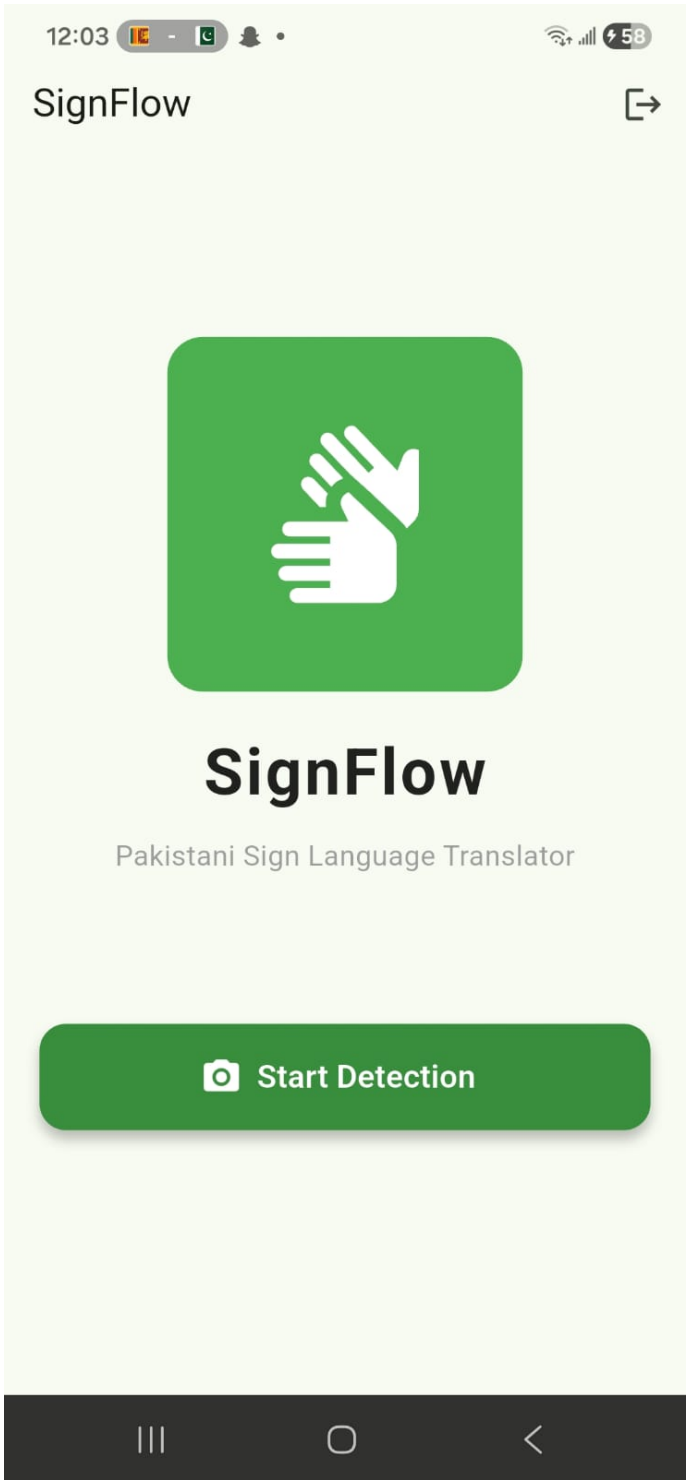

Figure 4.16: Home screen

4.7.4 Detection Screen:

Handles video capture and real-time translation.

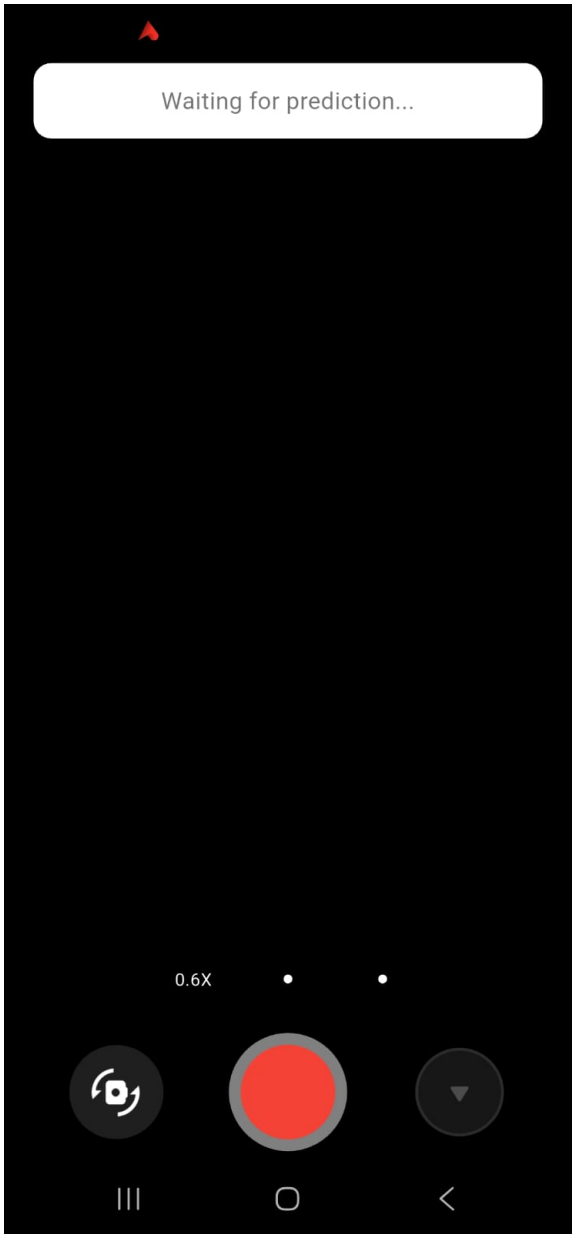

Figure 4.17: Detection screen

4.7.5 Loading Screen:

The Loading Screen ensures a smooth transition and prevents the user from interacting with the UI during background processing.

