

# Sign language translation web application

Oriane Cortes, k12348101 (Erasmus – Software engineering)  
Lukas Gattermayr, k11911639 (AI Msc)  
19. June 2024



## Documentation

This file contains the documentation of all committed files to the github repository, including all folder containing images, the source code for the frontend written in JavaScript, the backend written in Python, as well as supporting files, such as configuration files to set up the environments. It also contains also contains the supporting plan of the project, including the goal of the system, domain model and many more.

## Inhalt

<a href="#">Goal of Your System.....</a>	<a href="#">1</a>
<a href="#">Requirements.....</a>	<a href="#">1</a>
<a href="#">Use Case Descriptions.....</a>	<a href="#">2</a>
<a href="#">Use Case Diagram.....</a>	<a href="#">14</a>
<a href="#">Implemented Use Cases:.....</a>	<a href="#">14</a>
<a href="#">Traceability Matrix.....</a>	<a href="#">15</a>
<a href="#">Domain Model.....</a>	<a href="#">17</a>
<a href="#">Architecture Diagram.....</a>	<a href="#">17</a>
<a href="#">Components Description.....</a>	<a href="#">17</a>
<a href="#">File documentation.....</a>	<a href="#">20</a>

## Goal of Your System

The system is designed to be a web application to translate sign language into text. The application enables the user to upload images of signs that correspond to the official American sign language (<https://www.nidcd.nih.gov/health/american-sign-language>) and translate it to the corresponding alphabet letter.

The system also ensures the ability for the user to validate the translated text, with a default or „standard“ image of a corresponding image to the predicted letter, for the user to have a better understanding whether the correct letter was translated when comparing the uploaded user image and the default image.

Furthermore, small customizations of the interface can be done by the user, to increase the user friendliness of the application.

The whole project is set up as a pilot project and is therefore only able to run locally at the moment, which can be further changed in the future to be accessible via the internet without needing to set up specific environments.

## Requirements

### System requirements:

**NfReq<ID1> The system shall seamlessly integrate with hardware components such as cameras for capturing live videos or images and with the device's image gallery for uploading images.**

NfReq<ID2> The system shall have a fast response time for translating, with a maximum latency of 2 seconds.

NfReq<ID3> The system shall maintain high translation accuracy, with a minimum accuracy rate of 90%.

NfReq<ID4> The system shall comply with relevant standards and regulations for accessibility, privacy, and data protection

NfReq<ID5> The system shall support translation into multiple languages.

NfReq<ID6> When pressing the save button the system shall save images and memories for later.

## Use Case Descriptions

Use Case : Identify Sign	
ID	UC1
Description	From an image or video, translates the sign into text
Actors	The application user and the artificial intelligence model responsible for sign recognition
Stakeholders	Users, instructors and regulators
Pre-conditions	The system must have access to the device's camera
Success and condition	The sign is correctly identified
Failure and condition	The sign is not recognized or mistranslated

### Main Success Scenario

1	The user make a sign visible for the camera	
2	The system accesses the video feed	
3	The Image Processor Model analyzes the video feed	
4	System displayed the translated sign as words	

### Alternative Scenario

1	The user upload an image	
2	The system accesses the image	
3	System translate the image and send it back	
4	System displayed the translated sign as words	

### Exception Scenario

1	The user make a sign visible for the camera	
2	There is a problem with the connection	
3	The system ask the user to do it again	

Use Case : Identify Translation	
<b>ID</b>	UC2
<b>Description</b>	From an image or video, translates the sign into text
<b>Actors</b>	The artificial intelligence model responsible for sign recognition
<b>Stakeholders</b>	Users, instructors and regulators

<b>Pre-conditions</b>	The system must have received a video or photo to enable it to carry out its analysis
<b>Success and condition</b>	The translation is correct
<b>Failure and condition</b>	The translation is wrong or did not take place

### **Main Success Scenario**

1	The image is sent to the AI model	
2	Data is inputted to the model	
3	Model processes the data	
4	Model outputs the prediction	

### **Alternative Scenario**

1	User uploads a video showing a sequence of signs	
2	Video is cut into frames and sent to the model	
3	Model outputs the prediction for every frame	

