Qazaq Sign Language. Bridging Communication Barriers

*This is a Senior Project on Qazaq Sign Language translation

1st Maksat Faizuldanov, Kamila Arslanova, Bekzhan Bolat, Zhansultan Assemzhar, Daniyal Bayanov
Dept. Computer Science, Nazarbayev University, Astana, Kazakhstan
maksat.faizuldanov@nu.edu.kz, kamila.arslanova@nu.edu.kz, bekzhan.bolat@nu.edu.kz,
zhansultan.assemzhar@nu.edu.kz, daniyal.bayanov@nu.edu.kz

***Abstract*—The project is aimed to address the issue of communication barriers encountered by individuals with hearing disabilities, given the scarcity of resources in sign language for people in Kazakhstan. The project tackles the issue by developing a web application that provides the translation of spoken language (Kazakh, English, and Russian) to Kazakh-Russian signed language. The project aims to facilitate accessibility and communication of the dead community by utilizing Language Technology, specifically the Sign Language Synthesis, Sign Language Translation, and Natural Language Processing. Additionally, the web application expands educational opportunities of the individuals with hearing disabilities by providing a book library, which could be also translated to sign language.**

***Index Terms*—sign language, sign language translation, natural language processing, Kazakh signed language, hearing disabilities**

I. Introduction

According to the World Health Organization [11], more than 5% of the population has hearing loss. The major issue this leads to is the disconnectedness of the people with hearing disabilities with the rest of the world, since it poses a significant obstacle to information sharing - it becomes challenging to perceive information between individuals. Today, there are only two ways of communication between the hearing people, and deaf community – text and sign language. The fact that the vast majority of hearing people do not have any experience in sign language exacerbates the issue even more, as written language could be difficult to comprehend for deaf people [3]. Additionally, it causes further adverse effects such as the challenge of limited opportunities in education since the vast majority of educational and scientific resources are typically presented in written form.

Nevertheless, today the issue of deaf community disconnectedness can be tackled by Language Technology, a field related to the computational analysis of human languages. The aim of The Sign Language Synthesis and Sign Language Translation areas in this field is the translation of the natural language to sign language [1]. Our project's purpose is to include these areas to counter the obstacles posed to the community of people with hearing disabilities by means of developing a web application that is based on generating animations of sign language translation from spoken language (text). In our project, the Sozder [8] database is used, which consists of words and links them to short videos and their translation in sign language.

Our project is aimed at solving the problem of the communication barrier between the hearing and deaf communities, while also expanding the educational opportunities for people with hearing disabilities. The project addresses both issues, as it allows the user to translate textual spoken language to sign language. Additionally, the web application provides a library of books which facilitates deaf people to read, since the books can also be translated into sign language word by word.

The first section of this report includes background information and previous research done over the topic. The next section discusses how we approach to solve the problem, which tools we used and general structure of the web application's components. After that, we describe how the project was actually implemented and which problems we encountered during the work. Lastly, we evaluate the project based on the results of the user testing.

II. Background and Related work

Sign Language Processing is the area in Language Technology, which is a field of the computational processing of human languages. The Sign Language Synthesis and Sign Language Translation areas of this discipline together aim to translate natural language to a sign language [1]. The research related to use of technology for translating spoken language to sign language dates back to recent years. The area is relatively new, and started developing in the last decade [5]. There exist several papers on this topic, covering both sign language recognition and generation.

In Kazakhstan, one of the projects that involves information technology in the area of sign language is a sign language recognition project that uses a neural network. The project was developed by a faculty member of Eurasian National University, and its goal is to help people to learn sign language and prepare future interpreters. The sign language generation from spoken language is a future plan of this application [2]. Another contribution by Kazakhstani researchers is the datasets for Kazakh Sign language, which will be discussed later.

Our inspections over the available literature showed that sign language generation has not been addressed yet in Kaza-

khstan comprehensively. For our work, we rely on several previous works done to develop our own system of sign language generation as below:

A. Sign.mt [4]

Sign.mt is an open-source project that functions as a translation from spoken language into sign language and vice versa. It is implemented as a website which allows users to translate text input to a video. The website includes various spoken and signed languages, such as German, English and some others, but does not cover Kazakh sign language. As for our project, we will use this work as a reference to implement a similar idea, but for Kazakh sign language.