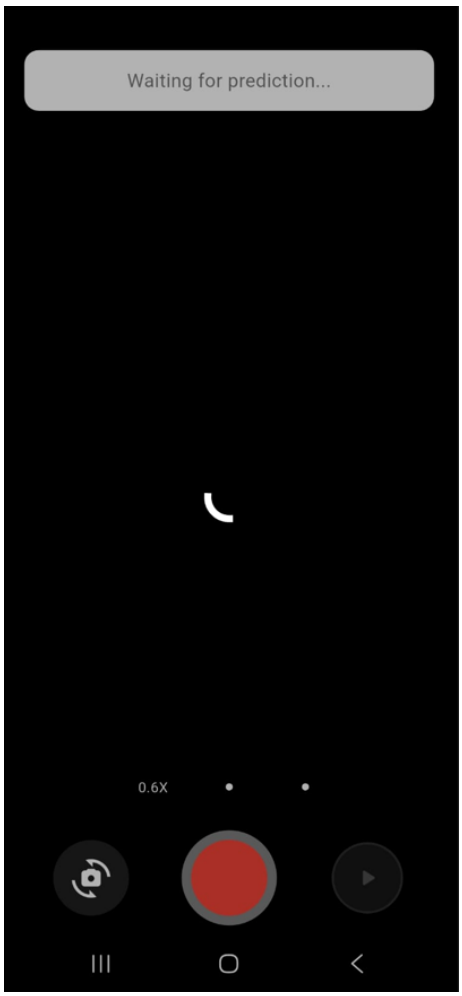

Figure 4.18: Loading screen

4.8 External Interfaces

The SignFlow Translator operates by interacting with several external hardware and software components. These interfaces are critical for data capture, processing, and storage.

4.8.1 Hardware Interfaces

- **Camera Module:** The application interfaces directly with the mobile device's camera hardware to capture a live video stream at a minimum resolution of 720p (30fps). This is the primary input source for the gesture recognition pipeline.

- **Network Interface:** The mobile device utilizes the Wi-Fi or 4G/5G cellular radio to transmit serialized keypoint data to the backend server.

4.8.2 Software Interfaces

- **MediaPipe Framework:** The system interfaces with the Google MediaPipe library to perform real-time extraction of 65 skeleton keypoints (Hand, Pose, and Face landmarks) from the raw video feed.
- **Django REST API:** The frontend communicates with the backend via a RESTful API, sending JSON payloads containing keypoint sequences and receiving translated text responses.
- **SQLite Database:** The backend interfaces with an SQLite database to store user credentials.

4.8.3 Summary

This chapter successfully translated our requirements into a technical design. The three-tier architecture provides the necessary speed and stability. We formalized the structure using DFDs and UML diagrams, ensuring that the MediaPipe and EfficientGCN-b0, is correctly integrated into the app.This comprehensive plan is now fully prepared and it has become the road map of the development team in future.

Chapter 5

System Implementation

Implementation referred to as the process of converting the SignFlow Translator design into reality. This stage involves all the elements integration and implementing the system to achieve the correct translation which is real-time to run our 20-sign Pakistani Sign Language vocabulary.

5.1 System Architecture

SignFlow App is designed on the basis of 3-tier architecture. Imagine it as a three-layered construction with every layer having a unique task. With this strategy, things are well arranged and it becomes easier to repair or upgrade certain elements of the app in the future without disrupting all the other parts.

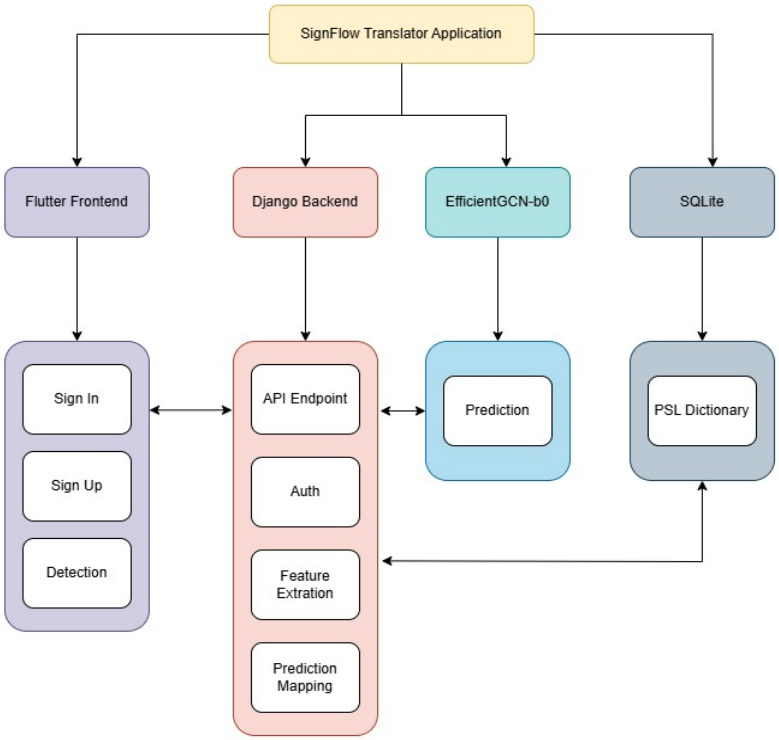

Figure 5.1: System Architecture of SignFlow

1. The App Frontend: It is a part of an app that you might look at and operate. We built it with Flutter. It will display all screens and the buttons, will capture a picture of your hand signs with the cameras of your phone will relay the information to our server.

2. The Backend: It is the primary part of our app, which is founded on Django. It is the backend that is doing the heavy lifting when the data is transmitted to the backend via the app. It text messages what you are texting, translates it and sends it back to your phone.

3. The Database: Here we were required to store the information like user accounts, and previous translations. For this, we used SQLite. It is quite simple and an efficient database that is compatible with our Django backend.

5.2 Tools and Technologies

Various tools and technologies have been applied to create the SignFlow Translator App. Table 5.1 lists the main ones that we used.