### **Exception Scenario**

1	Sent image is in the wrong format	
2	Model can not process data	
3	Model throws an exception	

<b>Use Case : User validation</b>	
<b>ID</b>	UC3
<b>Description</b>	Ask the user to validate the translated sign

<b>Actors</b>	User and system
<b>Stakeholders</b>	Users, instructors
<b>Pre-conditions</b>	System sent back image after translation
<b>Success and condition</b>	User validates the image
<b>Failure and condition</b>	User does not validate the image

### **Main Success Scenario**

1	Systems sends the user an standard image of the predicted sign	SUC1
2	Image corresponds to the sign the user made before	
3	User validates that the system correctly identified the sign	SUC2

### **Alternative Scenario**

1	The system sends the user an standard image of the predicted sign	
2	Image does not correspond to the sign the user made before	
3	The user does not validate the image	
4	System asks the user to make the sign again	

### **Exception Scenario**

1	There is no standard image for the predicted sign	
2	System throws error that the image is missing	

3	This error is indicated as a message for the user	
---	---	--

Use Case : Image memory	
<b>ID</b>	UC4
<b>Description</b>	Store the uploaded images of the user
<b>Actors</b>	User and system
<b>Stakeholders</b>	Users, instructors and regulators
<b>Pre-conditions</b>	User has to provide image data
<b>Success and condition</b>	Images are stored in the application and are visible for the user
<b>Failure and condition</b>	Images are not stored and not accessible afterwards

### Main Success Scenario

1	After an image is successfully validated, the system asks the user to save the image and corresponding translation	SUC3
2	The image and text are saved in the system	
3	Saved images and translations can be used to form sentences	SUC4

### Alternative Scenario

1	After an image is successfully validated, the system asks the user to save the image and corresponding translation	
---	--	--

2	The user does not want to save the image and translation	
3	Image and translations are not saved	

### **Exception Scenario**

1	The user does not validate the the translation within 60 seconds	
2	Image and translation are deleted and not saved	

Use Case : Interface settings	
<b>ID</b>	UC5
<b>Description</b>	Ability for the user to customize the application interface
<b>Actors</b>	User and system
<b>Stakeholders</b>	Users, instructors
<b>Pre-conditions</b>	Open the application
<b>Success and condition</b>	Interface changes depending on the user action
<b>Failure and condition</b>	Interface does not change or changes undesirable

### **Main Success Scenario**

1	The user navigates to the settings	SUC5
2	The user changes the interface	

3	The user saves the modifications	
4	Modifications are applied	

### Alternative Scenario

1	The user navigates to the settings	
2	The user changes the interface	
3	The user does not save the modifications	
4	Modifications are not applied	

### Exception Scenario

1	The user changes the interface	
2	The user does not choose between saving or not saving the changed settings for 60 sections	
3	Modifications are not applied	

Supporting Use Case : Standard image	
<b>ID</b>	SUC1
<b>Description</b>	Each sign attains a saved standard or default image that is sent after prediction for the user to validate if this sign was made
<b>Actors</b>	User and system
<b>Stakeholders</b>	Users, instructors
<b>Pre-conditions</b>	Standard image saved for every available translation
<b>Success and</b>	Standard image is displayed for user to verify



<b>condition</b>	
<b>Failure and condition</b>	No standard image can be shown

### **Main Success Scenario**

1	Standard image is saved already saved for every available translation	
2	The standard image is displayed for the user after every translation	

### **Alternative Scenario**

1	Standard image is saved already saved for every available translation	
2	User decides after translation if the standard image should be displayed	
3	Depending on the user's decision, the image is displayed or not	

### **Exception Scenario**

1	Standard image is not saved already saved for every available translation	
2	Error message shown that no default image is available for validation	

### **Supporting Use Case : User validates standard image**

<b>ID</b>	SUC2
<b>Description</b>	After every translation the user can verify if the made sign corresponds to the standard image corresponding to the predicted sign

<b>Actors</b>	User and system
<b>Stakeholders</b>	Users, instructors
<b>Pre-conditions</b>	Standard image displayed after every translation
<b>Success and condition</b>	User verifies the image
<b>Failure and condition</b>	User does not verify the image

### **Main Success Scenario**

1	User verifies that the standard image corresponds to the made sign	
2	Correct translation and user made sign image are saved if needed	