B. "Sozder" video collection [8]

The Sozder video collection is a dictionary dataset which contains translation of words to sign language by categories. It contains translations for 1570 words in Kazakh and Russian. The video materials will be converted into .POSE files to be used later in our translation service.

C. Pose-Format library [6]

The pose-format is a Python library developed by Moryossef et al. It was developed with the purpose to contribute to the development of Sign Language Processing and provides tools to work with poses in .pose format. This library uses Mediapipe to estimate body positions from a video. For our project, the library allows us to convert an mp4 video to the .pose file, which contains such information as the pose points, their positions and connections between points. The resulting .pose file can then be converted into a video animation. This approach enables the anonymization of sign language interpreters. By using this library, we convert the videos in the Sozder collection into a database of corresponding .pose files.

D. Gloss-Based Pipeline for Spoken to Signed Language Translation [7]

This work represents a pipeline of translation from spoken language to sign language: text-gloss-pose-video. It relies on the Pose-Format library discussed above and includes German, French, and Italian sign languages, converting the grammar structure of sentences in spoken language to glosses, for each of which a corresponding .pose file is selected. The output of this pipeline is the sequence of .pose files (representing words in a sentence) that are concatenated smoothly and can be converted into a video. In the case of our project, we use this work to implement normalization and concatenation of poses that we previously generated from interpreter's videos in the Sozder database.

E. Kazakh Sign Language Datasets

With the perspective of future work, more comprehensive video databases are available for now in Kazakh Sign Language and can be used. First one is KRSL-OnlineSchool: Large Vocabulary Kazakh-Russian Sign Language Dataset [9], which is based on recording of 890 hours of video material from El-arna TV's sign language-interpreted lessons. The dataset includes translations by 7 interpreters with annotations for approximately 300 hours of video, and transcriptions for 4,009 out of 4,547 lessons using automatic speech-to-text software.

Another dataset is covered in paper done by Mukushev et.al. [10] with 28250 videos. The work focuses on distinguishing similar signs in Kazakh-Russian Sign Language (K-RSL) based on non-manual components. The study assesses the impact of non-manual features, such as question signs.

F. Video Generation Models

There are several papers focusing on image and video generation models for sign languages. Some methods include the Style-Based Generator Architecture for Generative Adversarial Networks, Variational Diffusion Models, High-Resolution Image Synthesis with Latent Diffusion Models, High Definition Video Generation with Diffusion Models, and High-Resolution Video Synthesis with Latent Diffusion Models [5].

Other work related to stylized sign language generation is done by Yu et al. [12], which provides a holistic motion dataset and benchmark for 3D animated sign avatars.

III. Project Approach

A. Web Application Fuctionality

Our solution to the problem of communication barrier is the web application that lets you translate the spoken language text for any of Kazakh, English, and Russian language to the Kazakh (Qazaq) Sign Language [further QSL]. For translation purposes, our own translation service API has been developed. User of website is allowed to:

1) Dynamically enter the text into the input box and get real time translation to QSL.
2) Trigger the translate function by clicking on specific words within the content of the book to read it in QSL. Real-time translation to QSL of the clicked word also occurs.
3) View the translation of the interface elements by hovering on the certain element.
4) All processes above lead to the display of a GIF file containing a stick figure demonstrating signs.
5) To register and log in, leading to the functionality of adding the books in their own library ("My books" section).
    a) Opportunity to save progress and read the added book from the last read page.

B. High-level Structure

The general architecture of our system is Model-View-Controller as shown below on Figure 1.

