IMMERSIVE JAPANESE LANGUAGE LEARNING WEB APPLICATION USING SPACED REPETITION, ACTIVE RECALL, AND AN ARTIFICIAL INTELLIGENT CONVERSATIONAL CHAT AGENT BOTH IN VOICE AND IN TEXT

Approved by:

German Harvey Alférez

Professor Germán H. Alférez, Ph.D., Adviser

Kenneth E Caviness

Professor Kenneth Caviness, Ph.D.

Roberto Ordoñez

Professor Robert Ordóñez, MSc.

Date Approved 4/12/2024

IMMERSIVE JAPANESE LANGUAGE LEARNING WEB APPLICATION USING SPACED REPETITION, ACTIVE RECALL, AND AN ARTIFICIAL INTELLIGENT CONVERSATIONAL CHAT AGENT BOTH IN VOICE AND IN TEXT

by

Marc Butler

A PROJECT DEFENSE

Presented to the Faculty of

The School of Computing at the Southern Adventist University

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Germán H. Alférez, Ph.D.

Collegedale, Tennessee

April, 2024

IMMERSIVE JAPANESE LANGUAGE LEARNING WEB APPLICATION USING SPACED REPETITION, ACTIVE RECALL, AND AN ARTIFICIAL INTELLIGENT CONVERSATIONAL CHAT AGENT BOTH IN VOICE AND IN TEXT

Marc Butler, M.S.

Southern Adventist University, 2024

Adviser: Germán H. Alférez, Ph.D.

In the last two decades various human language learning applications, spaced repetition software, online dictionaries, and artificial intelligent chat agents have been developed. However, there is no solution to cohesively combine these technologies into a comprehensive language learning application including skills such as speaking, typing, listening, and reading. Our contribution is to provide an immersive language learning web application to the end user which combines spaced repetition, a study technique used to review information at systematic intervals, and active recall, the process of purposely retrieving information from memory during a review session, with an artificial intelligent conversational chat agent both in voice and in text. The proposed solution created with Japanese as the target language and English as the understood or source language. One benefit of this approach is that two end users do not have to be present in order to encounter new words through a natural conversational context. Another benefit is that a given end-user can use the web application without the fear of offending another person who speaks the language to be learned. What makes the proposed solution, called Immersio, different than other existing applications is the combination of the space repetition system which is tied to the chat page. Immersio was evaluated by a native Japanese speaker. It received an average score of 3.24 out of 5.0 points

out of seventeen questions. In terms of five open-ended questions, the reviewer likes the following: the chat page worked well and the grammar was mostly correct and natural. However, he dislikes the following: Part-of-Speech and the dictionary, which were not readily accessible. Also, he was unable to upload audio. Moreover, he indicated that the review page was confusing, and he could not hear most of the audio voice from the AI chat.

Contents

List of Figures

Listings

List of Tables

Chapter 1

Introduction

This chapter discusses the outline for this project. First, it states the problem statement and the motivation. Additionally, this chapter covers high-level implementation details of the proposed solution. The chapter concludes with project objectives, limitations of existing solutions, delimitations, and an overview of the project.

1.1 Problem Statement and Motivation

In the last two decades there has been a surge of several language learning applications, both including web and mobile applications, which either employ spaced repetition or artificial intelligence, usually through a conversational chat agent. For example, SuperMemo[1], a reputable language learning web application, employs a flashcard system as well as an OpenAI[2] GPT model. Another example is Univerbal[3], a web and mobile application which employs a conversational

[1] https://www.supermemo.com/
[2] https://chat.openai.com/
[3] https://www.univerbal.app/

agent. Both of these applications utilize their conversational agent to provide dynamic interactions for a user including text messaging and speaking. However, throughout the various applications that exist there does not currently exist a web application that integrates both artificial intelligence, chat agents, an intuitive user interface, and algorithms which space language learning into manageable units of time.

