Software Requirements Specification

for

Sign-Bridge

Version 1.0 approved

Prepared by <Wong Wei Ming, Heng Yun Tat, Sathishkumar Avanesh, Sancheti Aanya, Soh Yong Hao, Elayalwar Surya Prakash>

<Nanyang Technological University>

<13th April 2025>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1.Introduction

1.1.Purpose

This document outlines the software requirements for the “SignBridge, version 1.0”. This SRS outlines the functionalities, features, and constraints of the platform aiming to provide a clear and comprehensive framework for development. Communication barriers between deaf and hearing individuals often lead to misunderstandings due to the limited understanding of sign language. Existing translation solutions are often inaccurate or lack essential features. This app addresses these challenges by offering real-time sign language translation, pre-recorded video analysis, and text-to-sign conversion. With advanced moderation tools to ensure accuracy and prevent misuse, it supports multiple sign languages, enhancing accessibility for deaf individuals, educators, healthcare professionals, and service workers in various social and professional settings.

1.2.Document Conventions

This SRS follows standard documentation conventions to ensure clarity and consistency. Key conventions include:

- Font Styles: 'Arial' for body text, 'Times New Roman' for headers.
- Highlighting: **Bold** for key terms, *italics* for emphasis.
- Requirement Prioritization: Higher-level requirements have inherited priority levels, cascading down to detailed requirements. Each requirement statement is explicitly labeled with a priority level (High, Medium, Low).
- Requirement Identification: Each requirement is uniquely identified for easy reference.
- Versioning: Changes in requirements through different versions are tracked for historical reference and future revisions.

1.3. Intended Audience and Reading Suggestions

This document is structured to cater to a wide array of stakeholders and should be read differently depending on the reader’s role:

- **Project Managers and Stakeholders**: Begin with the Introduction and *Overall Description* sections to understand the system goals, users, and context. These sections provide a broad overview of how SignBridge addresses communication barriers between deaf and hearing individuals.

- **Developers**: Focus on *System Features, External Interface Requirements*, and *Software Interfaces.* These contain functional breakdowns, sequence descriptions, and API/service-level integrations that support SignBridge's real-time video translation, video upload, and text-to-sign features.

- **Testers**: Prioritize *System Features, Use Case Descriptions*, and *Nonfunctional Requirements*. These sections help formulate both black-box and white-box test cases for validating live translation, video uploads, UI behavior, and language selection.

- **User Experience Designers**: Reference *User Interfaces* under *External Interface Requirements* and *Appendix B Dialog Map and Use Case Flow*. These define the app's navigation structure, real-time feedback systems (e.g., lighting prompts, multiple signer warnings), and accessibility features.

- **Technical Writers**: Utilize all sections, especially *User Documentation*, *Glossary,* and *Assumptions and Dependencies*, to create onboarding flows, in-app help tooltips, and user manuals tailored for both regular users and admins.

- **Quality Assurance Engineers**: Review *System Features*, *Nonfunctional Requirements*, and *Appendix C* for missing requirements or edge cases. Pay particular attention to performance thresholds (e.g., translation latency < 2s), security (e.g., role-based access), and usability (e.g., onboarding guidance, theme/layout customization).

All readers are encouraged to review the document in its entirety, as each section builds on the information presented in the preceding ones, providing a complete understanding of the SignBridge's requirements.

1.4.Product Scope

Deaf and Hard of Hearing Individuals

In Singapore, it's estimated that approximately 500,000 individuals experience some degree of hearing loss, accounting for about 8.4% of the population. Among the 5,400 individuals registered with the Singapore Association for the Deaf (SADeaf), about one-third are proficient in sign language.

Family Members and Friends

Relatives and friends of deaf individuals often seek effective ways to communicate.

Educators and Students

Teachers, interpreters, and students in educational institutions can utilize SignBridge for instructional purposes, enhancing the learning experience for both sign language users and learners.

1.5.References

OOP Principles:
https://khalilstemmler.com/articles/object-oriented/programming/4-principles/

Hand Sign Recognition:**https://link.springer.com/article/10.1007/s00521-024-09503-6**

azure:
https://learn.microsoft.com/en-us/azure/app-service/scenario-secure-app-overview

fastapi: https://fastapi.tiangolo.com/reference/

mongodb: https://www.mongodb.com/docs/manual/reference/database-references/

yolo pose recognition: https://docs.ultralytics.com/tasks/pose/

argon templates: https://github.com/creativetimofficial/argon-react-native

react native website : https://reactnative.dev/

expo eas: https://docs.expo.dev/tutorial/introduction/

2. Overall Description

2.1.Product Perspective

This SRS specifies the requirements for Signbridge, a novel creation that aims to bolster the communication between Deaf persons and the deaf community. Although similar applications may persist in the respective fields, this is one of the few initiates that serves the deaf community directly. Our platform functions independently and leverages on APIs. A *System Architecture Diagram (as shown below)* is provided to illustrate the interconnections and external interfaces.

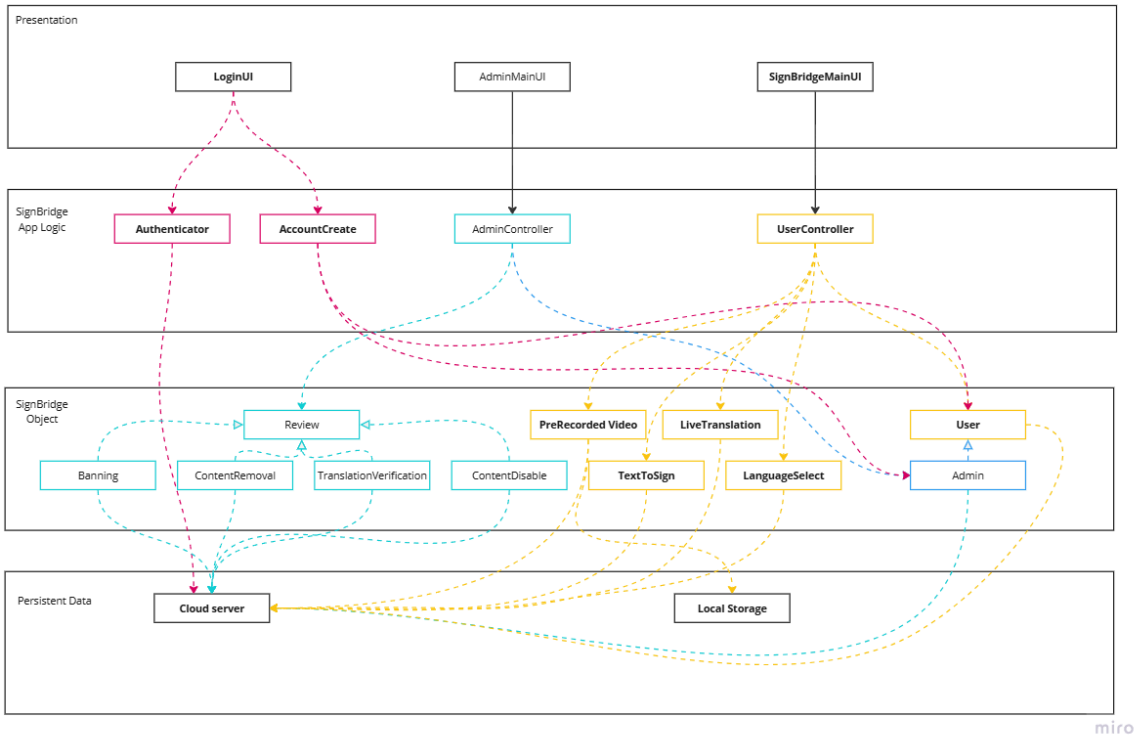


Figure 2.1.1: System Architecture Diagram

2.2.Product Functions

2.2.1.Use Case Diagram

The following use case diagram (Figure 2.2.1) depicts the key product functionalities that SignBridge includes. There are 2 primary users of Signbridge – Admin and Users. Each

type of user interacts with Signbridge to use their role-specific functions.The following Use Case Diagram depicts the key functionalities of Singbridge – notably the 4 key functions of *Live Translation, Pre-Recorded Video Upload, Text-to-Sign and Language Select.*

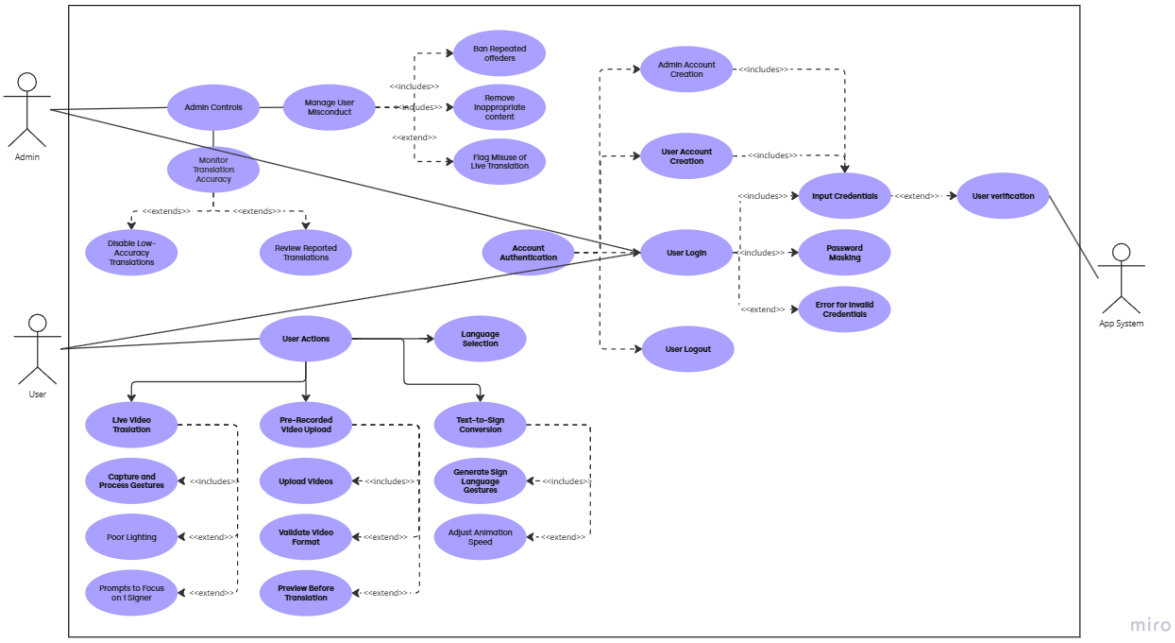


Figure 2.2.1: Use Case Diagram

2.2.2.Major Product Functions

Signbridge will allow users to perform the following functions:

- **Authentication**
  - Signup (with different roles – Admin, User)
  - Login
  - Logout

- **Features**
  - Live Cam translation
  - Pre-recorded video translation
  - Text to Sign Translation
  - Language Selection
- **User Functions**
  - **Live Cam translation**
    - Record live with camera and get text result
  - **Pre-recorded video translation**
    - Upload Videos
    - Preview them

        - Process the videos and get translated result
    - **Text to Sign translation**
        - Convert any given text into sign language
    - **Language Selection**
        - Select language for UI
- **Admin Functions**
    - Verify User
    - Ban User
    - Process Flagged Reviews
    - Notify User (Admin can send custom notifications)

- **Customer Service Support**
    - Real-time Messaging between Users and Admins
- **Other Functions**
    - Settings
    - Edit Profile

2.2.3 Class Diagram

The following Class Diagram (Figure 2.2.3) shows how different classes interact with each other, as well as the key functionalities that Signbridge needs to perform (refer to the controllers in the middle row section).