### **Alternative Scenario**

1	User does not verify the correct sign	
2	User is asked to redo the wanted sign, until the translation corresponds to the made sign	
3	When the correct sign is shown, image and translations can be saved	

### **Exception Scenario**

1	User does not validate the correct image for 60 seconds	
2	Image and translation is not saved	

### **Supporting Use Case : Sign image and translations saving**

<b>ID</b>	SUC3
-----------	------

<b>Description</b>	User made image of sign and corresponding translations saved in the system
<b>Actors</b>	User and system
<b>Stakeholders</b>	Users, instructors
<b>Pre-conditions</b>	Successful validation of the corresponding standard image for the sign and the made sign and demand to save both
<b>Success and condition</b>	Image and translation successfully stored
<b>Failure and condition</b>	Image and translation not stored

### **Main Success Scenario**

1	User presses button to save both image and translation after successful validation	
2	Image and translation successfully stored	

### **Alternative Scenario**

1	User does not want to save image and translation	
2	Nothing is saved, systems goes back to default	

### **Exception Scenario**

1	Error occurs while trying to save the image and translation	
2	Image and translation is not saved, system	

	goes back to default	
--	----------------------	--

Supporting Use Case : Sentence creation	
<b>ID</b>	SUC4
<b>Description</b>	Every validated translation is saved and can be displayed sequentially to form sentences
<b>Actors</b>	User and system
<b>Stakeholders</b>	Users, instructors
<b>Pre-conditions</b>	Successful saving of all previous translations and images
<b>Success and condition</b>	Sentence is formed using the sequentially translated signs using words and saved images
<b>Failure and condition</b>	Sentence is not correctly displayed

#### **Main Success Scenario**

1	User has the ability to display the previous translations and images	
2	Previous translated words form a sentence and previously saved images form a sequence of corresponding signs	

#### **Alternative Scenario**

1	User does not want to form a sentence using the previous translations	
2	Only the current translation is displayed without the context of the previous ones	

### Exception Scenario

1	Missing previous images or translations	
2	Incorrect sentences and image sequences are shown	
3	User can delete the whole incorrect sentence	

### **Supporting Use Case : Settings customization**

<b>ID</b>	SUC5
<b>Description</b>	Ability to customize the system appearance, including colors, font styles, font sizes etc...
<b>Actors</b>	User and system
<b>Stakeholders</b>	Users, instructors
<b>Pre-conditions</b>	Pressing button to navigate to the settings
<b>Success and condition</b>	Changes in settings successfully saved
<b>Failure and condition</b>	No changes made in system appearance

### Main Success Scenario

1	User navigates to the settings by pressing a button	
2	Changes saved	
3	Appearance of application changes depending on the new user settings	