Type	Name	Rationale	Version
Tools	Visual Studio Code	Primary IDE for both Flutter and Django	1.85
	Android Studio	For Android SDK, emulator, and build tools	2023.1.1
	Git & GitHub	Version control and collaborative development	2.39.1
	LaTeX	For project documentation and report writing	MiKTeX 2.9
Tech.	Flutter	Cross-platform mobile UI for Android & iOS	3.16
	Dart	Language for Flutter development	3.2.3
	Django	Backend web framework for server-side logic	4.2
	Python	Language for Django and machine learning	3.10
	SQLite	Lightweight, file-based database	3.39.4
	EfficientGCN-B0	Lightweight version of ResGCN model	B0
	MediaPipe	Feature extractions from video	0.10.21

Table 5.1: Tools and Technologies Table

5.2.1 Explanation of Major Tools

- Visual Studio Code: It was our main code editor. We have already experienced it when writing the code of the Flutter app or the Django backend because it is simple and has numerous convenient functions. Android Studio: Android Studio: Windows VS Code is our main operating tool; nevertheless, we had to use Android Studio. It helped us deal with all the toolkit to develop the Android version of our application and be able to test it on a computer using a virtual phone.

- Git & GitHub: In order to protect our code and structure it, we used Git. It is one of the systems of recording what we change. We do host our code in GitHub, and it is convenient to share with one another and have a backup.

- LaTeX: In order to produce this report, we have utilized LaTeX. It also helps in making documents look professional and clean enough and that is very good when one has to operate with university projects.

- Flutter & Dart: Flutter is the application on the mobile platform that we selected because the mobile app enables to write the code once and run the code on the Android and iPhones. Dart is the syntax that is to be used with Flutter.
- SQLite: SQLite was our database type as it is not complicated and it does not presuppose numerous setups. It belongs to Django and this was the best choice as far as our project was concerned. • Django Rest Framework (DRF): To make sure that our flutter application was in the position to interact with our Django database, we implemented Django Rest Framework. It helped in the creation of the API that is similar to a messenger that sends out requests to the server and responses to the app.
- EfficientGCN-B0: The reduced version of Graph Convolutional Network that implements the single sign classification and translation properly.
- MediaPipe: The application that is utilized to accomplish the extraction of features x 65 keypoints in the real-time in the video frames in order to engineer features.

5.3 Model Performance and Validation

The implemented GRU model has a high level of predictive accuracy. Evaluation of performance came in the form of conventionally aided statistical evaluation:

- Root Mean Square Error (RMSE): 0.08804
- Accuracy: 91.92 percent

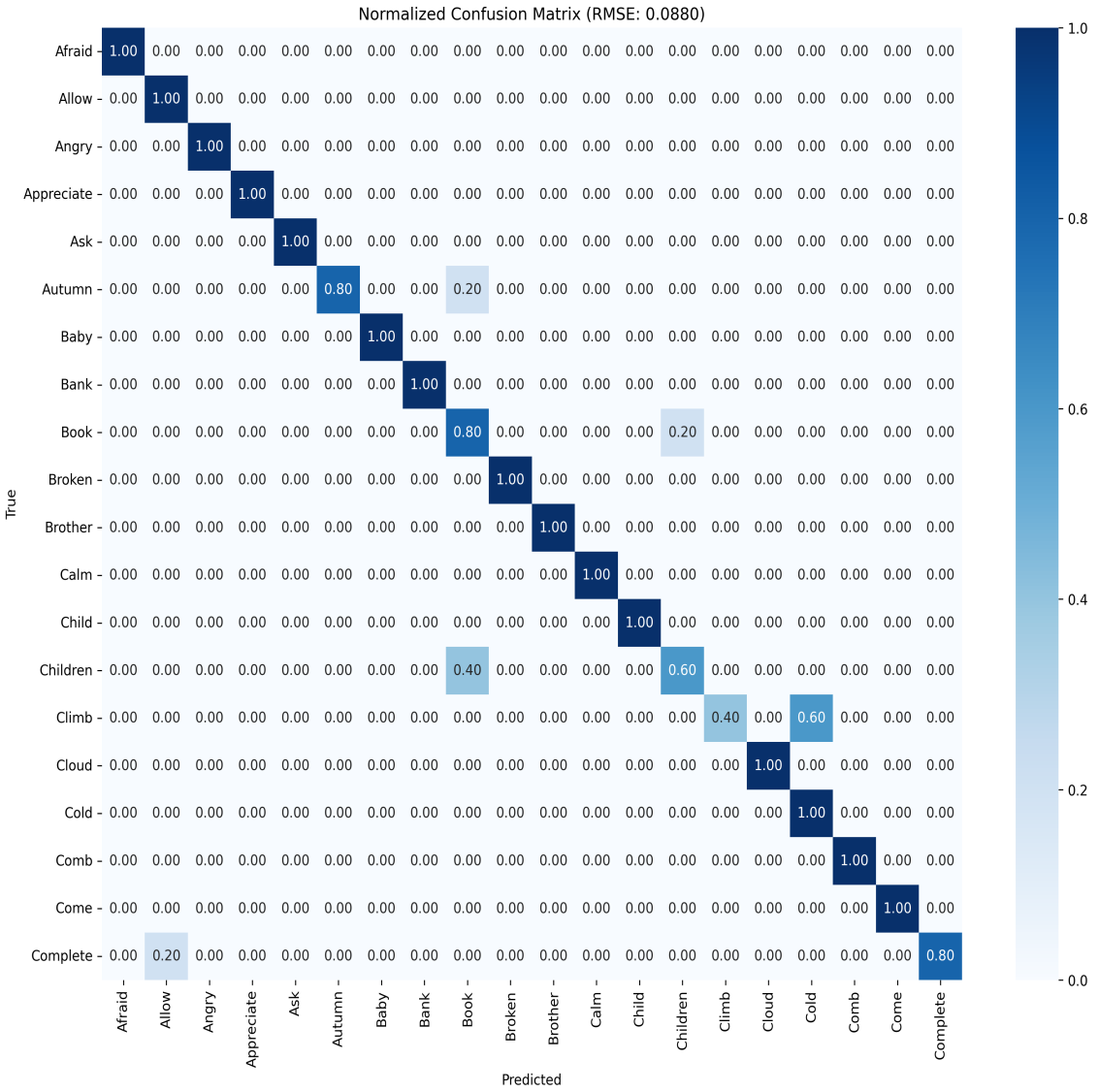

Figure 5.2: Confusion Matrix of 20 PSL signs

High reliability and accuracy in the identification of 20 different PSL signs can be seen in such results and they were checked on real data in the test as displayed in the normalized confusion matrix.