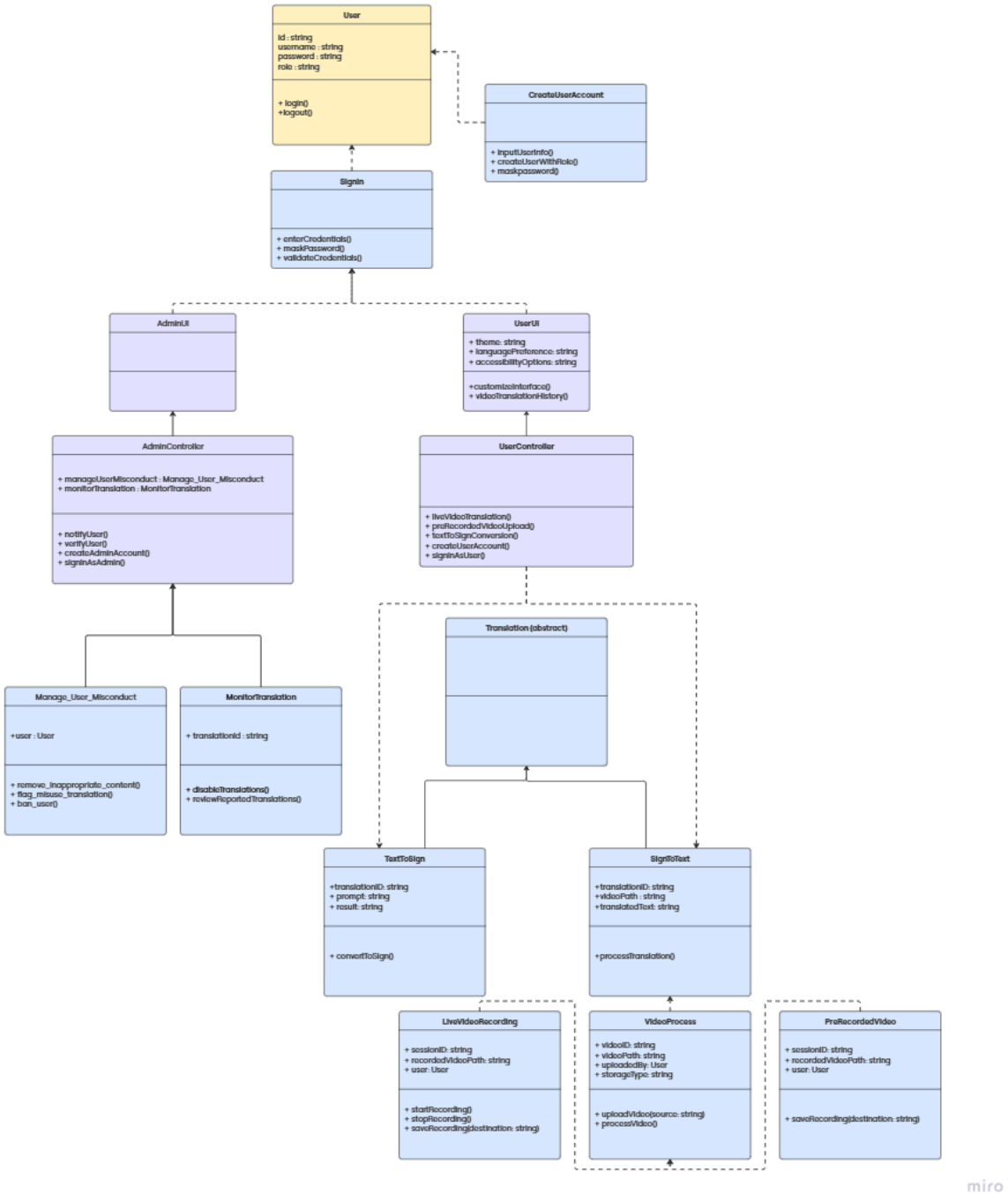


Figure 2.2.3: Class Diagram

2.3.User Classes and Characteristics

Users of the platform are categorized into:

- **Users:** ordinary users of the application, who can even be further classified into new and returning users of the application.
- **Admins:** privileged individuals with administrative power inside of the app, tasked with helping to moderate the environment inside the application.

2.4.Operating Environment

The SignBridge application is deployed in a cloud-based environment using **Microsoft Azure**. The frontend is hosted on **Azure Static Web Apps**, while the backend services — including the **custom trained computer vision (CV) model** and the sign-to-text conversion engine — are implemented using a **Flask API** and deployed on an **Azure Virtual Machine** running **Ubuntu Server 20.04 LTS**. Continuous integration and deployment are managed through **GitHub Actions**, ensuring reliable and automated updates from the GitHub repository to the Azure environment.

The Flask-based backend exposes RESTful APIs that handle image and video input processing, inference using AI models, and text generation. These services interface with a **MongoDB** cloud database used to store user information, video metadata, and processed translation results.

The entire system operates on cloud-hosted **x86_64 architecture**, and relies on a modern software stack including **Python**, **Node.js**, **React Native Web**, and standard browser media APIs (e.g., *getUserMedia*, *WebRTC*) for live camera and audio access. The client-side application supports **cross-platform operation across major operating systems** (Windows, macOS, Linux, iOS, Android) and browsers (Chrome, Firefox, Safari, Edge).

The application does not depend on any legacy systems or on-premise infrastructure. All components are cloud-native, container-compatible, and communicate over secure HTTPS using standardized data formats such as JSON.

2.5.Design and Implementation Constraints

2.5.1.Deployment Constraints (Cloud)

- **Cloud Deployment Limitations**
  Deployed on Microsoft Azure (unlike peers using local environments), introducing strict service constraints. Limited by:
    - Azure resource quotas (CPU, memory, storage)
    - Network latency and outbound data costs
    - GitHub Actions deployment workflow constraints
    - Need for containerization (e.g., Docker)
- **CI/CD and Infrastructure Constraints**
  Uses GitHub Actions for automated deployment pipelines to Azure VMs. Requires:
    - Custom build/test/deploy stages per environment
    - Secure management of environment secrets and tokens
    - Synchronization of frontend and backend deployments
- **Backend Processing Constraints**
  Flask API running on Azure VM handles inference and sign-to-text processing. Limited by:
    - Virtual machine performance variability
    - Cold-start delays and process spawning
    - API rate-limiting under high user loads
- **Real-Time Responsiveness**
  System must support real-time video and gesture processing from the browser to the API and back. Performance requirement:
    - Inference and response latency < 300 ms. Affected by:
        - Webcam input resolution and browser framerate (≥30 FPS)
        - Python Flask processing speed on VM
        - Network round-trip latency between frontend and backend

- **Security & Regulatory Constraints**
  Must comply with Azure and organizational IT security policies. Requires:
    - Encrypted communication (HTTPS for all endpoints)
    - Secure credential storage (e.g., Azure Key Vault)
    - Data privacy safeguards for user-submitted videos/text
- **Technology Stack Constraints**
  Specific tools and services were mandated by cloud compatibility and project scope. Must use:
    - Flask (Python) for backend API
    - MongoDB for cloud-based database
    - React Native Web for frontend
    - Browser-native APIs (getUserMedia, WebRTC)
    - GPU availability (Model Training)

2.5.2. Frontend Constraints

- **Real-Time Video Processing Latency**
  Must render translations within 300ms of gesture capture. Limited by:
    - Device camera framerate (minimum 30 FPS)
    - GPU acceleration availability
    - iOS/Android video pipeline differences
- **Accessibility Compliance**
  Requires WCAG 2.1 AA standards for deaf/hard-of-hearing users:
    - Custom contrast ratios (4.5:1 minimum)
    - Haptic feedback for alerts
    - No audio-only cues

2.5.3. Backend Constraints

- **Model Size Limitations**
  On-device ML models must stay under 50MB (mobile app store limits) Impacts:
    - Gesture recognition accuracy
    - Supported sign language variants
- **Data Privacy Regulations**
  Raw video frames cannot be stored/transmitted without consent. Must comply with:
    - GDPR (EU)
    - HIPAA (if used clinically)
- **Language Constraints**
  Regional Gesture Variations. Will need to support:
    - ASL vs. SGSL differences (e.g., two-handed alphabets)
    - Cultural gesture conflicts (e.g., "thumbs up" meanings)

2.5.4. Device and Platform Constraints

- **Mobile-First Optimization**: The application must be fully functional on mobile devices (iOS and Android), with responsive layouts and camera access permissions.
- **Camera Access**: Live translation relies heavily on camera input. This introduces dependency on hardware support for resolution, frame rate, and low-light performance.
- **Storage Permissions**: Required for video upload and history logging features on both web and mobile platforms.

2.5.5. Integration Constraints

- **Azure Cloud Services:**
  The system is tightly integrated with Azure for:
    - Hosting (App Service)
    - Authentication (API Management)

- Storage (Blob Storage)
- Monitoring (Azure Monitor, Application Insights)

- **Third-party APIs:**
  - Google Translate API – used for translating spoken language components
  - MongoDB Atlas – for storing user data, history, and role-based access control

2.5.6. Real-Time Constraints

- **Latency Requirements**:
  Real-time translation for live video must have a latency under 2 seconds. This requires optimization in video processing and model inference on the backend.

- **Concurrency Handling**:
  The system must support multiple simultaneous live translations and uploads without degradation in performance.

2.5.7. Security and Privacy Constraints

- **Role-Based Access Control:**
  Admin and user roles must be clearly enforced.

- **Encryption:**
  All user data (especially translation logs and credentials) must be encrypted during transmission and at rest (e.g., on Azure Blob and MongoDB).

- **Data Retention Policy:**
  Translation history must be stored locally for 30 days and encrypted.