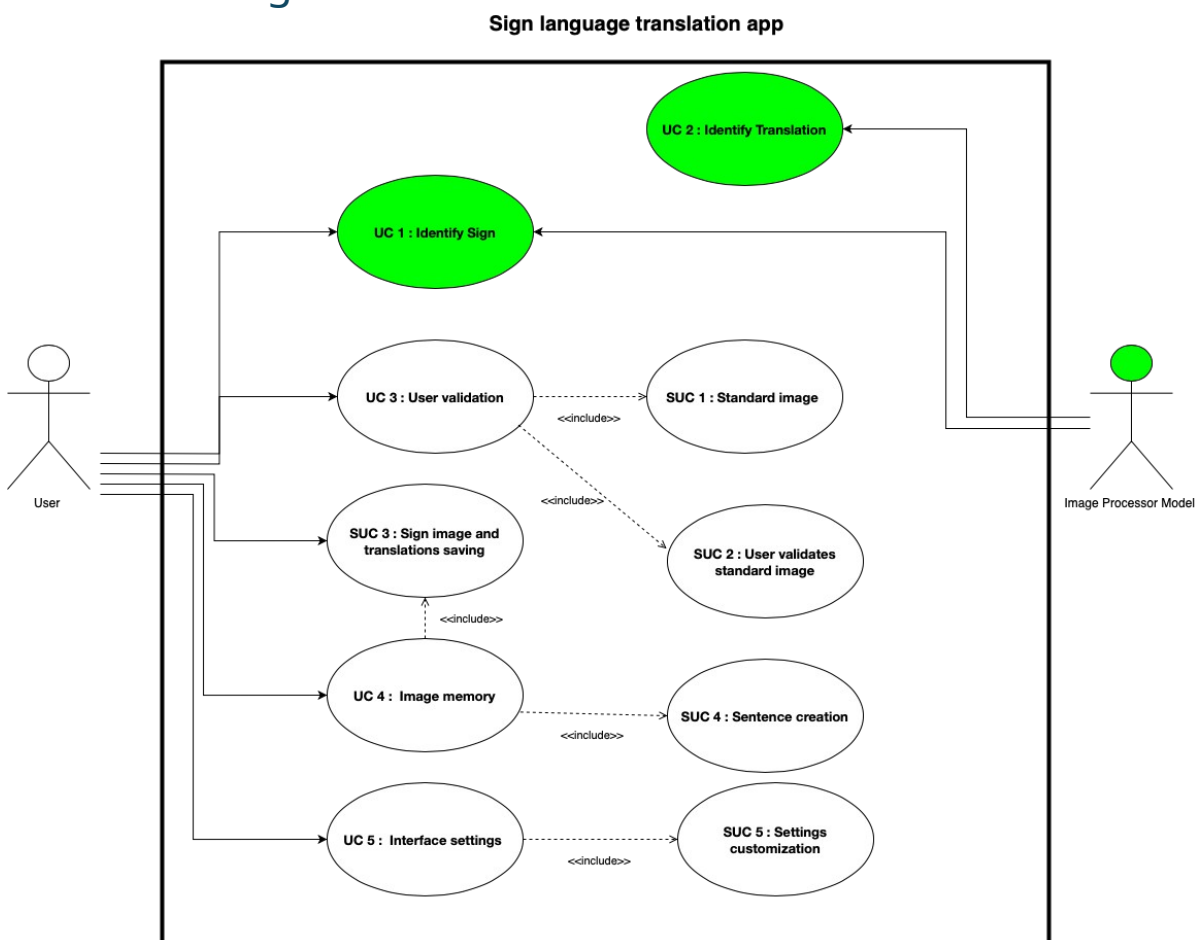
### Alternative Scenario

1	User navigates to the settings by pressing a button	
2	Changes are not saved	
3	Appearance does not change	

### Exception Scenario

1	User navigates to the settings by pressing a button	
2	Settings are changed, but the user does not decide to save or not to save within 60 seconds	
3	No changes are made, appearance stays the same	

