Demo Script

Good Morning! We are team 4 and today we will be presenting our project **SignBridge-Bridging Deaf and the World**

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 (soon in multiple languages), 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.

The target users for our product is **Deaf and Hard of Hearing Individuals, Family Members and Friends, and Educators and Students.** Whether you're using live video or uploading pre-recorded clips, SignBridge interprets sign language into text and audio—fast and accurately.

The app's **functional requirements** include enabling users to perform tasks such as real-time translation, text-to-sign conversion, language selection, and account management. Admins are equipped with functions to moderate content, respond to user violations, monitor translation accuracy, and manage newly added sign languages. On the other hand, the **non-functional requirements** focus on enhancing user experience, performance, and security. These include a customizable user interface with theme and layout options, low-latency processing, efficient resource usage, secure and searchable translation history, and accessibility features like screen reader support and gesture customization. Together, these requirements ensure that SignBridge is both powerful and user-friendly, promoting seamless communication across diverse user needs.

Now lets talk about the Use Case Descriptions

Our Live Video Translation Use Case, which involves the user recording and translating a live sign language video. Here, the user is required to log into SignBridge, grant camera and microphone permissions, and select a target language. Upon accessing the live translation feature, the app opens the camera feed, and the user begins performing sign language gestures. These gestures are processed by SignBridge's AI engine, which translates them into real-time text. The translated text is then displayed with options to copy, save, or share. If any issues arise during the recording—such as poor lighting, connection problems, unrecognized gestures, or multiple signers—the app prompts the user with instructions to resolve them. **We emphasize real-time responsiveness, with system feedback provided when environmental conditions (like poor lighting) affect the experience.**

Our Pre-Recorded Video Use Case outlines how a user uploads a pre-recorded video for translation. In this case, the user must be logged in and have at least one video stored locally or in a supported cloud storage (e.g., Google Drive or iCloud) in a compatible format like MP4, MOV, or AVI. The user navigates to the upload section, selects the video, and optionally previews it. They then choose a target language from a predefined list, after which

SignBridge processes the video and converts the sign language gestures into text using AI. The translated output is then presented on-screen with options to copy, save, or share.

The app also supports **a text-to-sign conversion feature**. Here, users input text into the system, and SignBridge generates a sign language translation displayed back to them. This is useful for learners and those seeking to communicate with Deaf individuals. If the input text is too complex, contains improper grammar, or uses vocabulary not easily convertible into signs, the system alerts the user and suggests a simpler alternative for successful translation. This ensures that translations remain accessible and effective. **Our app promotes user support by simplifying complex inputs and enhancing understandability for both learners and fluent users.**

The next Use Case is centered on language selection. This use case enables users to choose their preferred sign language from a predefined list. Once selected, the system stores the preference and adjusts translation outputs accordingly. If no language is selected, the system defaults to ASL. If a desired language is unavailable, users are prompted to submit a request via the contact page. The assumption is that the system supports multiple sign languages and can adapt its UI and output to match user preferences. **We ensure user autonomy and personalization, which are key usability principles.**

When it comes to account creation and login, we have implemented authentication, security and data validation to make sure that only verified users have role-based access to the system. We once again have error handling through informative prompts.

Next, we have **Use Case 3.1**, which involves user account creation. A new user navigates to the sign-up page and inputs required details such as name, password, email, and profile picture. The system then validates the information and, if successful, creates the account with the “User” role. If no language is selected during registration, ASL is chosen by default. If the preferred language is not supported, users may request it via the contact form. This ensures users are onboarded easily and securely.

In parallel, **Use Case 3.2** handles admin account creation. It follows a similar process to the user sign-up, but with elevated privileges. Admins input their credentials, and the system validates their identity before assigning them the “Admin” role. Security is a critical requirement here. If the entered email is invalid, already registered, or if the password is weak, the system prompts the admin to revise their input. This ensures that only verified individuals are granted administrative access.