2.5.8. Accessibility Constraints

- **Screen Reader Support**:
  UI must be compatible with VoiceOver, TalkBack, and other assistive tools.
- **Customizable UI**:
  Users with visual or motor impairments can choose high contrast modes and layout profiles (e.g., "Accessibility Mode").

2.6.User Documentation

SignBridge provides a comprehensive set of user-facing documentation to facilitate ease of use, clarity, and accessibility for all user roles, including Admins, Deaf or Hard-of-Hearing Individuals, Educators, and Friends/Family.

2.6.1. README Files

- **Main Repository**

  The main repository includes a *README.md* that outlines:

  - Project Overview (SignBridge's purpose and target users)
  - Features Summary (Live Translation, Pre-recorded Video Upload, Text-to-Sign, Admin Moderation)
  - System Architecture overview
  - CI/CD Deployment workflow (Azure + GitHub Actions)
  - Folder structure and relevant links to frontend, backend, and documentation

2.6.2.Frontend Repository

The frontend README contains:

- Prerequisites: Node.js, Expo CLI
- Deployment Notes (Azure Static Web Apps)
- Build Instructions for Web

2.6.3.Backend Repository

The backend README provides:

- Python version: 3.10+
- Dependency Installation guide
- API Startup Instructions

- Deployment strategy (Azure VM + SSL Proxy)
- API structure (e.g., /predict, /auth/login , /flaskapp/predict)

2.6.4. Code Comments

- All core modules (e.g., live translation, user authentication) are commented using conventional inline documentation.
- Functions and classes include docstrings for parameters, return values, and exception handling.
- TODO: and FIXME: tags are used for features under development or requiring fixes.

2.6.5. Backend API Documentation

- **Framework:** FastAPI
- **Interactive Docs:** Swagger UI auto-generated at /docs
- **API Capabilities:**
    - /auth/signup – user/admin registration
    - /auth/login – login endpoint with JWT token
    - /flaskapp/predict – accepts real-time camera feed for gesture-to-text translation
    - /predict – accepts MP4/MOV uploads and returns gesture interpretation
    - /admin/review, /admin/ban, etc. – moderation endpoints
- **All routes specify:**
    - **Input format:** JSON or video blob
    - **Security:** Bearer Token auth
    - **Response codes:** 200 (OK), 400/401/403/500 for validation/failure

2.6.6.User Manual Overview

Role	Key Tasks
User	Account creation, Live Translation, Upload Pre-recorded Video, Text-to-Sign
Admin	Monitor flagged content, Ban/Review users, Improve translation accuracy
All	Language selection, Translation history logs, Layout & Theme customization

Tooltips, onboarding tutorials, and error prompts are built-in to assist first-time users.

2.7.Assumptions and Dependencies

2.7.1.Assumptions

It is assumed that the platform will have access to the internet at all times.

2.7.2.Dependencies

- **Front-end Libraries**

The user interface of FeedItForward depends on numerous front-end libraries:

Library	Purpose
react	To build a composable, reusable and dynamic app

react-dom	To render and manage React components in the browser's DOM
react-i18next	To enable internationalization for the purpose of language selection
i18next	An internationalization (i18n) library for JavaScript apps, enabling translation management, pluralization, and language detection
react-native	To ensure that the app supports cross platform function in the future
react-native-animatable	To add animations to the app
react-native-gesture-handler	To handle touch gestures and user interactions with the app
react-native-modal-dropdown	To implement dropdown menus with a modal overlay
react-native-paper	To provide material design components (eg. snackbar pop ups)
react-native-reanimated	To create smooth, high performance animations
react-native-safe-area-context	To handle safe area insets for devices with UI obstructions
react-native-screens	To optimize screen navigation and memory usage
react-native-web	To run React Native in a web browser
react-webcam	To access and display webcam streams in our app

Expo	To streamline our React Native app development with pre-configured tools/services
expo-app-loading	To manage app startup and loading resources

expo-asset	To load and manage assets (images/fronts)
@use-expo/font	For certain fonts
expo-av	To handle video playback and recording
expo-font	To load and use custom fonts
expo-image-picker	To access the device's gallery and camera
expo-localization	To retrieve device locality and language settings
expo-modules-core	To provide core functionality for Expo modules
expo-splash-screen	To customise and control Signbridge's splash screen behaviour
@expo/metro-runtime	To enhance metro bundle with Expo-specific features
@expo/vector-icons	To use common icons (eg. Material)

@react-navigation/native	Core library for navigation structures
@react-native-masked-view/masked-view	To assist in navigation (by stack navigator)
@react-navigation/stack	To allow stack-based navigation
@react-navigation/bottom-tabs	To allow tab-based navigation at the bottom of the screen
@react-navigation/drawer	To allow a slide-out side menu navigation
@react-navigation/compat	To mend compatibility issues

babel-plugin-module-resolver	To simplify import paths in JavaScript projects by aliasing directories
galio-framework	To build customizable components with clean design, built on top of the base React Native App

glob	A Node.js utility to match file paths using wildcard patterns
moti	To simplify creating smooth, declarative animations with intuitive APIs
prop-types	A runtime type-checking utility for React props to catch bugs during development by validating data types passed to components.
uuid	To generate universally unique identifiers

- **Deployment based Front End Libraries**

Node.js	Required for build steps
npx expo export	Builds static web version of Expo React Native app
React / Vite / Expo CLI	Framework-specific build tooling
actions/setup-node@v3	Installs Node.js (v20.x)
npm install --legacy-peer-deps	Installs JavaScript/TypeScript dependencies from *package.json*
Azure/static-web-apps-deploy@v1	Deploys built frontend to Azure Static Web Apps
actions/checkout@v3 (CICD Based Library)	Checks out your code from GitHub

- **Back-end Libraries**

The backend of FeedItForward depends on numerous back-end libraries:

Library	Purpose

fastapi	To provide a framework for developing the backend server
flask	To provide a framework for developing the backend server
uvicorn	To serve the backend servers
sqlalchemy	To provide a Python interface to SQL
httpx	To provide a more reliable HTTP request interface
pymongo	To connect to Azure Cosmos DB and perform database operations
python-dotenv	To load environment variables from a *.env* file
pytorch	To train and load deep learning models for translation
ultralytics	Provides easy-to-use implementations of custom-trained YOLO models for sign language detection
bcrypt	To provide hashing functions
python-multipart	To enable the server to accept file uploads

2.7.3.External Services

SignBridge depends on external services:

- **Cloud hosting services:**
    - GitActions Pages
    - Azure App Functions
    - Azure Virtual Machine

Hence, SignBridge relies on the availability and reliability of external services.

3.External Interface Requirements

3.1.User Interfaces

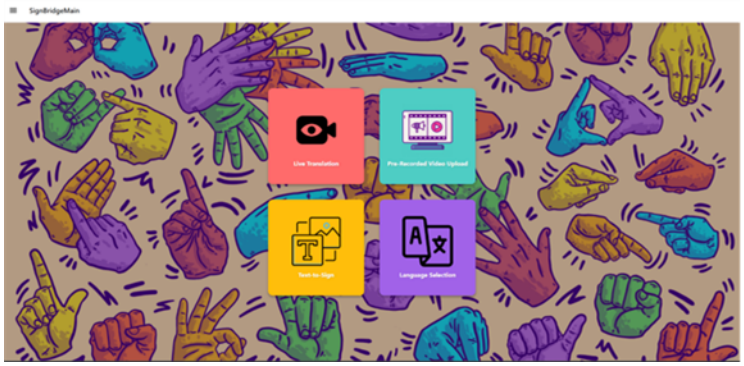
This section describes the logical characteristics and design requirements for the user interfaces (UIs) of the SignBridge system. It outlines the key interface screens, layout conventions, visual standards, and user interaction flows.