## Use Case Diagram

































## Implemented Use Cases:

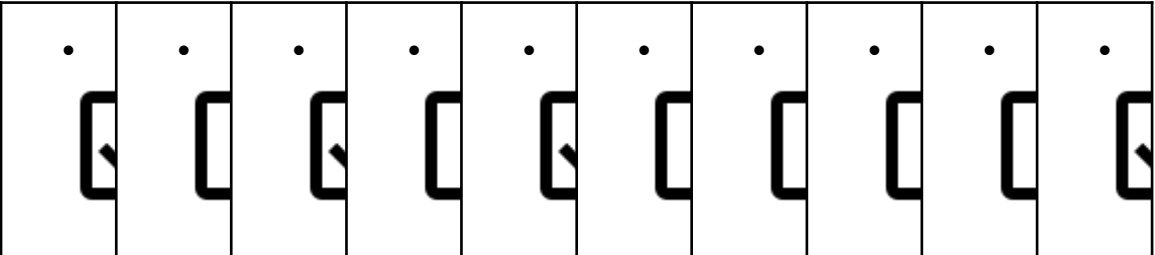
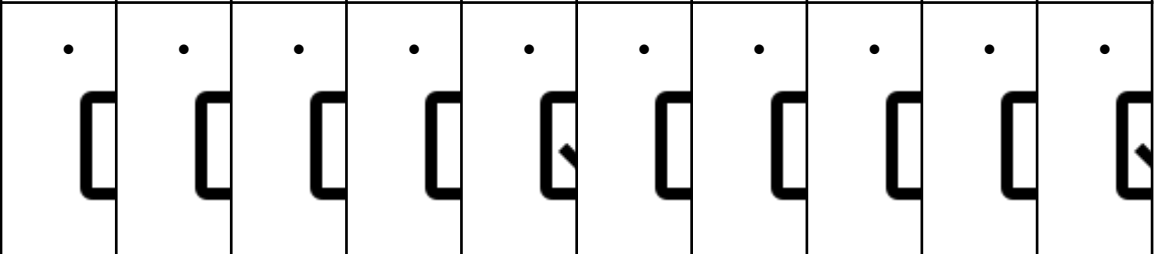
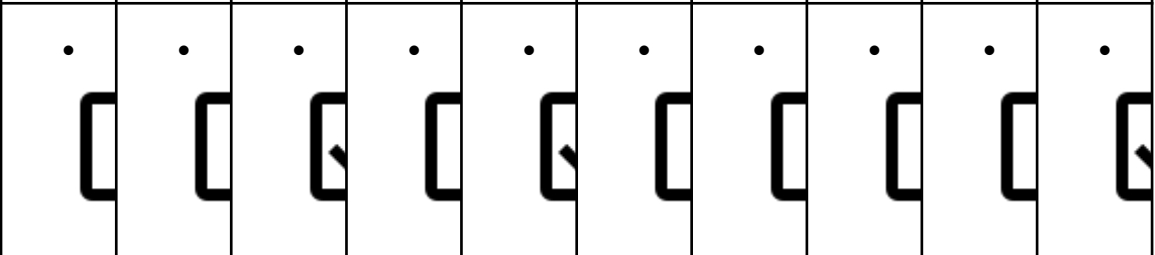
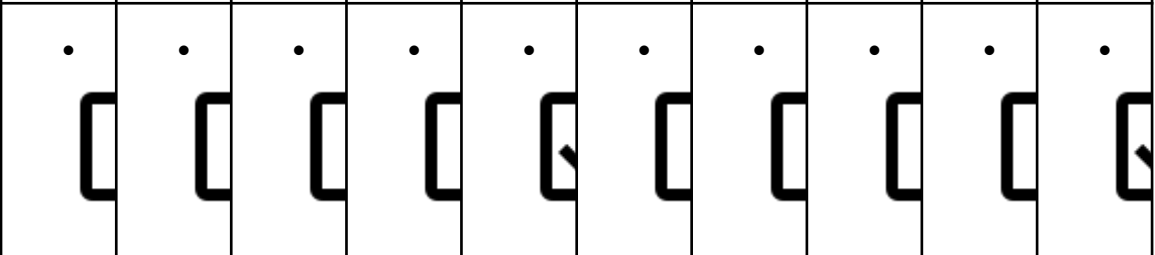
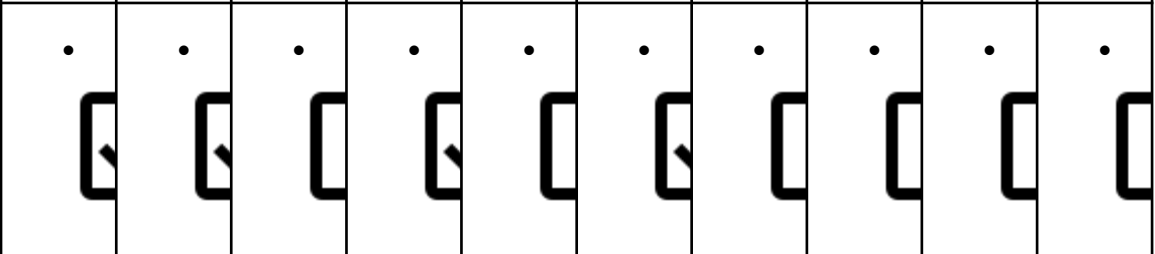
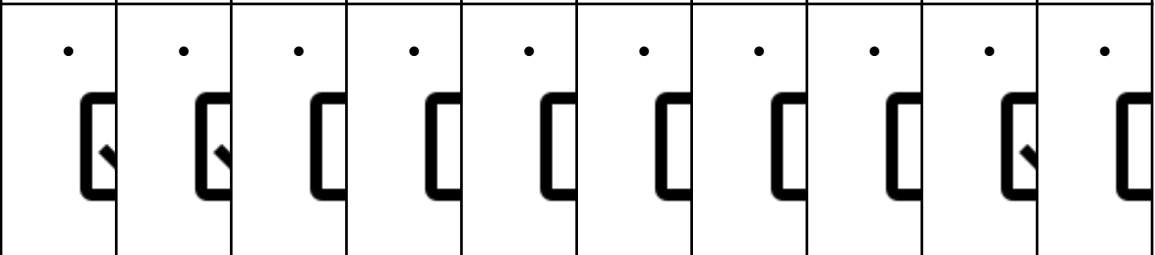
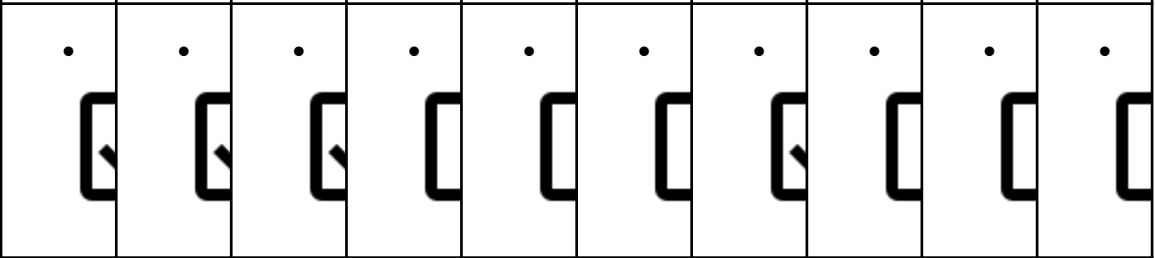
In the source code of the project, so in the prototype version all use cases except the ability for the user to save its translation and create sentences are implemented which can be implemented in further versions of the system.