Our project could be divided into 3 major parts, as shown on Figure 2:

- Front-End (JavaScript, ReactJS, CSS, Tailwind CSS, NPM)

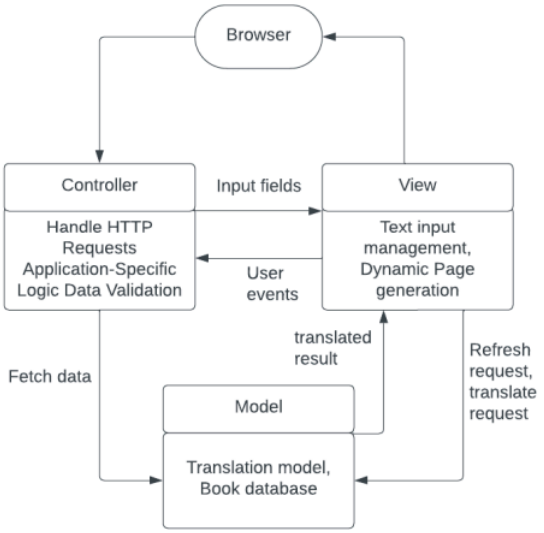

Fig. 1. MVC Architecture of the Website

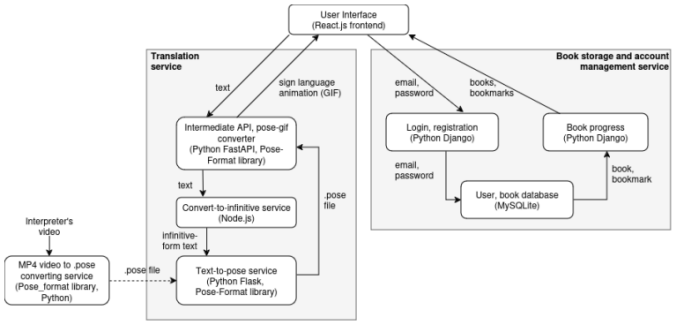

Fig. 2. General Structure of the System

- Back-End (Python, Django, Django REST, SQLite3) for book reading progress keeping and managing user accounts
- Our own translation service API consists of a few microservices (Python, JavaScript, NodeJS, Flask, FastAPI).
- In addition, we implemented a service outside of the system to convert all videos in the Sozder database to .pose files that are uploaded to the translation service (which is a part of the website).

C. Detailed description of architectural components

1) Front-End: Front-End part focused on developing five different pages, which are named "Home", "Translate", "Books", "Help", "About us".

- "Home" is the first page that users will see. It will help them to navigate throughout the web application.
- "Translate" is the page where users can input their texts and receive an output in the form of an animated sign language translation.
- "Books" is the section where users can choose and read a book by translating certain words in real time and view the result in the form of the animated stick figure demonstrating signs.
- "Guide" includes guidelines which users can utilize to understand how the system works.
- "About us" page gives users information about the project, its mission and the team behind its development.

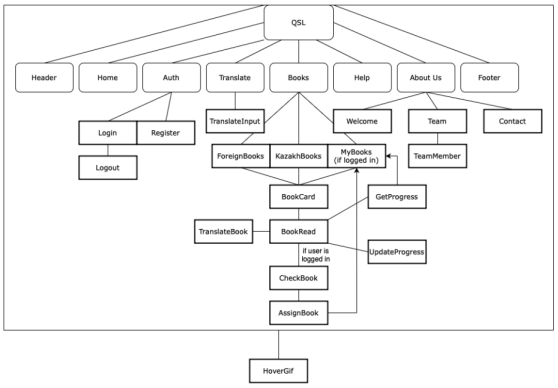

Fig. 3. Component-based structure representation.

For front-end development, the decision was made to utilize JavaScript's React library and adopt a component-based architecture. Additionally, to enhance the UI/UX experience, Tailwind CSS has been incorporated for styling the web application. Given the component-based nature of our project, the components can be described as follows:

The figure 3 briefly describes the structure of the component based architecture. To be precise, it might be noted that there are 8 main "parent" components that have few child components under them, as described in the figure. To discuss the complex ones:

- **Auth.** Component created to handle user log in, registration, log out processes. If the user is logged in, then there is the rendering of the Logout component.
- **Translate.** The component developed to dynamically accept the input in textarea and translate it. TranslateInput child component is responsible for handling the interaction of users and sending requests to the back-end server and our own developed translation service API endpoint.
- **Books.** Most complex page that consists of parent and many children components with conditional rendering. Since our library displays the books for Foreign Literature, Kazakh Literature, and conditionally MyBooks, we have separate components for each of them.
  - To handle the displaying of a lot of books, a BookCard component was implemented. By passing the array of objects, we could achieve the rendering of the same book cards with different information.
  - The read button of the book card triggers the render of the BookRead component. That component lets the user view and read the full content (text) of the book in a separate box.
  - Meanwhile, if the user is logged in at the moment of clicking the read button, then the book's addition