3.1.1.Welcome Page

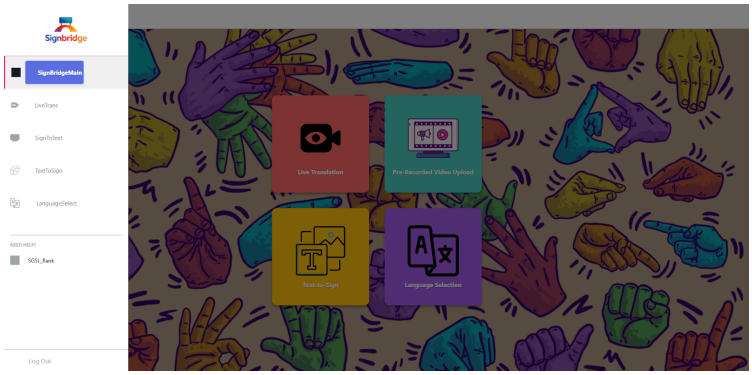
3.1.2.Login Page/ Account Creation

- *Failed Authentication for login*

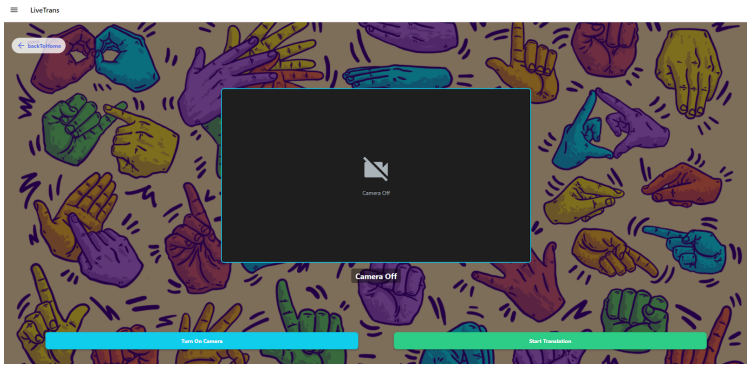
3.1.3.Main Home Page



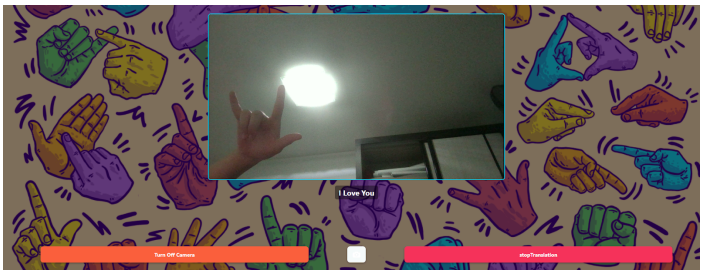
3.1.4.SideBar



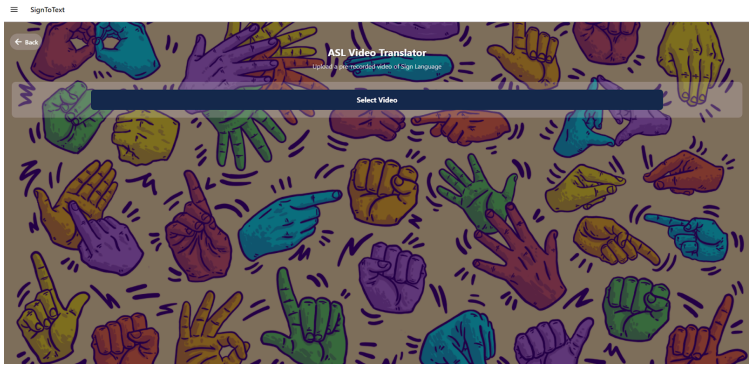
3.1.5.Live Video Translation Page



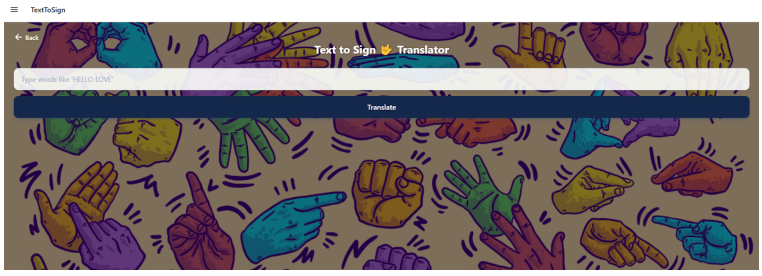
- Successful Translation of live video



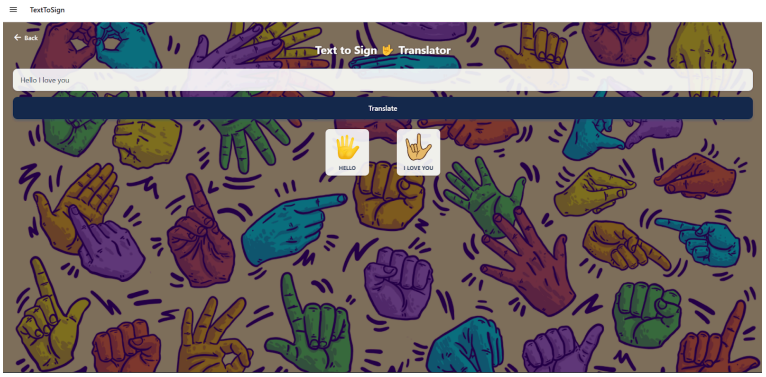
3.1.6.Sign to text video translator page



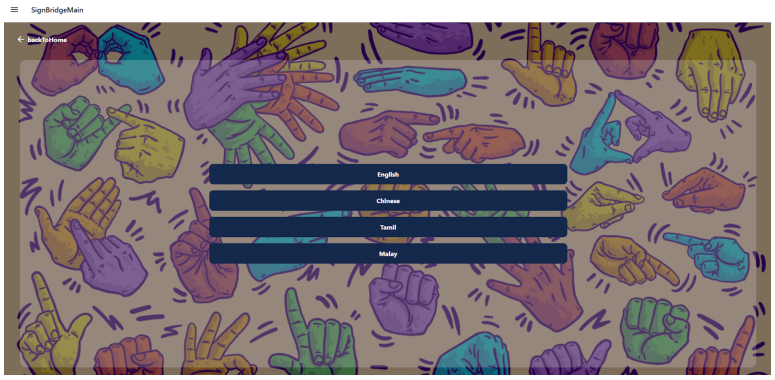
3.1.7.Text to sign translation page



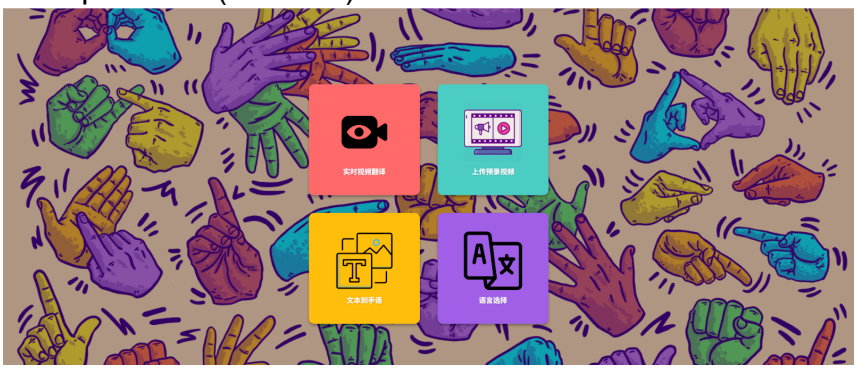
- Translated text “Hello I love you”



3.1.8.Language customisation page



- Example Case (Chinese)



3.1.9. 🎨 GUI Styling Guidelines

Category	Guideline
General Aesthetic	• Use bright, welcoming colors to represent diversity in communication • Sans-serif fonts • Large font sizes for visibility

Standard Buttons	• Buttons (e.g., “Let’s Go!”, feature tiles) must include:
	  ○ Hover state
	  ○ Active state
	  ○ Disabled state

Navigation	• All screens must support intuitive navigation via:
	  ○ Mouse / Touch
	  ○ Keyboard

Error Handling	• Errors must be shown via modals or toast notifications • Messages must explain the issue and suggest solutions (e.g., "Upload failed. Please try again.")
Accessibility Standards	• Comply with WCAG 2.1 Level AA • Include screen reader support, ARIA labeling, and sufficient contrast • Plan for future features like language toggles and sign-language avatars

3.2.Hardware & Interfaces Requirements

3.2.1.Supported Devices

- **Desktops**, **Laptops**, **Tablets**, and **Smartphones**
- Operating Systems: **iOS**, **Android**, **Windows**, **macOS**, **Linux**

3.2.2.Peripheral Interfaces

Peripheral	Purpose	Connection Method
Webcam	Required for real-time sign language detection and translation	Browser API (e.g., getUserMedia)

Microphone	Not core in current version; reserved for future speech-to-text features	Built-in, USB, or Bluetooth
Speakers	Used for auditory feedback in text-to-speech modules (if enabled)	Built-in, USB, or Bluetooth

3.2.3.Communication Protocols

- **Video & Audio APIs**:
    - getUserMedia, WebRTC, HTML5 Audio/Video
- **Frontend-Backend Transmission**:
    - Protocol: HTTPS
    - Format: JSON

3.2.4.Video Processing Specifications

- **Supported Formats**: Raw, MJPEG, H.264
- **Preprocessing**:
    - Resize to fit AI model input (e.g., 224×224, 640×640)
    - Channels: RGB or grayscale
- **Performance Requirements**:
    - **Minimum Camera Framerate**: 30 FPS
    - **End-to-End Latency Goal**: ≤ 300 ms for real-time responsiveness

3.3.Software Interfaces

3.3.1.Operating System

- **Supported Platforms**:

- Mobile: Android, iOS
- Desktop: Windows, macOS, Linux

- SignBridge is designed to function seamlessly across all common consumer platforms to support accessibility and device flexibility.

3.3.2.Database

- **Type**: Cloud-based NoSQL Database
- **Technology**: MongoDB
- **Data Stored**:
  - User credentials (email, password hash)
  - Translation history and metadata
  - Uploaded video references and processed results

3.3.3.Backend API

- **Framework**: Flask (Python)
- **Responsibilities**:
  - Handle image and video uploads
  - Run inference with ML models
  - Return translated text or animation data
- **Endpoints** (examples):
  - /translate/video
  - /translate/live
  - /text-to-sign

3.3.4.Machine Learning Model

- **Core Function**: Sign language recognition and translation
- **Input**: Video frames (RGB or grayscale)
- **Output**:
    - Translated text
    - Animated sign language visuals
- **Model Interaction**:
    - Exposed via backend API
    - Runs on cloud or edge inference server depending on deployment

3.3.5. Cloud Storage Interfaces

- **Supported Services**: Google Drive, iCloud
- **Functionality**:
    - Uploading pre-recorded sign language videos
    - Saving translated output
- **Integration**: Access via external APIs based on user authentication and permissions

3.3.6. Frontend Framework

- **Technology**: React Native Web
- **Purpose**: Cross-platform frontend supporting both mobile and web deployments
- **UI Elements**: Feature tiles, input forms, camera view, language toggle

3.3.7. Browser-based APIs

- **APIs Used**:
    - *getUserMedia* – access to webcam/microphone

- *WebRTC* – real-time video streaming
- *HTML5 Audio/Video* – playback and capture support

3.3.8.Data Exchange

- **Formats**: JSON over HTTPS
- **Data Types Exchanged**:
  - Video input frames to ML inference module
  - Translated text or animation from ML module
  - Authentication credentials (email/password) with the database
  - Metadata (e.g., timestamps, device info, user preferences)

3.4.Communications Interfaces

3.4.1.Web Communication Protocols

- **HTTP/HTTPS**:
  - All frontend-backend interactions are transmitted over secure HTTPS.
  - Ensures confidentiality and integrity of data during transmission.
  - Used for:
    - User authentication
    - Translation requests
    - Data retrieval
    - General frontend-backend API communication

3.4.2.Data Format

- **Format**: JSON (JavaScript Object Notation)
- **Usage**:

- All data sent between the React Native Web frontend and Flask backend API is formatted as JSON.
- Supports structured data exchange for user profiles, translation metadata, and output content.

3.4.3. Email Communication

- **Protocols**: SMTP (Simple Mail Transfer Protocol)
- **Usage**:
  - Account verification
  - Password reset functionality
  - User notifications or updates
- **Security**: Emails are sent securely over encrypted channels.

3.4.4. Sharing Mechanisms

- Users may optionally share translated output (e.g., text) via:
  - **Email**
  - **Social media platforms** (e.g., through web share APIs)

3.4.5. Backend API Interface

- **API Type**: RESTful
- **Backend**: Flask-based API server
- **Function**:
  - Exposes endpoints for all core functionalities
  - Serves as the primary communication layer between frontend and backend

3.4.6. Security Considerations

- All communication involving sensitive information (e.g., login credentials, translation logs) is:
    - **Encrypted via HTTPS**
    - **Authenticated using secure tokens (JWT or session-based)**
- The system ensures compliance with data privacy and transmission security standards.

4.System Features

4.1.Admins can perform administrative tasks

4.1.1. Description and Priority

SignBridge shall allow Admins to perform administrative tasks

4.1.2. Stimulus/Response Sequences

- **Moderation of User Violations**
  Stimulus: A user uploads content with offensive gestures.
  Response: The system automatically detects inappropriate content and flags it to the Admin dashboard.

  Stimulus: Admin accesses the flagged content.
  Response: The system displays the reported content and options to remove, warn, or ban the user.

  Stimulus: Admin chooses "Remove Content."
  Response: The system removes the content from public view and logs the action.

  Stimulus: A user repeatedly violates community guidelines.
  Response: The system aggregates violation history and flags the user profile for Admin attention.