Implemented use cases: UC1, UC2, UC3, UC5































Implemented supporting use cases: SUC1, SUC2, SUC5

## Traceability Matrix

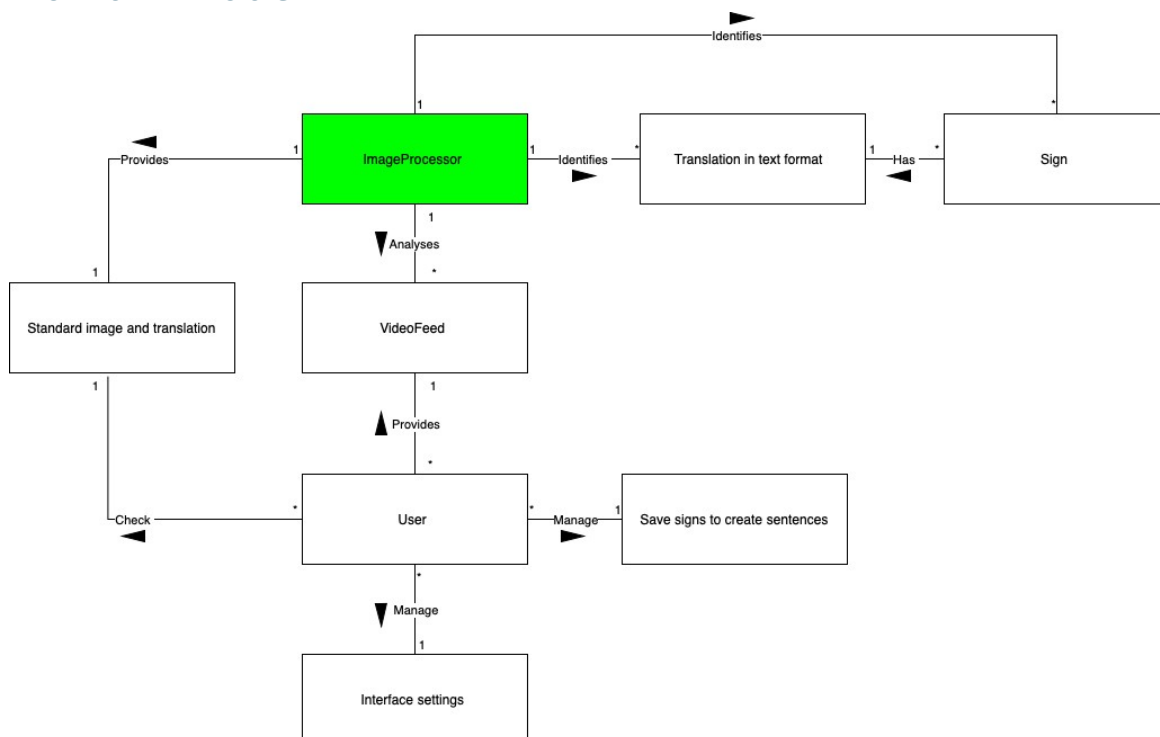
Use Cases	UC1	UC2	UC3	UC4	UC5	SUC1	SUC2	SUC3	SUC4	SUC5
Requirements										
Req1										
Req2										
Req3										