5.4 Security

The safety of the application and the information of our users was a major priority. There are some important aspects that we prioritized on our backend in security:

- User Logins: We used the default system that is available with Django. It is rather safe and manages all the user accounts and passwords on our behalf. It also auto hashes (randomizes) passwords therefore no body can view the passwords.

- Secure API: To make sure that only our app will be only able to communicate with our server we used tokens. In order to log in, the app provides the user with a secret special token. Each time the application needs an item of the server to which it submits this token to prove to the server that it is an authentic user.
- CSRF Protection: Django has also introduced protection system with respect to one of the most common attacks referred to as CSRF. This was enabled to see to it that the app was that much safer.

5.5 Summary

That is why in this chapter we discussed the way it was really done to build the SignFlow Translator App. Our 3-layer strategy with Flutter (app) and Django (server) as the workhors and SQLite (data store) was a presentation of our application. These tools contributed to the creation of a strong and properly functioning and secure app. Once we assembled all these, we could transform our plan into an actual application which is now ready to take the subsequent steps such as testing.

Chapter 6

System Testing and Evaluation

Having developed our SignFlow Translator App, the next step important was to test it accordingly. Testing also assisted us in ensuring that the entire work we did including the mobile interface and the AI-assisted translation accounted to our expectations. System testing implies not only the hardware but also the software to run as a whole system to verify that the app will carry out all the functions it was created to do. It is also highly important to test the software before any release to ensure that any bugs or other issues are identified and addressed to ensure that the software does not have any impact on its performance or even its user-friendliness. Our approach was simple and efficient: Code, Implement, and Test. At the end of the process (after coding a module, such as the login, or camera) we tested it individually. After it was capable of working on its own, we added other modules and ran the system through more testing. We conducted different forms of tests to ensure that our app is reliable and user-friendly:

- Graphical User Interface (GUI) Testing
- Usability Testing
- Software Performance Testing
- Compatibility Testing
- Exception Handling
- Load Testing
- Security Testing
- Installation Testing

6.1 Graphical User Interface (GUI) Testing

Here, the GUI testing was used to test functionality on the application buttons, forms and screens. As our app is created with the help of Flutter, we checked the responsiveness of various elements of the interface such as buttons, tabs, and icons to actions of the user. We were keen to ensure that nothing was messy, nothing worked well and was not difficult to handle.

Test Case ID	Test Case 1
Description	Tests the graphical user interface of the mobile app.
Initial Condition	App installed successfully, permissions granted.
Steps	1. Open the app. 2. Check if all tabs work properly. 3. Verify that buttons and links are responsive. 4. Check that icons and labels appear properly. 5. Ensure the app layout is consistent on all screens.
Result	PASS

Table 6.1: General GUI Test Case

Test Case ID	Test Case 2
Description	Tests input fields and forms (Login & Signup).
Initial Condition	Backend connected, app running.
Steps	1. Open the login page. 2. Enter valid and invalid credentials. 3. Check that error messages appear for invalid inputs. 4. Make sure password fields hide text. 5. Navigate to Sign up and verify smooth transitions.
Result	PASS

Table 6.2: Form GUI Test Case

6.2 Usability Testing

Usability testing allowed to realize how comfortable and easy users interaction with our app can be. Our target audience consists majorly of the hearing impaired users and therefore we concentrated on simplicity, clarity and accessibility. We listened to the navigation of the new users through the application, the speed of the translation that could be carried out, as well as the justification of the icons used and the guidelines provided to the user.

Test Case ID	Test Case 3
Description	Tests user experience and ease of navigation.
Initial Condition	App installed and camera working.
Steps	1. Launch the app. 2. Try using all main features (login, signup, translation). 3. Check how easily users find each feature. 4. Ask for user feedback on clarity and comfort. 5. Measure how long it takes to complete one full translation.
Result	PASS
Observation	Most users found the app easy to use and visually pleasant.

Table 6.3: Usability Test Case

6.3 Software Performance Testing

The performance testing also assisted us in testing the speed and stability of the app, particularly when it is translating signs in real time. The time in which it takes a sign to be noticed, decoded and put up as a text or speech was measured.

Test Case ID	Test Case 4
Description	Tests app speed, stability, and responsiveness.
Initial Condition	Backend server active with AI modules loaded.
Steps	1. Open the camera and start a live translation session. 2. Note how fast the translation appears. 3. Test on both Wi-Fi and mobile data. 4. Try running multiple translations back-to-back. 5. Observe any app lag or delay.
Result	PASS (Average translation delay: 2 seconds)

Table 6.4: Performance Test Case

6.4 Compatibility Testing

This part will be an overview of the devices being tested and the performance of the app in various hardware setups.

- Devices Tested: Pixel 6, Samsung S 23
- Result: PASS, The app performed well and looked consistent across all devices.

6.5 Exception Handling

This section would describe the typical exceptions that are experienced in the application, causes, and solutions that are provided to manage them gracefully.

Exception	Cause	Solution
Duplicate Registration	User tried to register with an existing email.	Showed message: “This email is already registered. Please log in.”
Camera Permission Denied	User didn’t allow camera access.	App showed prompt to enable camera permission.
No Internet Connection	Server unreachable.	Displayed retry dialog with “Check your internet connection.”
Invalid Form Input	Missing or incorrect data entered.	Shown inline error highlighting the missing field.

Table 6.5: Exception Handling Test Cases

6.6 Load Testing

- The system supported 5 simultaneous users without slowing down or crash of the server.
- Mean response time: 1.7 seconds
- Server CPU usage: 72%
- Result: PASS

6.7 Security Testing

This section describes the security tests that were done to confirm authentication, data protection, and session management mechanisms.