Lastly, **Use Case 3.3** explains how users sign into the platform. The user enters their email and password, and the system verifies the credentials. If correct, the user is logged in and redirected to the home screen. This basic yet essential use case supports secure access to all the features outlined above.

In conclusion, these use cases form the foundation of SignBridge’s operations, covering everything from live and recorded sign translation to content moderation, language customization, and account management. Together, they create an inclusive and functional

system that bridges communication gaps between sign language users and the broader community.

Next we will be talking about the various diagrams - **Use Case Diagram**, **Sequence Diagram**, **Dialogue Diagram**, **System Architecture Diagram**.

1. Use Case Diagram (UML)

Our UML diagram applies the principle of clarity and abstraction, showing actor interactions and distinguishing between required and optional functionalities through include and extend relationships. This also aids in modularity and scalability of the system.

The Use Case Diagram for the SignBridge application provides a clear overview of the system's functionalities by mapping out how different actors — Admins, Users, and the App System — interact with the app. Admins have access to features like monitoring translation accuracy and managing user misconduct. This includes actions like banning repeated offenders, removing inappropriate content, and flagging misuse of live translations. These actions are extensions of a broader category called Admin Controls, and they rely on the admin being authenticated. On the user side, users are able to perform actions such as live video translation, uploading pre-recorded videos, and using text-to-sign conversion features. These actions are supported by gesture capture, validation, previewing videos, and animation speed adjustment. Account authentication serves as a gateway for both admin and user login, which includes entering credentials, masking passwords, and handling invalid entries. Additionally, users can log out after completing their actions. Overall, this diagram outlines how various use cases are interconnected, showing mandatory ("include") and optional ("extend") functionalities in an organized structure.

2. Sequence Diagram

Our Sequence diagram reflects predictability and user flow, visually demonstrating how data flows through the system during operations like uploads, translations, and text inputs. It also supports real-time responsiveness, showcasing feedback loops at every step when using Signbridge

The process begins with the user entering their login credentials via the Login UI. If successful, they gain access to the Home Page UI. From here, the user can initiate various translation processes. If the user chooses to upload a pre-recorded video, the video is fetched from local storage and uploaded to the cloud server for processing. Once processed, the translated text is returned to the user. If the user opts for live translation, the app uses the device camera to record gestures in real time, sends the footage to the cloud server, and delivers a live-translated output to the user. For the text-to-sign feature, users input text, which is then processed by the cloud server and converted into animated sign language, which is shown on the screen. Users can also choose their preferred translation language, which the app stores and applies to all outputs. This diagram emphasizes the real-time and backend coordination between the UI, cloud, storage, and device hardware to facilitate seamless translation experiences.

3. Dialogue Diagram (Flow Diagram)

The flow diagram highlights decision points and user guidance, reinforcing learnability and task efficiency. Prompts and error messages ensure users are never lost—following the principle of visibility of system status.

After starting the app, the user is directed to the authentication section. They can either sign up or log in; if credentials are validated, they're redirected to the Home Screen. From there, users can choose between different functionalities like live video translation, uploading a pre-recorded video, or using text-to-sign conversion. If they choose live translation, the device camera checks for lighting conditions and number of signers. Prompts are displayed if poor lighting or multiple signers are detected. Once these issues are resolved, the video is recorded, processed, and output as translated text. For pre-recorded uploads, the app checks the video format and compatibility; invalid files trigger an error prompt, while successful uploads lead to output text. In the case of text-to-sign, the user inputs text, which the system analyzes. If the text contains complex terms, the app alerts the user and redirects to a correction screen; otherwise, it proceeds to animated translation. The user can also navigate to settings to change their language preference, which the system stores for future use. Overall, this diagram provides a user-centered flow showing how each screen and input decision is managed.

4. System Architecture Diagram

Built on a layered modular design, our System Architecture Diagram supports separation of concerns—from UI to logic to data persistence. This promotes maintainability, reusability, and security. Each layer is loosely coupled and highly cohesive, enabling flexibility and real-time updates.