Also, current applications do not utilize Part-of-Speech (POS) parsing of artificial intelligent text generated on-demand. POS can be thought of as pieces of sentences which are broken up into verbs, grammar points, words, nouns, and much more [2]. This is problematic because understanding how different POS interact with each other is vital to the language learning process. Furthermore, most POS items in existing solutions are hard-coded into a database which has a negative side-effect of not accurately portraying the other surrounding POS items.

In particular, current dictionary tools, such as the JMDict project[4] that consists of the work of Jim Breen[5] and EDRDG[6] or Tatoeba[7], another large language-to-language database or repository, only allows for a language learning application to provide POS information with predefined sentences from a database. Using tools such as these has benefits in the sense that predefined sentences may be grammatical and linguistically correct if the data was entered by a professional in the target language. However, these approaches have the pitfall of containing errors if the data was entered by a user with little knowledge on the target language.

Yet another issue of existing solutions is that audio generated via an artificial intelligent conversational agent that employs spaced repetition is not widely

[4] https://www.edrdg.org/jmwsgi/srchform.py?svc=jmdict&sid=
[5] http://nihongo.monash.edu/cgi-bin/wwwjdic?9T
[6] https://www.edrdg.org/
[7] https://tatoeba.org/en/

utilized. This is problematic because it can be expensive to hire voice actors who specialize in a particular language to read a script for a language lesson. Furthermore, scripts must be created by someone and validated such that the audio recorded by a given voice actor is linguistically correct. One of the existing applications that inherits this pitfall is known as Pimsleur[8], designed after Paul Pimsleur's method, which employs spaced repetition via audio lessons.

Additionally, it is rather difficult to find a real person who is willing to consistently practice with another during a language exchange. A language exchange is where two or more language learners engage in conversing with one another in their target languages and explaining intricate details of their own language to others. The caveats that typically come with traditional language exchanges is that the two or more language learners have to either converse asynchronously with a form of messaging or during a practical time in their schedules in which a voice call can take place. Unfortunately, it is difficult for two or more real people to find time to exchange languages. In addition, there may be cultural barriers which may cause difficulty by discussing a certain topic. For instance, it may be socially appropriate to speak about one topic in one culture but rather impolite to speak about it in another.

1.2 Proposed Solution

Our contribution is an immersive language learning platform, which is known as Immersio. Immersio was designed with Japanese as the target language for this project. The goal of Immersio is to combine the immersive interaction of speaking with a conversational agent that implements a spaced repetition system. Open

[8] https://www.pimsleur.com/c/how-to-learn-a-foreign-language

AI's GPT-3.5-Turbo model, via REST API, was used as the conversational agent. Moreover, the aim of Immersio is to provide a natural and believable language learning process which should cultivate skills such as speaking, typing, listening, and reading Japanese.

Immersio was constructed with various tools to facilitate learning, modularity, and cohesiveness. The project utilizes a micro-service architecture by using tools such as Docker[9] for containerization and RabbitMQ[10] for communication between Docker containers. With this architecture it is possible to place artificial intelligent models in Docker containers, which can easily be replaced in the case that a better model or models are found. Immersio contains audio transcription and audio generation with Whisper[11] and CoquiTTS[12] , respectively. Whisper and CoquiTTS have base models that are capable of understanding Japanese and so they are used in this project. For POS parsing, a spaCy[13] model, through a Python script, was placed in a Docker container. Because spaCy's base model contains the ability to understand Japanese, it was unnecessary to retrain spaCy's models. For translation and grammar correction of Japanese, HuggingFace transformer models[14] were placed in containers as well. It is important to note that the Hugging Face transformer models that were used are pre-trained and are available as a Python PIP package. Therefore, Immersio did not require additional training to these models. Specifically, the Hugging Face translation model that was used is `opus-mt-ja-en`[15], which is a translation model that translates Japanese text and transforms it into English. The model was developed by the Language Technology