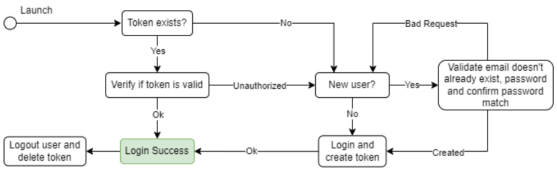

Fig. 4. Architecture diagram of back-end auth process

to the user's books section is triggered.

- ∗ Firstly, the book is checked to appear or not in the user's database to avoid duplicates (CheckBook).
- ∗ If the book is not present in the user's books database, then it is assigned to the certain user. The AssignBook component is responsible for that function.

– In the BookRead component, during the reading, if the user clicks the word, then the real-time translation of the word is triggered, that lets us translate the book. (TranslateBook component).
– After leaving the read page, if the user is logged in, then the data about the progress, specifically the last read page of the user is passed to the 'MyBooks' section. Later users will be able to read the book from the last read page, letting them to avoid the loss of the progress.

- **About Us.** The page was implemented to tell the users about our mission. The page is mostly static, therefore contains 4 child components that are responsible for the information provided.
  – Welcome – responsible for welcoming users and providing information about the mission of the project.
  – Team and TeamMember – components to display the team members as the names state. TeamMember is the component that handles displaying the same structured cards with different information about team members.
- Other components are not as complex as the ones mentioned above. Mostly, they are static components that display static information.
- **HoverGif.** Special and different component that is implemented independently. Its main goal is to display the sign language translation GIF file for each of the interface elements when that certain element is hovered by the user's mouse.

2) Book Storage and Account Management service: For the backend part we used Django Rest Framework of Python programming language, and for storing the user information and all about books such as content, progress etc. we used SQLite3.

- User authorization works using a token, for example, if the user has already logged into the site (a token is created) and simply closed it, then it is saved for a certain time. The token is deleted after the time has expired or if the user decides to log out.

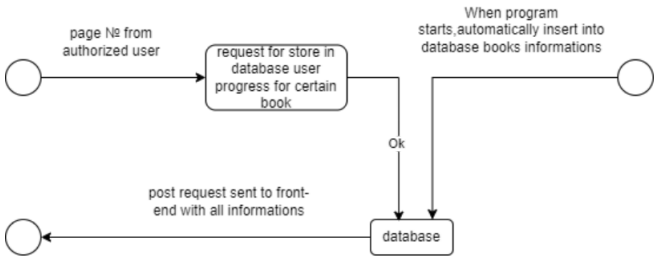

Fig. 5. Architecture diagram of books database interaction

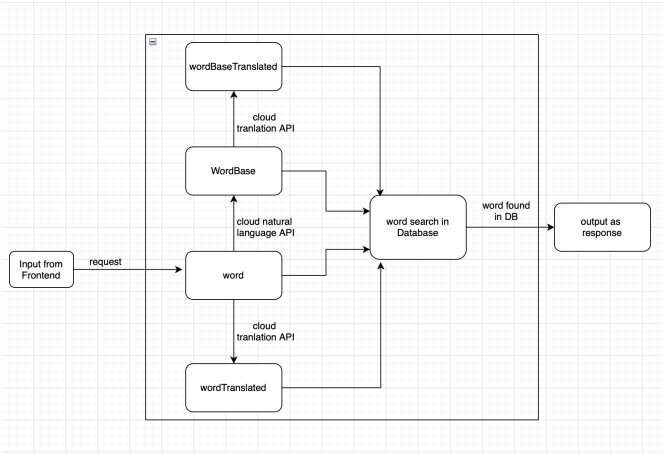

Fig. 6. Workflow diagram of word to infinite form conversion service.