  Stimulus: Admin selects "Ban User."
  Response: The system disables the user account and notifies the user of the ban and reason.

  Stimulus: A user misuses live translation with non-sign language gestures.
  Response: The system flags the session for review and alerts the Admin.
- **Monitoring Translation Accuracy**

Stimulus: A specific sign language translation receives multiple negative reports.
Response: The system decreases its confidence score dynamically.

Stimulus: Confidence score drops below 80%.
Response: The system generates a verification alert for Admins.

Stimulus: Admin opens the verification alert.
Response: The system displays the affected translation and user feedback.

Stimulus: Admin selects "Disable Translation."
Response: The system disables the translation temporarily and logs the admin action.

4.1.3. Functional Requirements

- **If there is usage misconduct or violation of terms, SignBridge shall allow Admins to take necessary actions against users.**
    - If a user uploads inappropriate content (e.g., offensive gestures, harmful signs), SignBridge shall allow Admins to remove the content.
    - If a user repeatedly violates community guidelines, SignBridge shall allow Admins to ban the user from the platform.
    - If a user attempts to misuse live translation for non-sign language gestures, SignBridge shall flag the user to the admin for review.

- **SignBridge shall allow Admins to monitor and improve language translation accuracy.**
    - If a specific sign language translation receives multiple negative reports, its confidence score will decrease. Once the confidence score is lower than 80 percent, SignBridge shall notify Admins for verification.
    - If a newly added sign language has low accuracy, Admins shall be able to disable it temporarily.

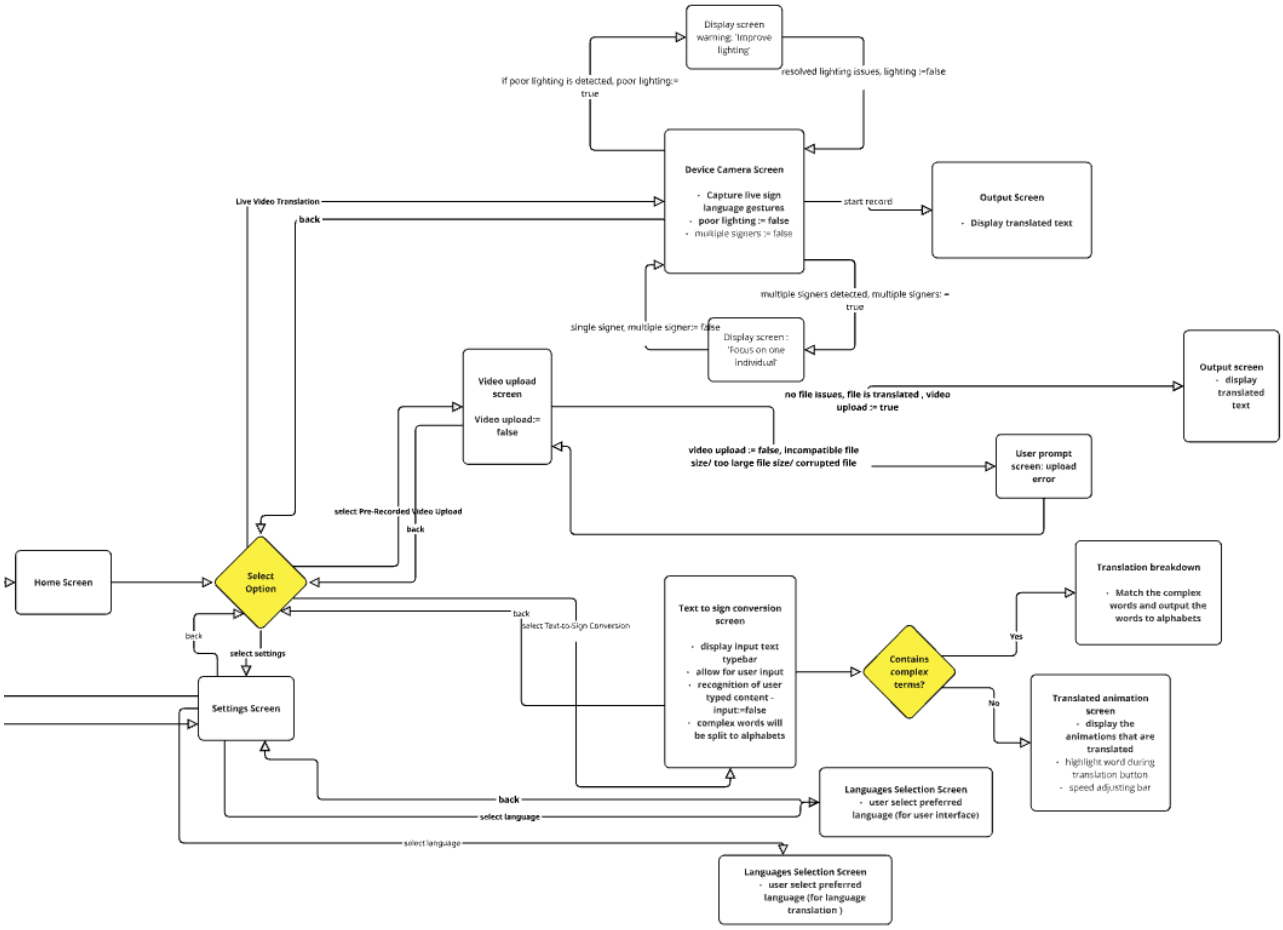
4.2. Users can perform user-specific tasks

4.2.1. Description and Priority

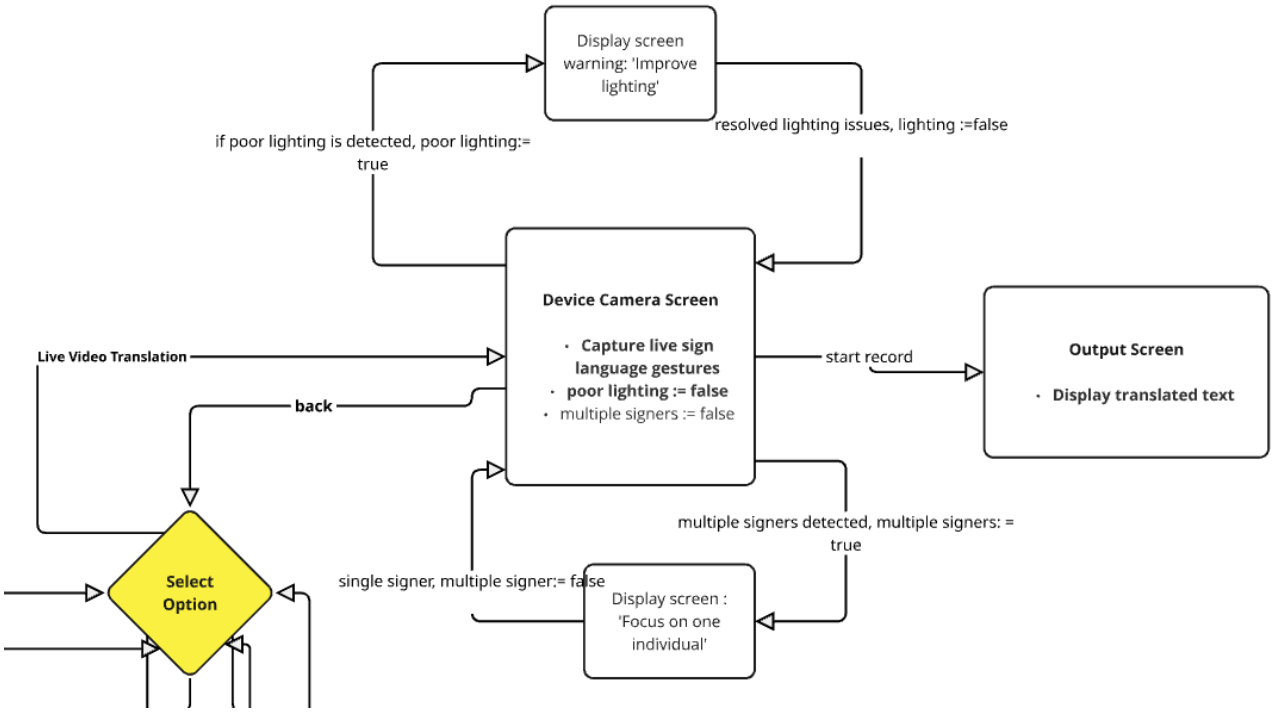
SignBridge shall allow Users to perform user-specific tasks