[9] https://www.docker.com/
[10] https://www.rabbitmq.com/
[11] https://github.com/openai/whisper
[12] https://github.com/coqui-ai/TTS
[13] https://spacy.io/
[14] https://huggingface.co/
[15] https://huggingface.co/Helsinki-NLP/opus-mt-ja-en

Group at the University of Helsinki[16]. Because it is possible for a Japanese-to-English translation model to output ungrammatical English, we therefore pipelined the output of the translation model into an English grammar model. The English grammar model takes ungrammatical English as input and returns grammatical English as an output. Ungrammatical English denotes English text which does not follow grammar rules in English. For instance, if a piece of text is written as, "My name is Sarah and lives in London," it is considered ungrammatical English text. To make this text grammatical we passed it in a grammar model in which the output returned is, "My name is Sarah and I live in London." The Hugging Face model that was used to achieve this is `coedit-large`[17] , maintained by Grammarly[18]. In order to cohesively bind the services together, Django[19] was employed to communicate with the micro-services using RabbitMQ with remote procedure calls (RPCs) and the frontend with REST API which was designed using React[20]. Because the conversational agent and the spaced repetition system are crucial in the language learning application, we placed OpenAI's[21] GPT-3.5-Turbo API in the same container as the Django backend along with LangChain[22] and PostgreSQL[23] to remember conversations.

There are also various instruments employed by these tools to make development more secure, fast, scalable, and reliable. For instance, Django uses Django

[16] https://huggingface.co/Helsinki-NLP
[17] https://huggingface.co/grammarly/coedit-large
[18] https://huggingface.co/grammarly
[19] https://www.djangoproject.com/
[20] https://react.dev/
[21] https://openai.com/
[22] https://www.langchain.com/
[23] https://www.postgresql.org/

Rest Framework (DRF)[24] where React uses TailwindCSS[25], Vite[26], Vitest[27], Mock Service Worker (MSV)[28]. We also used Figma[29] for constructing the user interface on the frontend. Moreover, we also employed continuous integration with Github Actions[30]. When a user chooses to save recorded audio files, these files are saved locally with IndexedDB[31] in a user's browser rather than expend server resources in the backend. Once all of the tools were ready to be used, Docker Compose[32] was used to start all the services at once. Finally, we deployed the application on Railway[33].

Immersio includes two main sections consisting of the chat page and the reviews page. The goal of the chat page is to emulate a dynamic and somewhat realistic conversation between a human and another human. The difference is that a given registered end user can be set up to chat with an artificial intelligent conversational agent which is capable of speech replies, speech comprehension, text replies, text comprehension, memorization of previous messages in the conversation, parsing POS, and dictionary lookup.

Whenever a user desires to save a message from a conversation they have had with their artificial intelligent language exchange partner, they can easily press a button to send the message, or some portion of it, to review. Reviews are Immersio's implementation of a spaced repetition system. The reviews are designed for a user to study content such as sentences, individual words, and

[24] https://www.django-rest-framework.org/
[25] https://tailwindcss.com/
[26] https://vitejs.dev/
[27] https://vitest.dev/
[28] https://mswjs.io/
[29] https://www.figma.com/
[30] https://github.com/features/actions
[31] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB
[32] https://docs.docker.com/compose/
[33] https://railway.app/

even POS. The reviews are completed through the review page which contains lists of various review items that a user has added to their review page. Upon selecting one of these lists on the review page the user can then select whether or not individual items show up during their review session with check-boxes in a table. A user also has the chance to add their items to a custom review list, reset what is known as a Spaced Repetition System (SRS) level, which is simply a metric to determine how often a review item is reviewed, or simply delete the item from their reviews. Moreover, a user is able to delete review items but will have to confirm deletion via a confirmation modal[34]. A modal is a frontend development term for a UI component that displays over the current page but does not fully cover the contents underneath it. A modal is useful because it assures a user that they have not left the page they were on. In Immersio's case, the modal verifies if the user actually wants to delete a review item by asking them to press a confirm button. Finally, a user is able to review these items with due care by pressing the start review button, which generates a review queue that utilizes a priority queue in JavaScript and React hooks.