Test Case ID	Test Case 6
Description	Tests authentication and data protection features.
Steps	1. Login with valid credentials. 2. Try accessing data after logout. 3. Attempt login with wrong password. 4. Check if passwords are encrypted in the database. 5. Verify token expiry after session timeout.
Result	PASS, All security checks passed successfully.

Table 6.6: Security Test Case

6.8 Installation Testing

In this section, the results of the installation testing are provided, and they have confirmed that the app has been installed successfully, demands the required permissions, and opens properly.

Test Case ID	Test Case 7
Description	Tests app installation and first launch.
Steps	1. Install the app on a new device. 2. Launch the app. 3. Check if it asks for camera and microphone permissions. 4. Verify that all screens load properly.
Result	PASS, The app installed perfectly and launched without any issues.

Table 6.7: Installation Test Case

6.9 Summary

The SignFlow Translator App was based largely on testing to stabilize and make the application user-ready. We did test images to the back-end speed and end-user security. All tests proved that the app works effectively and is convenient to work with. The Flutter frontend and Django back-end as well as the AI-driven gesture recognition were all integrated into each other. Our app successfully passes all the major test cases, is fully functional, and it is now secure and ready to be used in real-life.

Chapter 7

Conclusions

SignFlow Translator App can be used to transform the life of individuals with hearing or speaking disabilities in a very significant way. It can act as an intermediary between sign language and non-sign language users by offering real-time experience of signing and text translation provided by applying AI-based gesture recognition. The app also provides a user with the capability to speak out without the human interpreter. By simply using the cameras of the smartphones, one is able to sign and the app immediately translates into text and speech. It is much easier and more inclusive where people interact on a daily basis either in schools, work places, or hospitals or in the open areas. This report began by providing a description of the project, the motivation factor of developing such an application and how the application can help the society. Then we spoke about the literature regarding it, which gave us the base of learning about the sign language translation and its technical context. The entire software development process, including requirement gathering to system design, implementation, and lastly, system testing was also outlined in the document based on SDLC. We have managed to create a mobile cross-platform application based on Flutter in the frontend, Django as the backend, and AI models to detect gestures and translate languages, which are used in this project. The project was done on schedule and reached its primary objective, which was to design a quality and user-friendly mobile solution that helps the deaf to the community using technology.

7.1 Contributions

It was successfully able to meet all technical and non-technical requirements of the project. SignFlow Translator App does not only present a real-life communication issue solved but serves an example of how machine learning, natural language processing, and mobile development can go together in one fully integrated system.

Key contributions include:

- A working AI-based sign recognition and translation system.
- A secure backend for user management and translation history.
- A simple and intuitive mobile interface suitable for all age groups.
- Real-time interaction between sign language users and non-signers.

The project is a basis of future development of the sphere of the sign language technology and accessibility apps.

7.2 Disciplined Project Management

The proper planning, time management, and use of teamwork helped in the successful completion of the SignFlow project. All of the stages, including the research and deployment were taken under a firm regime to make sure we delivered on time and quality. During the project, we have got to understand that there should be an equilibrium between technical work and a clear communication and good scheduling. Wisdom in managing our resources and aligning our goals were used to make us able to deliver the project successfully within the given time.

7.3 Team Communication

The importance of team communication in our project was critical to the success of our project. We made sure that all ideas of team members were recorded and taken seriously. Planning technical issues and having open communication also enabled us to surmount technical difficulties, as well as make improved design choices. Having open and fair and consistent communication also made sure that everyone was an equal contributor and motivated during the process. Teamwork situation enabled us to work with different skills that we possessed and develop a powerful end product.

7.4 Future Work

Although the existing variant of SignFlow Translator App works just fine, there is some space to improve and widen it. We will continue to add features as well as platform support to the app so that it could be even more useful and available in the future.

7.4.1 Other Platforms

The app is currently created on Android and iOS with the help of Flutter. Next time around, we also want to develop an online form or a web version of SignFlow, meaning the person could use the translation service without leaving the browsers and so it would be much more convenient.

7.4.2 Additional Features

In upcoming updates, we plan to include:

- Multi-language translation that allows the signs to be translated into different languages that are spoken
- A voice to sign module whereby verbal communication can be transcribed to animated gestures, known as a sign.
- There is a user-offline option of translation, which provides access to restricted information in terms of access to the internet.
- A sentence generation module for smoother communication between deaf and non-deaf users.

7.5 Summary

To sum up, the SignFlow Translator App illustrates how technological advances can make a positive social difference by making the hearing-impaired community self-sufficient. It combines AI and computer vision with mobile development in a single solution. Teamwork, disciplined management and constant tests enabled us to produce a working system, which is innovative, reliable, and powerful. SignFlow will be one of the applications that can enable communication accessibility through AI in the global market with future additional development and support of additional platforms.

Appendix A

User Manual

This user manual explains how to use the SignFlow Translator App. It provides step-by-step guidance on creating an account, logging in, and using the app for real-time Pakistani Sign Language (PSL) translation. Some basic tips are also included to help users get better results while using the application.

A.1 System Requirements

To use the SignFlow Translator App smoothly, the following requirements should be met:

- **Device:** An Android or iOS smartphone.
- **Camera:** A working phone camera with permission enabled. A resolution of 720p or higher is recommended.
- **Network:** A stable internet connection such as Wi-Fi or mobile data, as predictions are processed on the server.
- **Lighting:** Proper lighting conditions are advised to ensure better gesture recognition.

A.2 Getting Started

A.2.1 Create an Account

To begin using the application, open the SignFlow Translator App and create an account using your email address and password. The Sign Up screen is shown in Figure A.1.