It breaks down the entire SignBridge app into four structural layers: Presentation, App Logic, SignBridge Objects, and Persistent Data. At the top, the Presentation layer includes the UI components such as LoginUI, AdminMainUI, and UserMainUI — the visual interfaces users interact with. These are directly connected to the App Logic layer, which includes components like Authenticator and AccountCreate for authentication, AdminController for admin actions, and UserController for user-related functionalities. The logic controllers then communicate with the SignBridge Object layer, which handles the business logic, such as reviewing content, banning users, removing content, translation verification, and translation processing. These modules also handle text-to-sign, live translation, language selection, and video upload. The lowest layer, Persistent Data, includes the Cloud Server and Local Storage, which store user data, uploaded videos, preferences, and processed translations. The arrows illustrate the direction and relationships between layers, showing how requests travel from the user interface through business logic to the data layer and back. This modular architecture supports scalability, security, and real-time responsiveness.

5. Design Principles

Before we jump into the demo, let's briefly talk about the **design principles** that guided our development process.

First, we followed **AGILE methodology**, which means we built SignBridge in short, iterative cycles. Each sprint focused on real feedback—especially from our Deaf testers—so we could continuously refine the user experience.

Next, we applied the **KISS principle**, or "Keep It Simple, Stupid." Our interface is deliberately clean and focused—no clutter, no distractions—so users can get to the core features quickly and intuitively.

We also made the architecture **modular**, following key **SOLID principles**. Each component, like translation, moderation, or language selection, has its own responsibility. This makes the app easier to maintain, test, and scale—for example, adding a new sign language doesn't break existing features.

Lastly, we embedded **security and privacy by design**. From the start, we included secure login systems, encrypted storage, and clear role separation between users and admins.

Altogether, these principles helped us create an app that's not only powerful and accessible—but also clean, scalable, and secure.

In conclusion, our system design and use cases reflect human-centered design principles—ensuring ease of use, clear feedback, reliable processing, and inclusivity for all users. SignBridge is more than just a translator—it's a thoughtfully engineered platform that bridges communication gaps through usability, accuracy, and design excellence.

How it was deployed in azure:

SignBridge's deployment utilized Azure's ecosystem for scalability and security. The frontend, hosted on Azure App Service, leveraged auto-scaling and CI/CD via GitHub. Backend APIs, including live video processing powered by Azure VMs, were dynamically scaled for cost efficiency. Authentication integrated Mongo DB and API Management for secure user logins. The translation feature routed Google Translate requests through Azure API Gateway. Application Insights and Azure Monitor tracked performance, with user data stored in MongoDB and media blob stored temporarily.The Sign to text model utilized the api hosted from Azure VM and accessed by a secure SSL/TTS Proxy to send the video Blob and retrieve the translated text . This end-to-end Azure approach delivered a resilient, scalable platform for SignBridge's real-time and multilingual capabilities.

We will now be showing a live demo of SignBridge.

Now that we've walked through the system architecture and use cases, let’s dive into how we ensure the reliability of SignBridge through Black Box and White Box Testing.

Let’s start with Black Box Testing, which focuses purely on inputs and expected outputs, without any knowledge of the internal code structure.

As you can see on this slide, we tested core modules like User Account Creation, User Login, Live Translation, Pre-recorded Video Translation, Text-to-Sign, Language Selection, which we will be focusing on just 2 which.
For each module, we prepared realistic input scenarios and validated whether the actual system response matched the expected behavior.

For example, during account creation, entering valid credentials redirects the user to the main UI , matching expectations.
But if the username already exists, or if fields like the password are left empty, the system correctly does not proceed, it stays on the login screen, which prevents user errors and ensures data validation.

We applied similar logic to the login process.
Valid inputs like “Test2” and “TestPassword” log the user in successfully, while invalid or empty inputs result in error prompts or no response, as expected.
These tests confirm that user authentication works as intended, with clear feedback.