- Saving the user's book reading progress occurs as follows: when the application is launched for the first time, all books are inserted into the database and each user's progress is created for each book. When the front is launched and the user opens the book, the progress of reading the book is sent to the backend.

3) Convert-to-infinitive service : The service was built using NodeJS and deployed to be used as an API.

- The service handles a request with an input (word)
- Due to having both Russian and Kazakh words in our words-videos database, the input is translated to Russian, or kept in Kazakh. The translation and language recognition is done by Google Cloud Translation API.
- It was also necessary to convert words into their base forms (infinitives), in order to find them in the database. The word conversion to their bases is done using Google Cloud Natural Language API
- Lastly, the server searches the converted and translated words in the database, as a response, the the found word proceeds further as shown in the figure of overall architecture of the web application

4) Convert text to pose service:

- **Conversion of interpreter-recorded videos to pose files (separate from website)**
  To begin implementing a service of generating animation of the sign language translation from text, we first needed to convert videos in the Sozder database to pose files.

Therefore, we developed a service to implement this task (the service is separate from the website). It performs:

- extraction of video names to corresponding translations in the Excel file from Sozder database (Excel coordinate of a word in Kazakh or Russian corresponds to its video name) and populating a CSV file that is further used for lookup by the translation service
- converting the videos into corresponding files in Pose format with the use of Pose-Format library for Python. After this procedure, the pose files and populated CSV were uploaded to the Text-to-pose translation service.

- **Text-to-Pose conversion service** As the pose files for all words were ready, we implemented the service that translates text to a pose file. The service was developed as a Flask application with the use of Gloss-Based Pipeline for Spoken to Signed Language Translation (discussed in section 3) and deployed on Google Cloud. The workflow of this service is the following:
  - It accepts a request with the text from Convert-to-infinitive service, in which words are already in the base infinitive form and are in Kazakh / Russian language as they are defined in the Sozder database, such that they can be found in the CSV file.
  - Then, the service uses Gloss-Based Pipeline to perform the following tasks:
    * Lookup in the CSV a mapping of a word to its corresponding .pose file.
    * After the words are found, their poses are appended to a list of .pose files.
    * Once the last word in the string input is processed and its pose file is appended to the list, the stream of pose files is concatenated into a single .pose file that encodes body pose vertice positions for the whole input string.
    * Since the database of the .pose files was obtained from different videos of different formats, sizes (vertical, horizontal, far, close, etc.) the body part positions and scale in each of them are different. Therefore, we used the gloss-based pipeline to normalize, trim, smooth concatenated poses, correct the wrists, and scale poses to produce the final output.
  - If some word is not in the database, the .pose file with meaning "no translation" is returned

The structure of the text-pose conversion service is shown on Figure 7.

5) Convert pose to GIF service: As the tool for interactions with translation API, our team developed a microservice using the FastAPI library of the python programming language. Its workflow is:

- To accept the request from front-end side.
- To send a request containing the accepted body from the front-end to the convert-to-infinitive service.