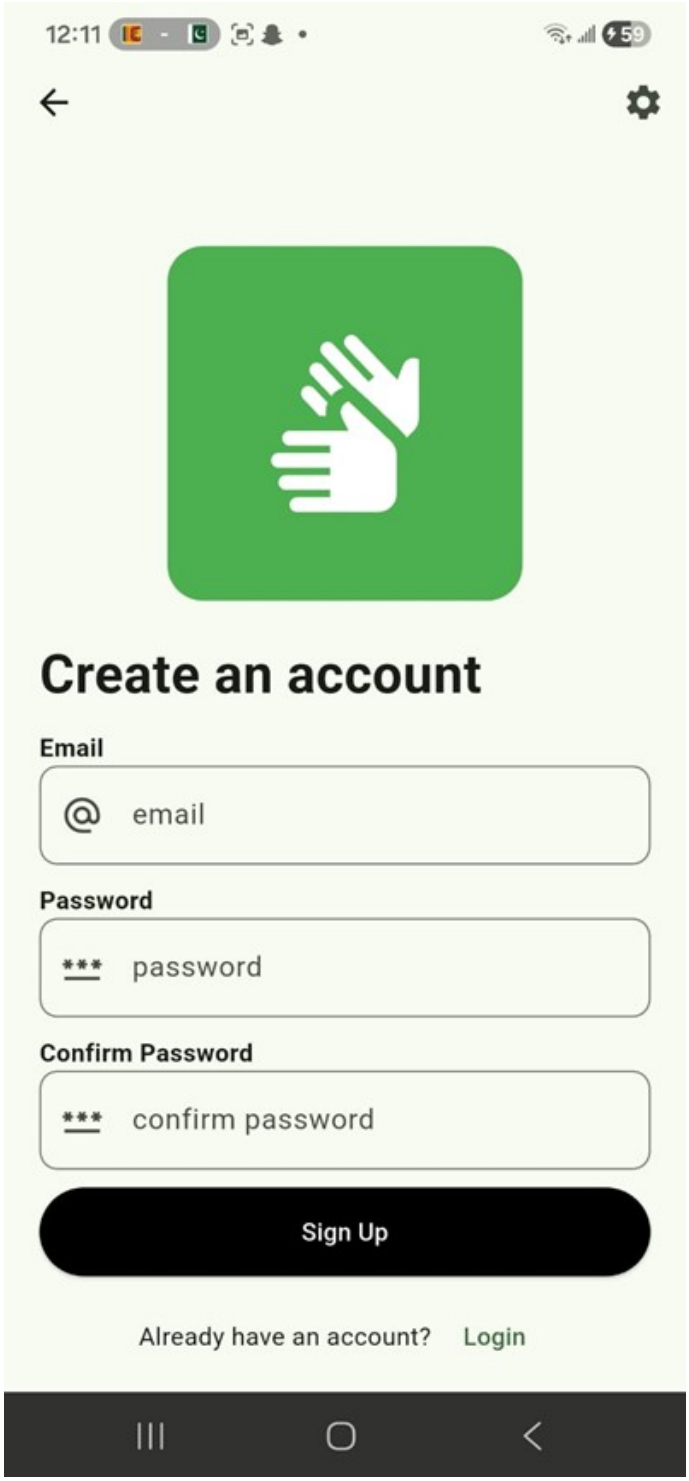

Figure A.1: Sign Up screen for creating a new account.

A.2.2 Log In

Once the account is created, return to the Login screen and enter your registered email and password to access the app. The Login interface is shown in Figure A.2.

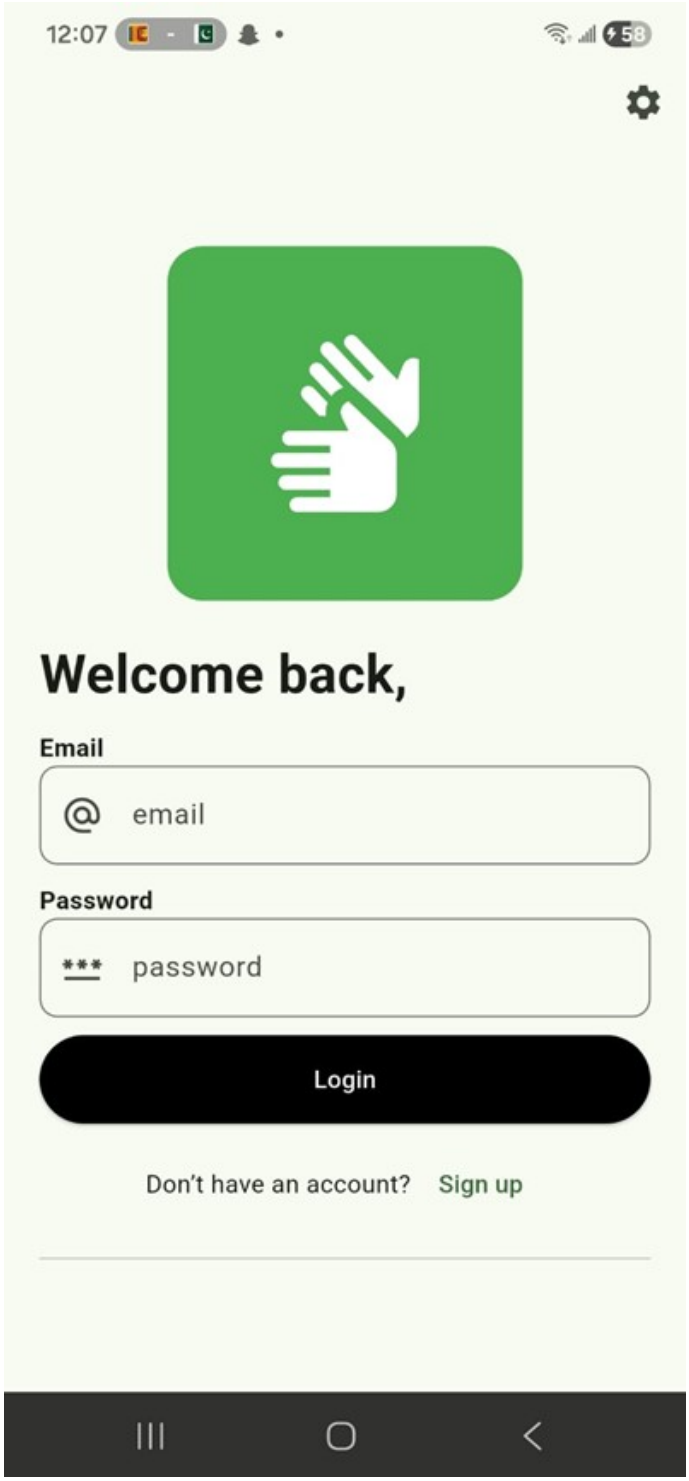

Figure A.2: Login screen for existing users.