4.2.2. Stimulus/Response Sequences

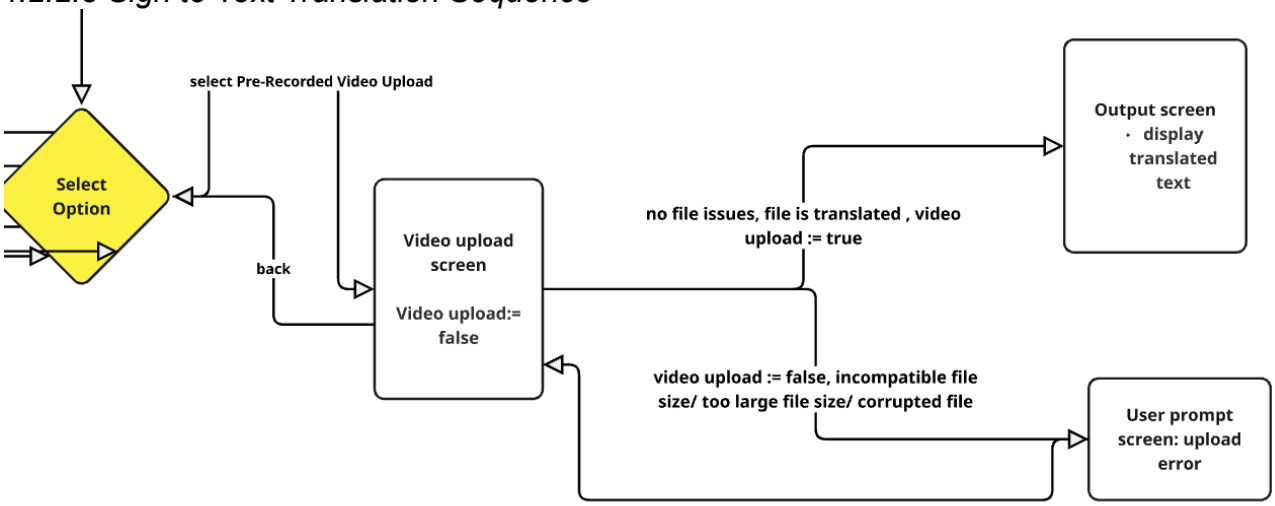
4.2.2.1 Overarching Sequence



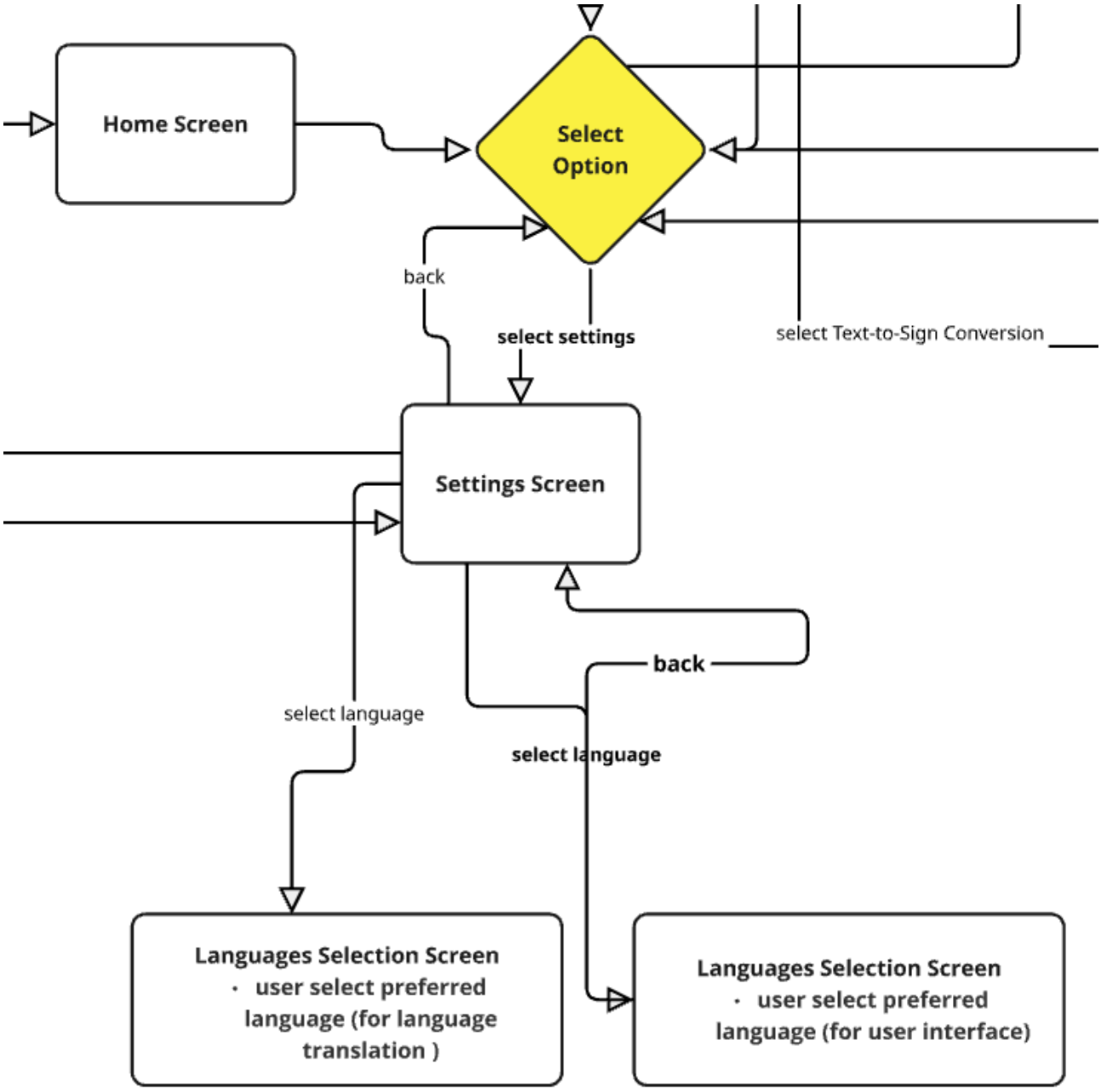
4.2.2.2 Live Video Translation Sequence



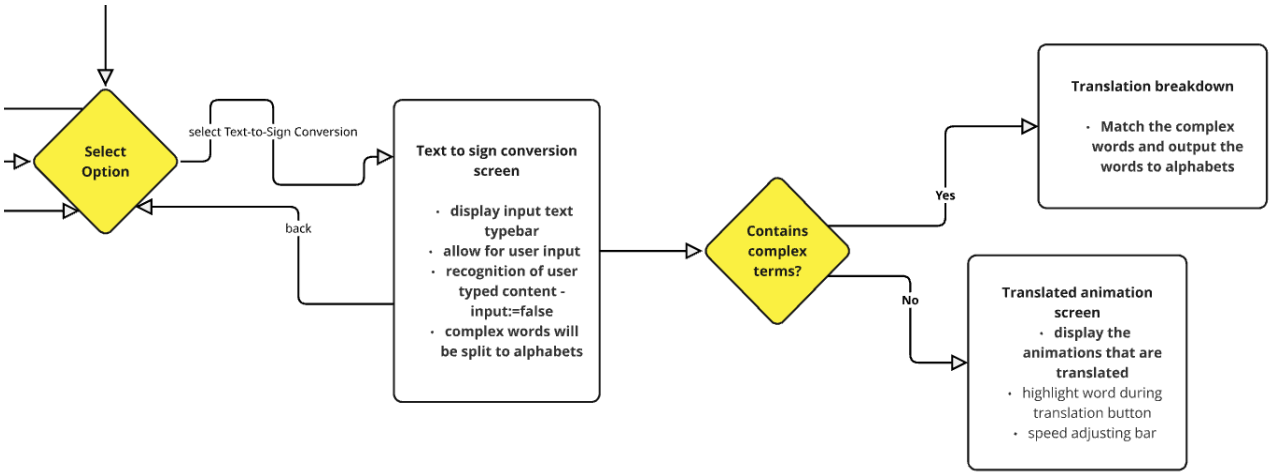
4.2.2.3 Sign to Text Translation Sequence



4.2.2.4 Language Settings Sequence



4.2.2.5 Text to SignTranslation Sequence



4.2.3. Functional Requirements

- **Live Language Translation**
  - When a User selects the "Live Translate" option, SignBridge shall begin real-time sign language interpretation.
    - The system shall activate the device camera to capture live sign language gestures.
    - The system shall process gestures and convert them into text/audio output in real-time.
    - If the lighting conditions are poor, the system shall display a warning message recommending better lighting.
    - If multiple signers are detected, the system shall prompt the user to focus on one individual for better accuracy.

- **Pre-Recorded Video Upload**
  - SignBridge shall allow Users to upload pre-recorded videos for translation.
    - Users shall be able to upload videos from local storage or cloud services (e.g., Google Drive, iCloud).
    - The system shall support common video formats (MP4, MOV, AVI).
    - If an unsupported video format is detected, the system shall prompt the user to convert the file.

        - Users shall have the option to preview uploaded videos before starting translation.
    - When a User submits a pre-recorded video, the system shall process it for sign language recognition.
        - The system shall analyze gestures in the video and generate corresponding text/audio output.
- **Text-to-Sign Conversion**
    - When a User enters text into SignBridge, the system shall generate sign language translation.
        - Users shall be able to type or paste text into a dedicated input field.
        - If the input text contains complex or technical terms, the system shall provide translation to alphabetical spelling
    - The system shall generate animated sign language gestures based on the input text.
        - The system shall provide an option to adjust animation speed (slow, normal, fast).
        - The system shall highlight each word as it is translated into sign language.
- **Language Selection**
    - SignBridge shall support multiple languages.
        - Users shall be able to select a preferred sign language from a predefined list for translation
        - Users are able to select z preferred language from a predefined list for User Interface language

5. Other Nonfunctional Requirements

5.1. Performance Requirements

5.1.1. Low Latency

The system must exhibit low latency to provide a seamless user experience. Specific latency targets are defined for key functionalities:

- **Video-to-Text Translation:** For video inputs shorter than 30 seconds, the translation process must complete within 2 seconds. This low latency is crucial for real-time understanding of short video clips.
- **Text-to-Sign Animation:** The rendering of text-to-sign animation must occur within 1 second after input submission. This ensures quick feedback to the user.

- **Live-to-Text Translation:** For every few seconds , data must be sent to the api and the output received within 1 second.

5.1.2. Resource Optimization

SignBridge must efficiently utilize device resources during real-time translation to avoid performance bottlenecks:

- **CPU Utilization:** The application must use no more than 15% of the CPU during real-time translation.
- **GPU Utilization:** GPU usage should be limited to a maximum of 20% during real-time translation.

5.1.3. Pre-Recorded Video Processing Time

- Must complete processing of standard 1-minute videos in <5 seconds.
- Enables quick turnarounds for recorded content.

5.1.4. Accuracy Threshold

- Translations must meet a minimum 80% confidence score to be displayed.
- Maintains trust and usability for both deaf users and hearing recipients.

5.1.5. UI Responsiveness

- All buttons and menus must respond within 200 milliseconds**.**
- Ensures smooth user experience, especially for accessibility users.

5.1.6. Storage Efficiency

- Processed video translations should consume no more than 100MB/min on the server.
- Helps scale the system with minimal server cost.

5.2. Safety Requirements

5.2.1. Content Moderation & Misuse Prevention

- **Automated Detection of Inappropriate Signs or Gestures**
  AI flags offensive or non-sign content for admin review

- **Admin Review Dashboard**
  Allows admins to ban, warn, or mute repeat offenders

- **Report Abuse Button**
  Users can flag harmful or offensive content in real time

5.2.2. Secure Authentication

- **Two-Factor Authentication (2FA)**
  Optional for both users and admins
- **Encrypted Login Credentials**
  Passwords securely hashed and stored
- **Role-Based Access Control (RBAC)**
  Limits access to admin functionalities

5.2.3. Privacy & Data Protection

- **Translation History is User-Private**
  Only viewable and accessible by the user
- **End-to-End Encryption for Live Translations**
  Ensures sign data is not intercepted during transmission
- **Data Deletion Options**
  Users can permanently delete their recordings or saved sessions

5.2.4. Safe AI Output

- **Confidence Threshold for Translation Display**
  Only outputs translations with ≥80% accuracy
- **Fallback Warning for Uncertain Translation**
  Alerts users when translation may be unreliable or partial

5.2.5. System Safeguards

- **Rate Limiting for Uploads**
  Prevents misuse via spam video submissions
- **Session Timeout & Auto Logout**
  Protects user accounts on shared devices

- **Server Health Monitoring & Failsafe Alerts**
  Prevents system crashes during live translations

5.3.Security Requirements

5.3.1. Authentication & Authorization

- The system **must implement secure authentication**, such as:
  - Email/password login (with salted and hashed passwords).
  - Optional OAuth2 or social login (e.g., Google, Apple).
- **Role-based access control (RBAC)** must be enforced:
  - Only **authenticated users** can access translation services.
  - Only **admins** can access logs, deploy models, or manage users.
- **JWT tokens** (or similar) must be used for session handling with **expiration and refresh mechanisms**.