Once a review session is started a user goes through various stages that can be represented in a finite state automata, as shown in Figure 1.1, which consists of translation of the review item, the context of a review item along with whether or not a user has answered correctly, and a review report. During this process, the user is graded against artificial intelligent models, and finally the review report determines how well a user has done during asking to translate the review items. The benefit of this finite state automata is that it can be expanded to accommodate a mode that only allows speaking in the user's target language or it could be

[34]A modal is a frontend and UI term that resembles a popup that prompts a user to take an action.

adapted to include that the user must select from a list of correct options that would also be generated by an artificial intelligent model in the backend.

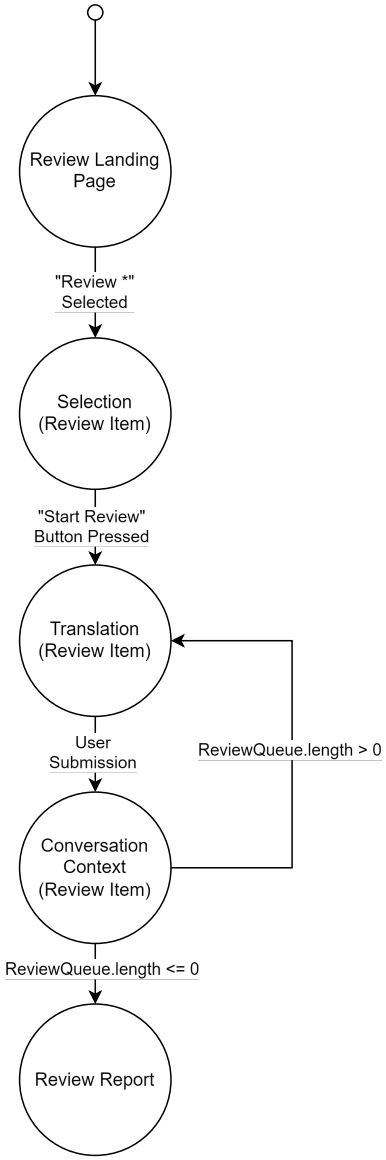

Figure 1.1: Finite state automata of the review process a user would take

1.3 Objectives

The main objective of Immersio is to deliver an immersive Japanese language learning web application using spaced-repetition, active recall, and an artificial intelligent conversational chat agent both in voice and text. For this reason Immersio is broken up into sub-objectives which are to:

1. Create the review section which includes:
    - A spaced-repetition system (SRS) and active-recall system which tracks a user's progress
    - A review landing page that includes:
        - The ability for a user to review all of their review items. A review item is a word, POS, or sentence that is prompted to translate by a user during a review session. A review session is simply an allotted portion of time in which a user reviews a subset of all of their review items
        - The ability for a user to review all of their sentence review items
        - The ability for a user to review all of their word review items
    - A review queue selection page which allows:
        - A user to add review items to a review session's review queue. A review queue is a data structure that manages the order of review items and determines the next review item that is displayed during a review session
        - A user to see how many review items are available
        - A user to add the review items to a custom review item list

  - A user to reset the SRS level of a review item. A SRS level is a value that determines the review frequency of a review item. A SRS level ideally measures how much a user memorized a review item
  - A user to delete a review item
  - A user to delete their entire review queue item list
- A prompt for the user to translate a review item that:
  - Checks the user's response against an artificial intelligent capable of comparing a user's translation to the the actual translation
  - Provides the ability to both type and speak the answer
- A conversation context panel to:
  - Show up after a user has attempted to translate the prompt
  - Provide whether or not the answer is incorrect
  - Provide the differences that were made in the answer they made compared to the artificial intelligent agent's translation it graded against
  - Provide the option for the user to assert that they were correct or incorrect in their translation
  - Provide the option for a user to undo their translated answer with an undo button
  - Have an option to toggle showing the corrections made by the artificial intelligent agent
  - Provide a continue to next prompt button