Req4	
Req5	
Req6	
Req7	
NfRe q1	
NfRe q2	
NfRe q3	

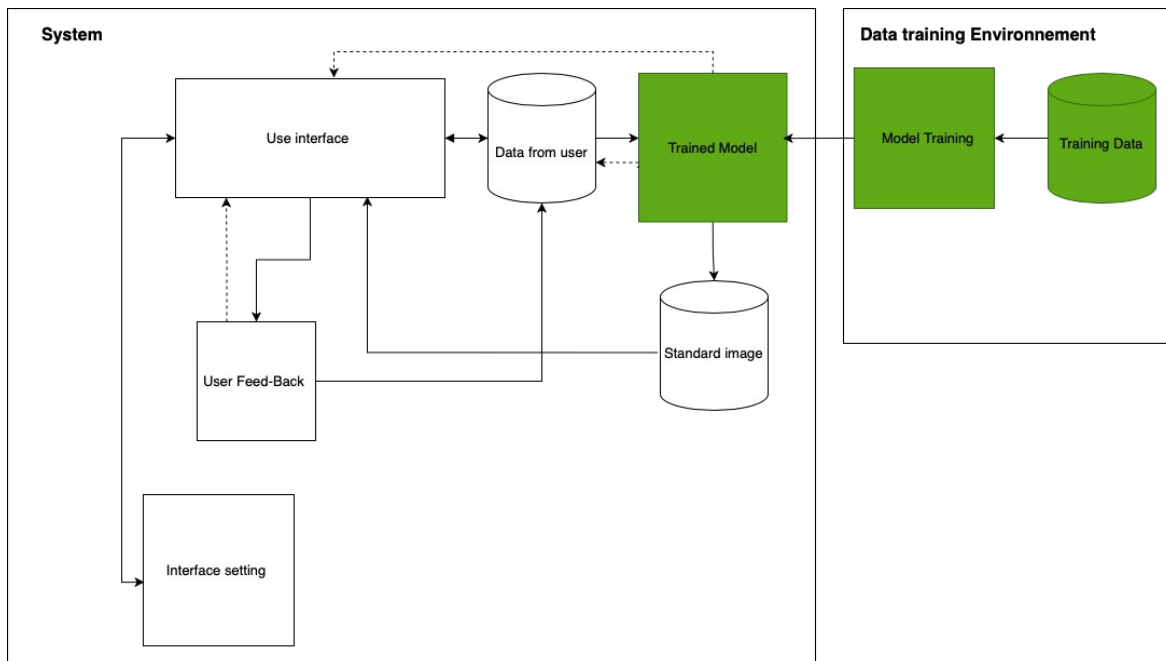


<b>NfRe q4</b>										
<b>NfRe q5</b>										
<b>NfRe q6</b>										

## Domain Model



## Architecture Diagram



## Components Description

**System:** The system is just the place where all the programs the user is going to use in the application.

1. **User Interface:** Default page of the application, user can either access the interface settings or upload an image. After an image is uploaded, the interface shows the user the translated sign and a standard image corresponding to this translation with the possibility for the user to validate the correctness.

**Requirements : ID1 ; ID2 ; ID4 ; ID6**

2. **Data from User:** After the user uploads an image, the image is saved in a user image database and subsequently sent to the model to predict the sign. Depending on the feedback, the images stay saved with the translation or are deleted. The user can access the upload and translation history to build sentences or delete all if the user wants to.

**Requirements : ID1 ; ID2**

3. **Trained model:** The trained model is the finished ML model, trained in the outside data training environment. It uses the uploaded images from the user that are saved in the user data to create a translation, this translation is then used to retrieve the corresponding standard image for this translation. If the uploaded image is not accepted by the model, the user is informed and the saved image is deleted.