A.3 Using the Application

A.3.1 Home Screen

After successful login, the Home screen is displayed. From this screen, users can start the sign language translation process by selecting the *Start Detection* option, as shown in Figure A.3.

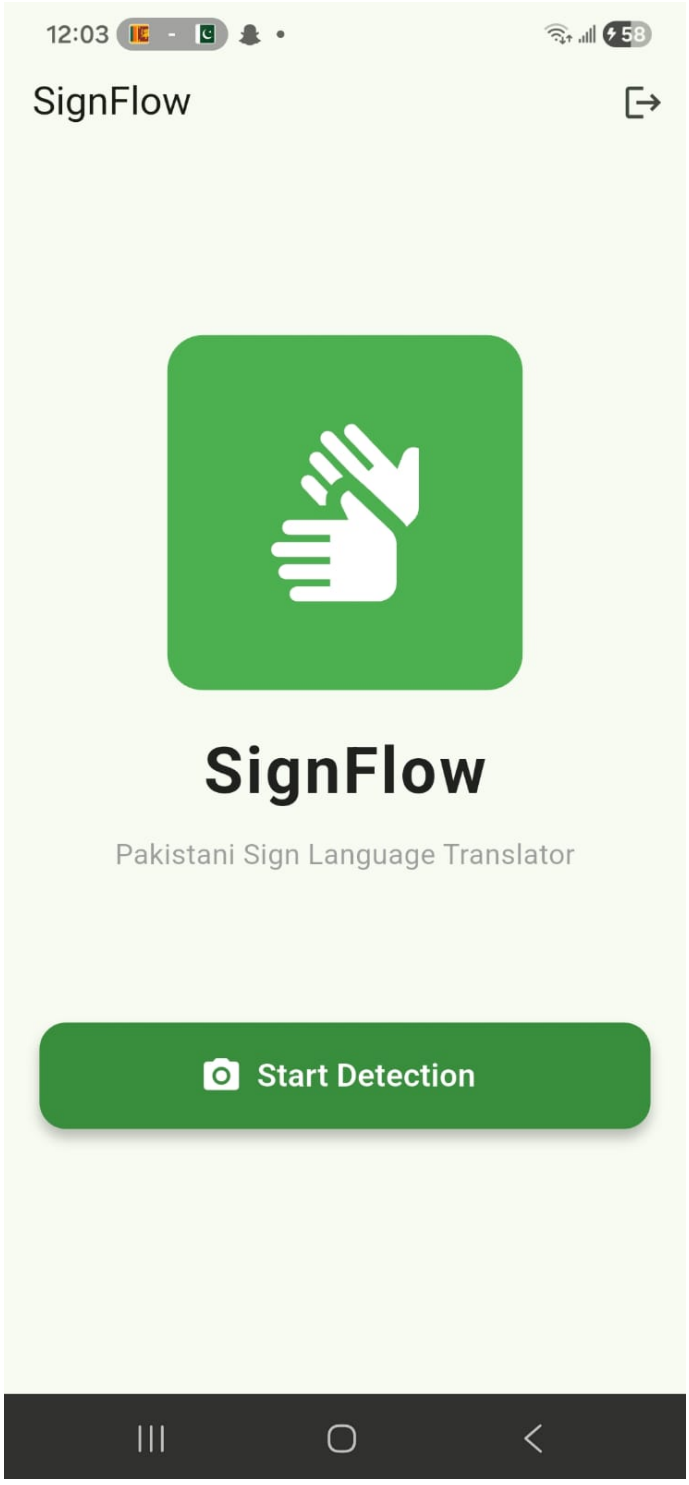

Figure A.3: Home screen of the SignFlow Translator App.

A.3.2 Detection Screen

The Detection screen allows the app to capture hand gestures using the phone camera. The recorded sign is then processed by the system to generate the corresponding text output.

The detection interface is shown in Figure A.4.

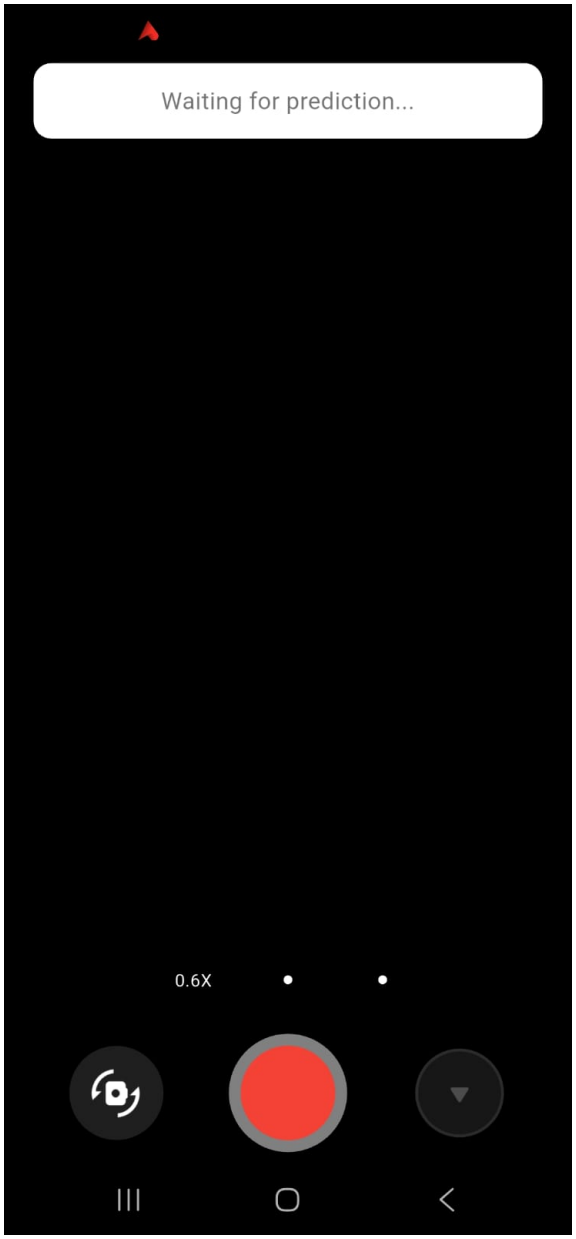

Figure A.4: Detection screen for PSL sign recognition and translation.