5.3.2. Data Privacy & Protection

- All communication between the client and backend must use **HTTPS (TLS encryption)**.
- Uploaded content (e.g., sign language videos) must be:
  - **Stored securely** (encrypted at rest, if stored).
  - Deleted automatically after processing
  - Sensitive user data (e.g., passwords, tokens) must **never be stored in plaintext**.

5.3.3. Input Validation & Sanitization

- All user inputs (video, text, parameters) must be:
  - **Validated** for correct format and type (e.g., file size, video type, text length).

        - **Sanitized** to prevent injection attacks (e.g., command injection, XSS).
    - File uploads must be scanned for **malicious content** and restricted to approved formats (e.g., .mp4, .mov).

5.3.4. API & Model Security

- The AI model endpoints must:
    - Be accessible **only via the backend API**, not directly exposed to clients.
    - Include **rate limiting** and **throttling** to prevent abuse or denial-of-service (DoS) attacks.
- Any admin functionality (e.g., model deployment, data access) must require **strong authentication** and possibly **2FA**.

5.3.5. Logging & Monitoring

- Security logs must be maintained for:
    - Login attempts, account changes, and translation requests.
    - Admin activities like model uploads and configuration changes.
- Logs should be **anonymized** where appropriate and stored securely.
- Intrusion detection or alerting mechanisms should notify admins of suspicious behavior.

5.3.6. Secure Development Practices

- The codebase must follow **secure coding guidelines** (e.g., OWASP Top 10).
- Dependencies must be regularly scanned for known **vulnerabilities** (e.g., using pip-audit, npm audit).
- Secrets (API keys, DB passwords) must be stored securely using **environment variables or secret managers**, not in source code.

5.3.7. Mobile App Security

- App must **not store sensitive data in plain text** (e.g., local storage or SharedPreferences).
- Authentication tokens must be stored in **secure storage (e.g., Keychain, Keystore)**.
- Implement **obfuscation** and **code signing** to protect the app from tampering.

5.4.Software Quality Attributes

5.4.1. Usability

- The application shall provide a clean and intuitive user interface for both novice and experienced users.
- Users shall be able to complete a translation task (video ↔ text) in **3 steps or fewer**.
- **Ease of use is prioritized over ease of learning**, assuming users will reuse the app frequently.

5.4.2. Correctness

- Translations should achieve **≥ 90% accuracy** in controlled evaluation datasets.
- Input/output validation shall be enforced to prevent incorrect formatting or unsupported file types.
- The system must verify that video input conforms to required specs (e.g., resolution, length).

5.4.3. Reliability

- The system shall achieve **≥ 99.5% uptime** over a 30-day rolling period.
- All translation requests must either complete successfully or return a meaningful error within **5 seconds**.
- Failed jobs must be logged for audit and recovery purposes.

5.4.4. Robustness

- The system must gracefully handle unexpected inputs (e.g., corrupt videos, empty text) without crashing.
- Partial failures (e.g., model timeout) shall trigger fallback messages and logging without interrupting other operations.

5.4.5. Maintainability

- The backend codebase shall follow modular design principles to isolate model logic, API handling, and user management.
- New model versions should be deployable with **zero downtime** using containerization or versioning.

5.4.6. Portability

- The app must run on **both Web and Android**, with consistent user experience across devices.
- The backend shall be active in the cloud to ensure portability

5.5.Business Rules

5.5.1. User Roles and Permissions

- Access limited to their own accounts, uploaded content, and translation outputs.
- Can perform:
  - Live video translation
  - Pre-recorded video translation
  - Text-to-sign conversion
- Can:
  - View/download their translation history
  - Change language preferences
  - Provide feedback after each translation

5.5.2. Administrators

- Full system access including:
  - All user activity logs and translation data

        - System health metrics and performance dashboards
    - Permissions:
        - Approve, suspend, or restrict user accounts
        - Moderate flagged content or user behavior
        - Upload, validate, and deploy AI models
        - Modify usage limits and manage feedback loops

5.5.3. Platform Access and Security Policies

- AI Translation Services
    - Only authenticated users may access translation endpoints.
    - All translation actions are initiated via backend API calls.
    - Every translation request is logged with:
        - Timestamp
        - User ID
        - Model version

- Data Privacy and Retention
    - User-submitted videos/text are private and viewable only by:
        - The submitting user
        - System administrators (for moderation or technical review)
    - Temporary files (e.g., uploaded video blobs) are auto-deleted post-processing unless the user opts to retain them.
    - All data handling adheres to applicable data protection laws (e.g., GDPR, PDPA).

5.5.4. Usage Limits and Rate Control

- Default rate limits: **10 translation requests per hour** per user.
- Admins can:
    - Grant temporary or permanent extended limits
    - Disable limits during testing phases or special use cases

5.5.5. Translation Feedback Loop

- Users are prompted to rate translation accuracy after each session.
- Feedback is:
    - Logged with request metadata
    - Analyzed periodically by admins

        - Used to refine AI model performance and usability

5.5.6. AI Model Lifecycle Management

- Only administrators can:
    - Upload and validate new model versions
    - Deploy models into the live environment
- Requirements:
    - Pre-deployment validation: Accuracy ≥ 90%
    - Version control: All requests must be traceable to the model version used

5.5.7. System Reliability and Error Handling

- Failures in model inference or API processing will:
    - Prompt users with clear recovery instructions
    - Be logged for diagnostic review
- Recovery mechanism:
    - Retry options or fallbacks where applicable
    - Monitoring dashboards for system uptime and request success rates

5.5.8. Content Moderation

- Users must not upload content that includes:
    - Offensive, obscene, harmful, or illegal visuals or gestures
- Admins reserve the right to:
    - Review any submitted content
    - Remove content that violates terms of service
    - Suspend users who repeatedly breach policies

6. Other Requirements

6.1. Internationalisation Requirements

- Initial Support: SignBridge will initially support American Sign Language (ASL) and English text/audio.

- Future Expansion: Later versions will include regional sign languages (e.g., Singapore Sign Language, British Sign Language) and local dialects for improved inclusivity.
- Time Zone Standardization: All timestamps (e.g., saved translations, logs) will follow the Singapore Timezone (GMT+8).

6.2. Legal Requirements

- PDPA Compliance: The app must fully comply with the Personal Data Protection Act (PDPA) of Singapore when handling user data, especially video uploads and personal information.

- Consent for Recordings: Users must agree to terms before uploading or saving live video translations to ensure ethical and legal usage.

- Third-Party Services: Any integration (e.g., cloud storage, analytics) must comply with global privacy standards (e.g., GDPR, PDPA).

6.3. Reuse Objectives

- Modular Design: Components (e.g., Translation Engine, Moderation System) should be modular and reusable across different accessibility tools and platforms.
- Open Integration: SignBridge APIs must follow RESTful standards to allow integration with external education, healthcare, or accessibility platforms.

6.4. Accessibility Requirements

- SignBridge must meet **WCAG 2.1 Level AA** standards for web/mobile accessibility.

- Features include:

    - **Screen reader compatibility**

    - **High-contrast display themes**

    - **Gesture alternatives for non-visual users**

Appendix A: Glossary

Data Dictionary

Term	Definition
Account	A registered user's personal profile associated with an application. It may include personal information including but not limited to contact details and email address.
Admin	A user with special administrative privileges who oversees the operation and function of an application
Application	A software programme downloaded onto the User's mobile application. Certain system permission such as storage access or camera and microphone access may be needed to perform some functionalities of SignBridgelication
Server	A Cloud Server where all data possessed by all owners of accounts of our App is stored
Customer Service Support (Customer Service Support)	A team of administrators available during business hours, contactable via email; present to assist Users with any problems or receive feedback regarding SignBridgelication and their experience
Home Page	The main page of SignBridge, where a user can access all functions of SignBridge, including the settings, post login.
User	Persons who physically use our application

Live Translation	The page of SignBridge where a user can record live with the camera and get the translated text.
Pre Recorded Video translation	The page of SignBridge where a user can upload pre-recorded videos from local storage and get the translated text.
Text to Sign translation	The page of SignBridge where a user can input a text / sentence and get the output containing the corresponding sign gestures
Translation history	A log of past translations, up to 30 days, stored locally for user reference.
Confidence Score	Accuracy level of recognized hand gestures
User Interface	The screen of SignBridgelication shown to the user; that which the user interacts with, includes but is not limited to: login UI, HomePage UI etc.