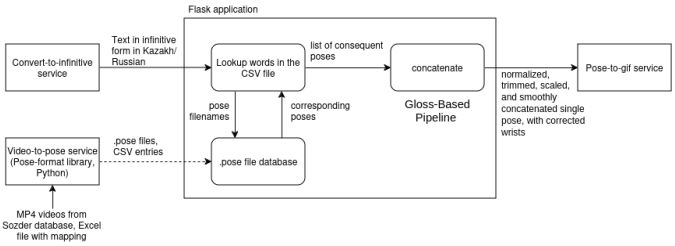

Fig. 7. Convert-text-to-pose service structure

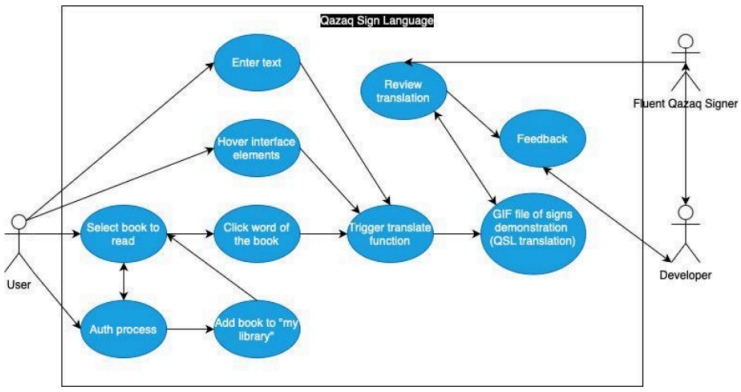

Fig. 8. Use case diagram

- To accept the response and send the further request to the convert-text-to-pose service.
- To accept the response containing .pose file.
- Using pose_format library to visualize the accepted pose file and draw gif file.
- Finally give an answer to the front-end side depending on the status of the response received from translation service api.

The Figure 8 shows the use case diagram for the system.

IV. Project Execution

A. Front-end

1) Issues:

- The Nuxt.js modules' not compatibility.
- Nuxt.js modules' dependencies' installment issue.
- Nuxt.js server's frequent interrupting.
- Sending requests by using AXIOS to the necessary APIs and endpoints.
- Handling logic of functionality and their structure.
- Creating responsive UI.
- Taking dynamic inputs to translate.
- Translating user interface to QSL to make it more usable for deaf people, who are not familiar with letters.

2) How were the issues resolved:

- Transferring the project and re-implementing the front-end using the React library has enabled us to address issues related to module resolution and server interruptions.
- Avoiding Axios and focusing on the use of Fetch.

- Practicing a component-based architecture involves striving to incorporate minimal functionality within each component and breaking down tasks into smaller, more manageable units.
- Incorporating responsive units such as rem and em, along with increased utilization of Tailwind CSS, has significantly enhanced the responsiveness of our web application.
- Utilizing promises, debounce, and timeouts enables us to pause for half a second to one second after input insertion, effectively managing the entered input.
- By implementing the HoverGif component and encapsulating all interface elements, we've enabled the functionality to display animated QSLs of specific elements upon hover.

B. Back-end

1) Issues:

- Due to the reason that the front-end part used Nuxt.js, we had problems with connecting it to the back-end in Django.
- User's book progress worked but it was not correct that it saves only the very first user's progress.

2) How were the issues resolved:

- Prisma is used as the back-end part of Nuxt.js because it was more suitable and had no problems such as Django connection.
- SQLite3 used instead of PostgreSQL database tool because in Django it's already automatically generated.
- We found that progress fields were created in the unique id format of the user and fixed it by removing the unique parameter.

C. Machine learning model construction

1) Issues:

- At the beginning of the entire project, we thought that we would create and train a machine learning model based on the sign.mt project, but it turns out that the creators of this site did not use any machine learning model.
- It turned out to be very difficult for us to create a machine learning model from scratch without instructions, since no member of the group had ever done this.

2) How were the issues resolved:

- We decided to do without a machine learning model and use the created database "sozder", that is, search for words there and, if found, take short videos of these words.
- Then, with the help of several services, we turned these videos of words into gifs and combined them into one big gif, which resulted in smooth gif animations.

D. Convert-to-infinitive service

1) Issues:

- Kazakh language is not supported in the Google Cloud Natural Language API
- No other services available to process Kazakh text
- Deployment problems - tried Heroku, Google Cloud deploy platforms

2) How were the issues resolved:

- Kazakh language processing was implemented manually, by taking the endings out, and leaving the basic forms of the words - stemming
- The service was later deployed on cyclic.sh hosting platform

E. Text-To-Pose-To-Gif converting service

1) Issues and Resolutions: The initial implementation of the service was that a Flask application would output not just a pose file but a GIF. However, it was changed due to the following reason:

- We encountered a problem with deployment that was persisting due to the large size of the service:
  - The database of poses consists of around 1500 files, resulting in a total more than a gigabyte. Many hostings that we tried to use had memory limitations.
  - When requirements are installed, they take several gigabytes of memory as well.
- We managed to deploy the project on Google Cloud since it did not have memory constraints.
- After that, the service seemed to be working, since it was accepting requests from the intermediate API, however, it did not provide the output.
- After inspecting the logs, it turned out that the generation of the GIF file took too long after the resulting pose file was ready, and then the service stopped generating itself. Possibly, due to low resource allocation.
- Therefore, we decided to output not a GIF file but a pose because it would be faster. So we changed the service to output just a pose file, which will be handled later by the intermediate API to generate the GIF with animated translation to sign language.