A.3.3 Loading Screen

While the gesture is being analyzed by the model, a loading screen is shown. This helps avoid interruptions and ensures that the translation process runs smoothly. The loading screen is shown in Figure A.5.

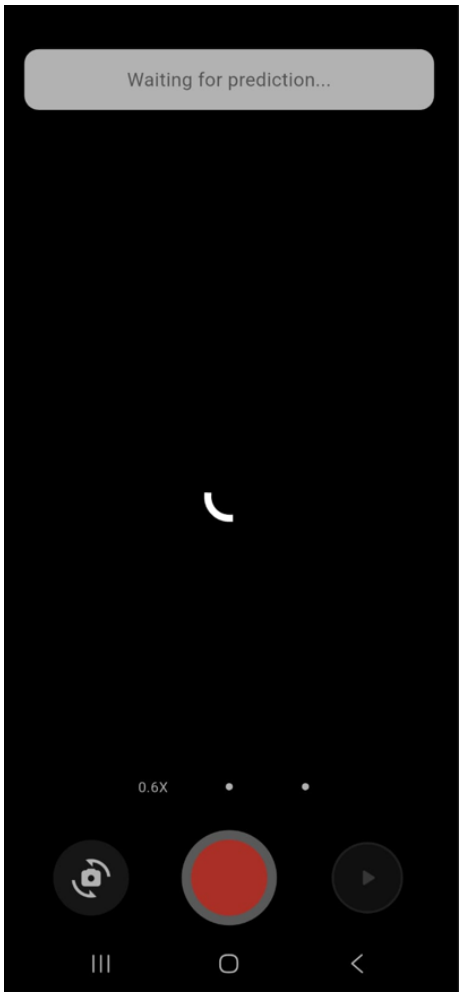

Figure A.5: Loading screen displayed during translation processing.

A.4 Tips & Troubleshooting

The following tips can help improve the overall experience while using the app:

- **Camera permission issue:** Make sure camera access is enabled in the phone settings.
- **Slow response:** A stable internet connection is recommended for faster prediction results.
- **Low accuracy:** Ensure good lighting and keep hands clearly visible within the camera frame.
- **Unrecognized sign:** The current version of the app supports a limited set of PSL signs (20 signs). If a sign is not recognized, try using a supported gesture.

A.5 Privacy Note

User login information is stored securely, and camera input is used only for translation purposes. The system does not store video data unnecessarily and is designed to protect user privacy.

References

[1] World Health Organization. Deafness and hearing loss: Fact sheet, March 2025. Accessed: 2025-11-25. Over 5% of the world's population – or 430 million people – require rehabilitation to address their disabling hearing loss (including 34 million children). Cited on p. 1.

[2] Irfan Bashir, Sidra Bano, and Afshan Naseem. Barriers to inclusivity: Sign language interpreters' challenges at higher education institutes. *Qlantic Journal of Social Sciences*, 6(1):213–219, 2025. Cited on p. 2.

[3] Ali Ahmed Raza and Ahtisham Fazal. Signvista. Bachelor's thesis, Department of Computer Science, Bahria University, Islamabad, October 2024. Supervisor: Mr. Qazi Mohsin. Cited on p. 4.

[4] Jahanzeb Naeem and Zain Ul Abedien Asif. Signchatter. Bachelor's thesis, Department of Computer Science, Bahria University, Islamabad, June 2023. Supervisor: Dr. Arif Ur Rahman. Cited on p. 5.

[5] Muhammad Haris Sheikh. Sign language translator application using AI. Bachelor's thesis, Department of Computer Science, Bahria University, Islamabad, October 2022. Supervisor: Dr. Asfand Yar. Cited on p. 5.

[6] Mian Afzaal Zahoor and Suleman. Urdu sign language recognition. Bachelor's thesis, Department of Computer Science, Bahria University, Islamabad, June 2022. Supervisor: Ms. Maria Mahmood. Cited on p. 5.

[7] Saad Ahmed, Hasnain Shafiq, Yamna Raheel, Noor Chishti, and Syed Muhammad Asad. Pakistan sign language to urdu translator using kinect. *Computer Science and Information Technologies*, 3(3):186–193, 2022. Cited on p. 5.

[8] Neelma Naz, Maheen Salman, Fiza Ayub, Zawata Afnan Asif, and Sara Ali. Paksign: Advancing dynamic pakistani sign language recognition with a novel skeleton-based dataset and graph-enhanced architectures. *Computer Vision and Image Understanding*, page 104458, 2025. Cited on p. 8.

[9] Yi-Fan Song, Zhang Zhang, Caifeng Shan, and Liang Wang. Efficientgcn: Constructing stronger and faster baselines for skeleton-based action recognition. *arXiv preprint arXiv:2106.15125*, 2021. Cited on p. 29.

[10] Zhiyun Zheng, Qilong Yuan, Huaizhu Zhang, Yizhou Wang, and Junfeng Wang. Lightweight multiscale spatio-temporal graph convolutional network for skeleton-based action recognition. *Big Data Mining and Analytics*, 8(2):310–325, 2025. Cited on p. 29.

[11] Jingyao Wang, Issam Falih, and Emmanuel Bergeret. Skeleton-based action recognition with spatial-structural graph convolution. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2024. Cited on p. 29.