Appendix B: Analysis Models

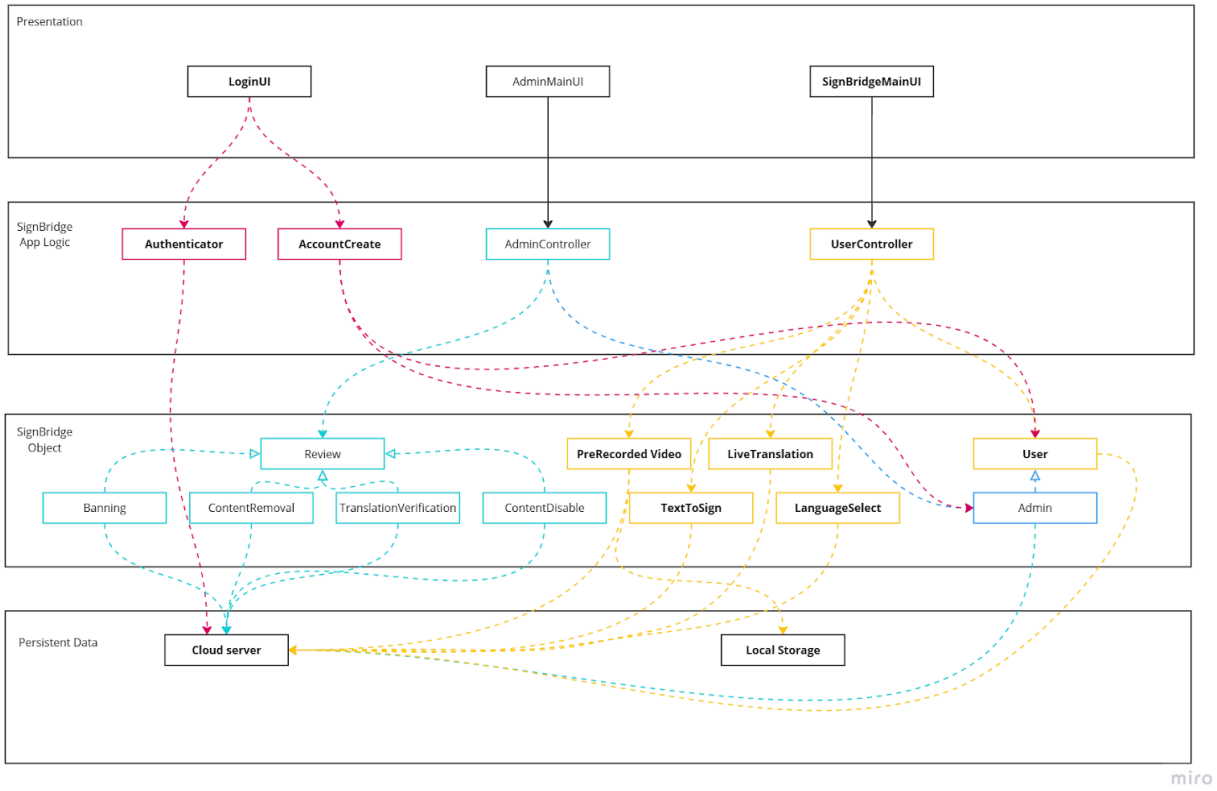


Figure 1: System Architecture Diagram

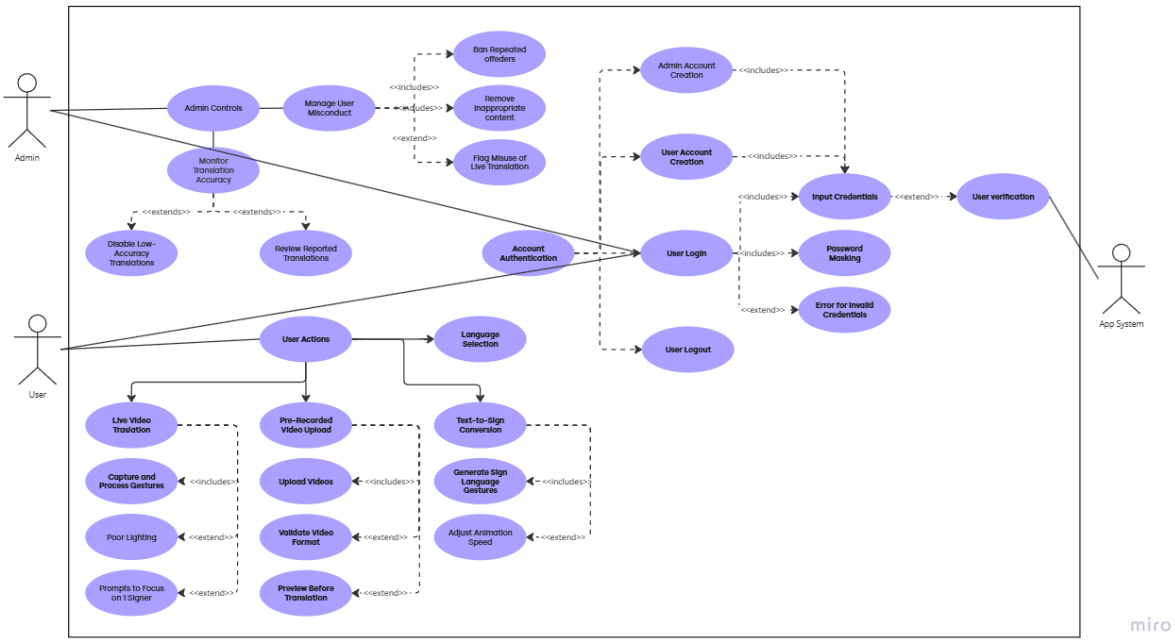


Figure 2: Use Case Diagram

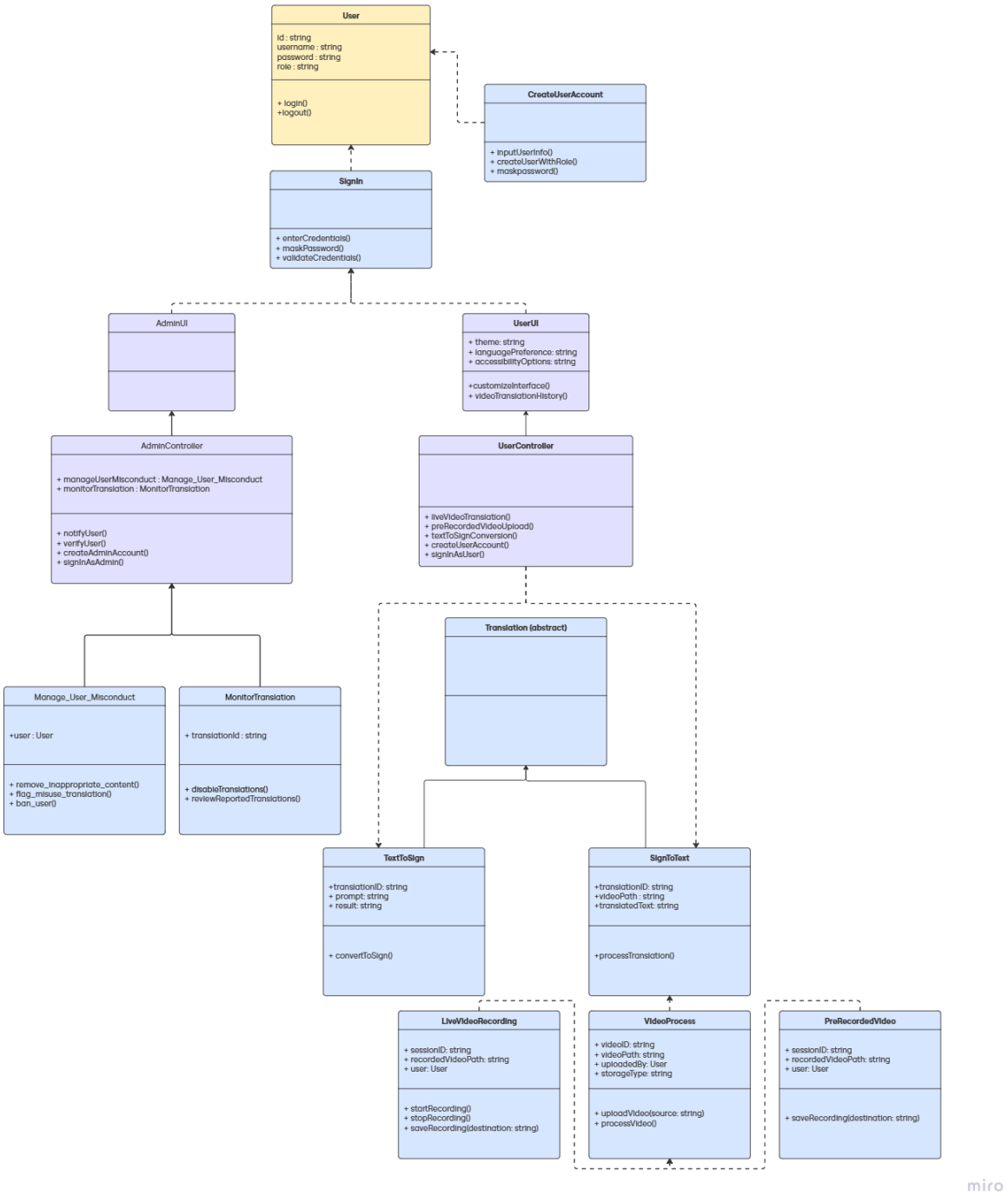


Figure 3: Class Diagram

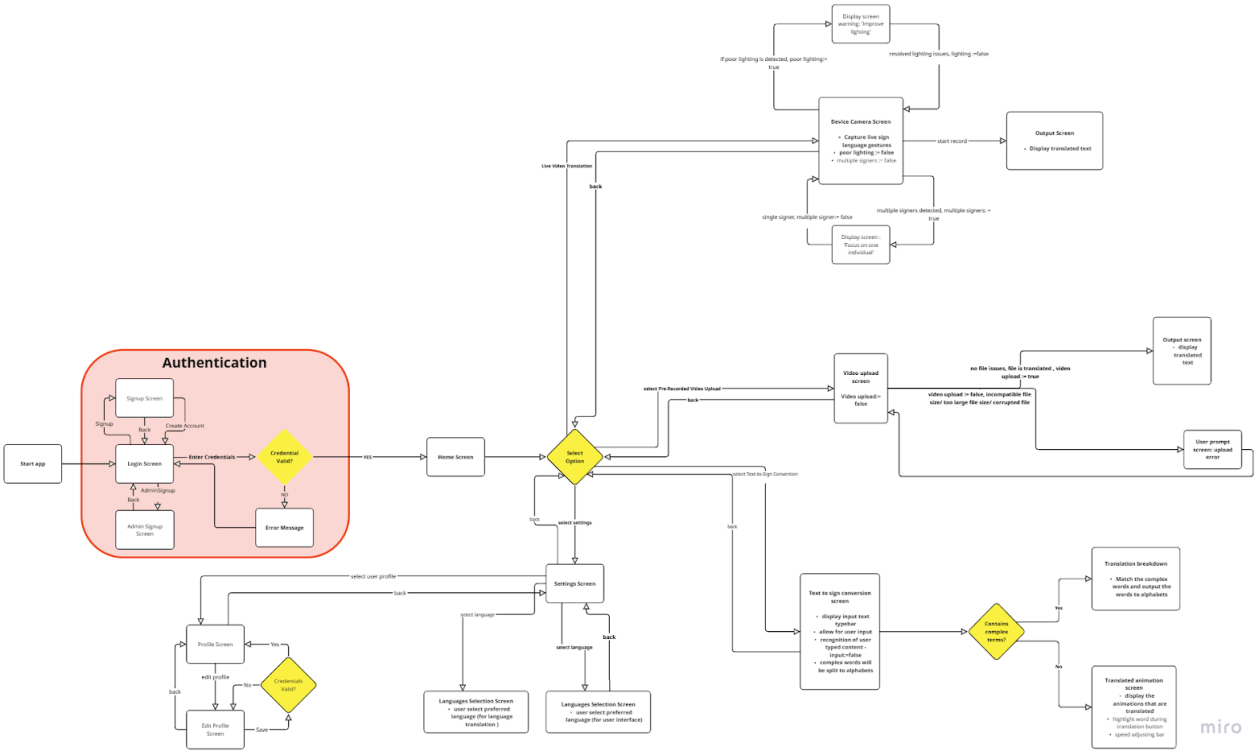


Figure 4: Dialog Map

Figure 5: Sequence Diagram

Appendix C: To Be Determined List

Source: http://www.frontiernet.net/~kwiegers/process_assets/srs_template.doc