Now, let’s move to White Box Testing, White Box Testing allows us to test the internal logic structure of the application which we’ll be covering on User Account Creation and Login.

This flowchart shows the logic behind account creation.
We tested three main paths:

Valid credentials → success
Empty fields → remain on login UI
Duplicate usernames → show error, no account creation

To ensure full path coverage, we calculated Cyclomatic Complexity, which tells us how many independent logic paths exist in the code.

Using Formula 1:
Cyclomatic Complexity = Decision Points + 1
= 2 + 1 = 3

Using Formula 2:
Cyclomatic Complexity = Edges - Nodes + 2
= 7 - 6 + 2 = 3

This confirmed we needed at least 3 test cases to cover all possible logic branches — and we created them accordingly. Taking Path 1->2, 1->3->4->5, 1->3->6->7

We repeated this process for User Login, testing all flow variations like Valid login, Empty field & Invalid password

Each path was mapped, tested, and validated. This gave us confidence that our internal logic is both complete and robust.In Summary With Black Box Testing, we ensured that the app behaves correctly from a user's perspective.With White Box Testing, we confirmed that our internal logic covers all decision branches.Together, these methods helped us ensure SignBridge is both reliable and user-friendly.

Live Translation

After successfully logging in, the user navigates to the "Live Translation" feature from the main UI. Upon clicking the "Turn On Camera" button, the system activates the webcam and displays the live camera feed directly on the interface, confirming successful access. The user then clicks "Start Translation" and begins performing sign language gestures. The system processes these gestures in real-time, converting them into corresponding textual output that appears on the screen. This confirms the core functionality of live translation. After the session, the user clicks "Turn Off Camera," and the video feed is cleanly terminated, reflecting correct system deactivation of the camera resource. To test language flexibility, the user selects the "Language" icon, which redirects them to the language selection interface as expected. Finally, the user clicks “Back to Home” and is redirected back to the SignBridgeMain UI, ensuring seamless navigation.

Pre-Recorded Video Upload

Next, the user accesses the "Pre-Recorded Video Upload" feature. They click "Select Video" and upload a valid MP4 recording. Upon selecting "Translate Video," the system successfully processes the file and converts the sign language into corresponding text, demonstrating correct behavior. To test error handling, the user uploads an invalid or corrupted file and again clicks "Translate Video." As expected, the system returns an error message—“Failed to fetch”—indicating that the system properly identifies and blocks invalid inputs. Lastly, the user clicks the “Back” button and is redirected to the SignBridgeMain UI without issue.

Text-to-Sign

The user then tests the "Text-to-Sign" feature. They input a valid sentence, “Hello I love you,” and press “Translate.” The system successfully converts this text into its sign language representation, confirming proper function. Next, the user enters a string composed solely of numbers, “12345,” and clicks “Translate.” The system does not display any output, and the user remains on the same UI, reflecting the expected behavior for invalid inputs. The user tests again by leaving the input field completely blank and pressing “Translate.” As with the previous invalid input, no output is generated, and no prompt is displayed. The user finally clicks “Back,” and the application returns to the SignBridgeMain UI.

Language Selection

In the "Language Selection" UI, the user first selects “Chinese.” Instantly, the application updates all interface text to Simplified Chinese, confirming language support. The user then

clicks “English,” and the UI reverts to English text as expected. Finally, the user selects “Back to Home,” and the system navigates smoothly back to the SignBridgeMain UI.

Sidebar Navigation

Lastly, the sidebar functionality is tested. From various interfaces, the user clicks on different sidebar options. Clicking “SignBridgeMain” navigates the user back to the home UI regardless of the current page. Selecting “LiveTrans” brings up the Live Translation interface correctly. Choosing “Pre-Recorded Video Upload” navigates to the corresponding UI as intended. Selecting “TextToSign” also works as expected, loading the text-to-sign translation screen. Finally, clicking on “LanguageSelect” reliably redirects the user to the Language Selection UI. All sidebar navigation functions perform consistently and reliably across different app states.