## **Requirements : ID2 ; ID3**

4. **Standard Image:** Database, where reference images are stored. After a translation is done by the trained model, the corresponding reference image is retrieved and sent to the user interface.

## **Requirements : No**

5. **User Feedback:** When the user gains the reference or standard image, the user can validate the standard image and save it with its corresponding translation in the user database. If the translation is wrong, the saved uploaded image from the user is deleted and the user is asked to try it again with a different image. If the user does not validate for 60 seconds, the image and translations are automatically deleted.

## **Requirements : No**

6. **Interface settings:** In the user interface, the user has the ability to navigate to the interface settings. Here the user can customize the interface and text as much as possible and the user can decide whether to save the new settings or not. If the user does not decide to change or leave the settings for 60 seconds, they are automatically deleted.

## **Requirements : ID4 ; ID5 ; ID6 ; ID7**

**Data training environment:** This is the outside environment to train the model, before using it in the system.

1. **Training data:** Pre-defined dataset used to train the model.

## **Requirements : It's outside of the system**

2. **Model Training:** The component that uses the training data to train the model to translate images to sign languages.

## **Requirements : It's outside of the system**

## **Design Questions and Answers**

### **Questions:**

1. What accuracy does the translation model have to have?

2. How do we store the data, which database type we use?
3. Can the model handle different user skin colors?
4. Can the application support mobile usage as well?
5. Is the usage of the application easy to understand for the user?
6. Is it possible for the user to be informed about the data usage?
7. Should the user be informed how the model works?
8. Can the model continuously be improved with new user data?
9. Should there be a minimum age for a user to use the application?
10. Is it possible to update the whole application?

#### Answers:

1. The translation model has to attain 65% accuracy.
2. We store the data locally in data folders
3. Yes, as the used training data also contains different skin colors and need to be converted to greyscale for the model to use it.
4. The application at first only is designed to work locally via the browser, which could be extended for mobile usage on a website.
5. The system interface will be designed very user-friendly and visually appealing to make the whole application easy to understand.
6. It is possible for the user to gain information about the user's data usage from the user interface.
7. We think that it is important for the user to have the possibility to understand how the model works, so we are going to implement a small explanation about the model.
8. At first the model is trained using a predefined database and can not be continuously improved.
9. There is no reason to create an age restriction.
10. It is something that we want to be possible in the long term, to improve the model accuracy by retraining and fine tuning the model on more data.

## File documentation

### **default\_images folder:**

This folder contains the default images that correspond to a given alphabet letter, which are used for the user to be able to validate the translation to a certain extent.

### **documentation folder:**

This folder contains this documentation pdf file.

### **models folder:**

Here the CNN ML models are stored that can be used for translation. At the moment only one fully trained model is accessible and used in the application.

**.gitignore:**

This is the classical git file to ignore temporary and unnecessary files.

**ReadMe.md:**

Here the documentation to install all the dependencies, create the right environments and start the frontend and backend server is accessible.

**environment.yml:**

This environment file is used for conda to create the correct environment for the backend python server.

**index.html:**

This file contains all the html code used in the frontend interface.

**network.py:**

This python file is used to load the model skeleton before loading the saved model state dict from the model folder. It was also used during training in the training environment to load the model and freeze different model parameters.

**package-lock.json and package.json:**

This file contains all the dependencies used in the frontend to run the server, these are created by first install the node package manager.

**Project.ipynb:**

This jupyter notebook was used as a separate training environment to train the model on the official American Sign Language MNIST dataset (<https://www.kaggle.com/datasets/datamunge/sign-language-mnist>). It is separate from the running application and is not needed to run the web application.

**scripts.js:**

This file contains all the JavaScript code and logic that is happening in the frontend of the application.

**server.py:**

This file is used to start the Flask (<https://flask.palletsprojects.com/en/3.0.x/>) backend python server, that serves as an API that receives the image from the frontend and returns the classified alphabet letter.

**styles.css:**

This file contains all the css code used to format and design the web interface.

**utils.py:**

This file contains all the helper functions used in training the model. It is also not used in the web application only in the training environment.