However, even with these refinements, the response time of the pose generating service was slow (about 10 seconds) on the hosting, while it was instant on a local machine. In addition to the time to generate a GIF file from pose (which also takes around 10-15 seconds for a single word) the total waiting time from sending a text request results in almost half a minute.

We believe that these issues can be resolved once the whole web application is deployed on a server with adequate resource allocation.

V. Evaluation

A. User Testing Organization

To evaluate the effectiveness of our system after the implementation, we conducted user test studies. For the user test we invited five subjects who were deaf adults, two male and three female. The results of this evaluation offer valuable insights into the accessibility and usability of our digital platform for this community.

Fig. 9. User test with a deaf person. From left to right (test subject 1, Kamila Arslanova, Maksat Faizuldanov, interpreter 1, and interpreter 2)

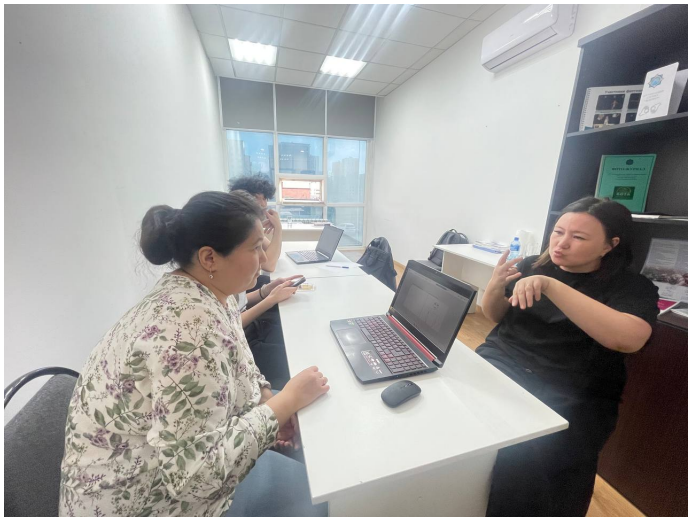

Fig. 10. Tester (on the left) and interpreter (on the right) discussing the implemented system

The testing place was chosen to be "Balam-ai", which is a rehabilitation center for adults and children with special needs. The image below is the process of user testing of subject 1.

B. Tester 1 Results

There were important notes taken from the criticism of the Tester 1 with respect to the Qazaq Sign Dialect (QSL) website. Tester 1 communicated disappointment with low flexibility in the movements of the translating stick figure. They recommended using smoother movements between concatenations of translated signs. Moreover, they highlighted the need to utilize human-like interpreters instead of an animated stick figure, for better understanding. Another proposal was to include a "translation" button. Moreover, Tester 1 pointed out that the accessible dataset of translated signs was incomplete. For example, within the "Books" page of the site, there was no sign for the word "she sewed". Additionally, they advocated for a larger avatar size on the main page as it was hard to understand the given animations by the stick figure.

C. Tester 2 Results

The second subject of the test gave a recommendation to make the preparation of the sign translation to be instant rather than having to wait several minutes. Also, they emphasized that hand movements of a stick figure should be more expressive, or even look like a human. Moreover, they wanted the translator on the main page of the website to appear larger than it was given to them, so that it is clear what is being shown to them.

D. Tester 3 Results

The next tester also shared concerns about the long delays in the preparation of translations of words into signs. The new piece of feedback apart from the comments of previous subjects, is that there might be dialects in gestures that we did not consider. They suggested that to avoid misunderstandings in the translation, there should also be a translation of the desired word in the context of a sentence. Similar to the previous test subjects, he wanted the stick figure to be more human-like.

E. Tester 4 Results

Tester number four provided us with valuable feedback such as the need of providing more detailed explanations for different interpretations of the translated sign words. This subject also suggested improving the smoothness of gestures given by the translator avatar. Furthermore, she would like to have sequential representation of signs of a word within a particular sentence, rather than only one word.

F. Tester 5 Results

The last test subject highlighted the importance of correcting mistakes in the translation of some words throughout the website. Moreover, they recommended using larger texts and animations for better comprehension of the information by the deaf users as it may be ambiguous for some people.

G. Feedback Analysis

Overall, the testers provided valuable feedback on the usability of our implemented system. The most frequent suggestions were to improve the efficiency of the translation process, enhance the smoothness of gestures, change the stick figure to be more human, and refine the visual elements of the website. Additionally, there was a consistent preference for clearer translations with additional explanations apart from pure translations. Although the suggestions to change the functionalities of the website were prevalent, there were also appreciative comments on some design and usability features. These include color scheme, icons, appearance of the translating stick figure on word hover, and structure of the site.

VI. Conclusion and future work

In conclusion, our team has developed the planned web application or service to address the issue of communication barriers within society. Throughout the implementation process, we prioritized creating a user-friendly UI and integrating our own Qazaq sign language real-time translation service.

However, during evaluation our users met the challenges and expressed their feelings, providing us with valuable feedback that were discussed above. To resolve these challenges, the development team in the future should:

- Note the specific translation methods of interpreters that provide us with a data set.
- Try to replace the stick figure with a human-like avatar, since the emotions and feeling expressions have significance in the translation.
- Optimize the translation service and try to speed-up the time of getting the result.
- Provide a definition of an unknown word, rather than just a translation.
- Make adjustments to the UX/UI design to represent states of waiting, success or unsuccess in translation.
- Implement a feedback feature such that users could evaluate the translation and suggest their own versions.

Moreover, there is a possibility of improving the project by combining the web application with other teams' works. For example, as our advisors suggested, the web service could be combined with a sign language recognition project of another team. Such a decision would lead to increase in the usability of the product.

References

[1] European Language Equality (ELE). D1.40: Report on Europe's Sign Languages, 2023. Accessed: September 6, 2023.

[2] Forbes Kazakhstan. Kazakhstanskiy Uchenyi Razrabotala Sistemu Raspoznavaniya Kazakhskogo Jestovogo Yazyika, July 2023. Accessed: July 31, 2023.

[3] C. McKeown and J. McKeown. Accessibility in online courses: Understanding the deaf learner. *TechTrends*, 63(5):506–513, 2019.

[4] A. Moryossef. sign.mt: Effortless Real-Time Sign Language Translation, 2023.

[5] A. Moryossef and Y. Goldberg. Sign Language Processing, 2021.

[6] A. Moryossef, M. Müller, and R. Fahrni. pose-format: Library for viewing, augmenting, and handling .pose files. GitHub, 2021.

[7] A. Moryossef, M. Müller, A. Göhring, Z. Jiang, Y. Goldberg, and S. Ebling. An open-source gloss-based baseline for spoken to signed language translation. In *2nd International Workshop on Automatic Translation for Signed and Spoken Languages (AT4SSL)*, June 2023. Available at: https://arxiv.org/abs/2305.17714.

[8] M. Mukushev. Sozder [video file]. Retrieved from https://drive.google.com/drive/folders/1s97j4AGIrVl65uNRc9uJiZybK9tMM5GG, 2023.

[9] M. Mukushev, A. Kydyrbekova, V. Kimmelman, and A. Sandygulova. KRSL-OnlineSchool: Large Vocabulary Kazakh-Russian Sign Language Dataset. Retrieved from https://krslproject.github.io/online-school/, 2022.

[10] M. Mukushev, A. Sabyrov, A. Imashev, K. Koishybay, V. Kimmelman, and A. Sandygulova. Evaluation of manual and non-manual components for sign language recognition. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 6075–6080, Marseille, France, 2020. European Language Resources Association.

[11] World Health Organization. Deafness and hearing loss, February 2023.

[12] Z. Yu, S. Huang, Y. Cheng, and T. Birdal. SignAvatars: A Large-scale 3D Sign Language Holistic Motion Dataset and Benchmark. *arXiv preprint arXiv:2310.20436*, 2023.