2. Create the chat page which allows a user to converse with an artificial intelligent agent, which has various features including:

- Asserting that a user must speak in their target language
- Sending a message in the user's target language that generates a response in the user's target language
- Generating audio via CoquiTTS's Kokoro model that is trained on the Kokoro dataset[35] and storing the audio locally in IndexedDB to save sever resources
- Ability to send audio messages to the backend that Whisper converts into text and the functionality for IndexedDB to save the audio locally to preserve server resources
- The ability to translate the messages on the interface
- POS high-lighting which includes:
  - Dictionary lookup of words or POS
  - Color coding to make it easier to understand what the purpose of the item is for
  - The ability to add the item to the reviews as a word

3. A dashboard page which shows a high-level overview of the information of a user

1.4 Limitations

Due to time, linguistic skills, hardware, and monetary constraints, this project has the following limitations:

[35] https://www.kaggle.com/datasets/kaiida/kokoro-speech-dataset-v11-small

- The quality of the the audio generated by CoquiTTS, a deep learning toolkit for Text-to-Speech (TTS) synthesis[36], is limited by the default model which is trained on the Kokoro Speech Dataset[37]. Additionally, the CoquiTTS toolkit may have limitations in itself or its dependencies such as Cutlet[38]. Cutlet is a tool to convert Japanese text into romanized text called Romaji. CoquiTTS uses romanized text produced from Cutlet in order for its Kokoro model to synthesize Japanese words into audio. Because the romanized counterparts of Japanese may not fully represent the native Japanese pronunciation, the audio generated by CoquiTTS may not speak with a native Japanese accent.

- There was only one native Japanese speaker who evaluated Immersio. Therefore, he evaluated all the Japanese grammar, translation, pronunciation, generation, and correction.

- Because we did not have anyone on our immediate team to validate the accuracy of a grammar or translation model in Japanese we used community translation models and grammar models from Hugging Face and did not of train them further.

1.5 Delimitations

This project is delimited in the following ways:

- We did not train a conversational language model as many options already exist to execute this functionality with high precision such as OpenAI's GPT

[36] https://github.com/coqui-ai/TTS
[37] https://www.kaggle.com/datasets/kaiida/kokoro-speech-dataset-v11-small
[38] https://github.com/polm/cutlet

models[39] and LangChain[40].

- We did not train a Speech-to-Text (STT) model as Whisper[41] was sufficient for our needs. Speech-to-Text (STT) is the process of converting human speech into human readable text. Whisper leads Speech-to-Text technology on the market.
- Because spaCy has out-of-the-box support for POS parsing of various languages – including Japanese – we did not train spaCy's predefined models.
- The Hugging Face models, `opus-mt-ja-en` and `coedit-large`, was not modified or trained in any way as these models were sufficient for the project.

1.6 Overview

This document is organized as follows:

- Chapter 2 provides the theoretical framework and state of the art
- Chapter 3 outlines the methodology for the construction of Immersio
- Chapter 4 delineates the procedures that were involved in the evaluation of Immersio
- Chapter 5 presents the finalized application and the evaluation results
- Chapter 6 summarizes the project outcomes and the future work

[39] https://platform.openai.com/docs/models
[40] https://python.langchain.com/docs/get_started/introduction
[41] https://github.com/openai/whisper

Chapter 2

Background

This chapter presents the theoretical framework and state of the art. First, the theoretical framework section delves into concepts and paradigms used in the creation of a language learning application. Then the state of the art examines the tools that have applied the concepts shown in the theoretical framework.

2.1 Theoretical Framework

This section explores the techniques used in language learning applications as well as inspects the brief history in contriving such techniques. It covers a range of topics such as algorithms, machine learning, linguistic procedures, and data used in language learning tools. Figure 2.1 summarizes these topics into what makes a language learning application.

2.1.1 Pedagogy

Pedagogy is the method and practice of teaching a particular subject. In Immersio's case, pedagogy refers to methods and techniques that involve efficient memoriza-

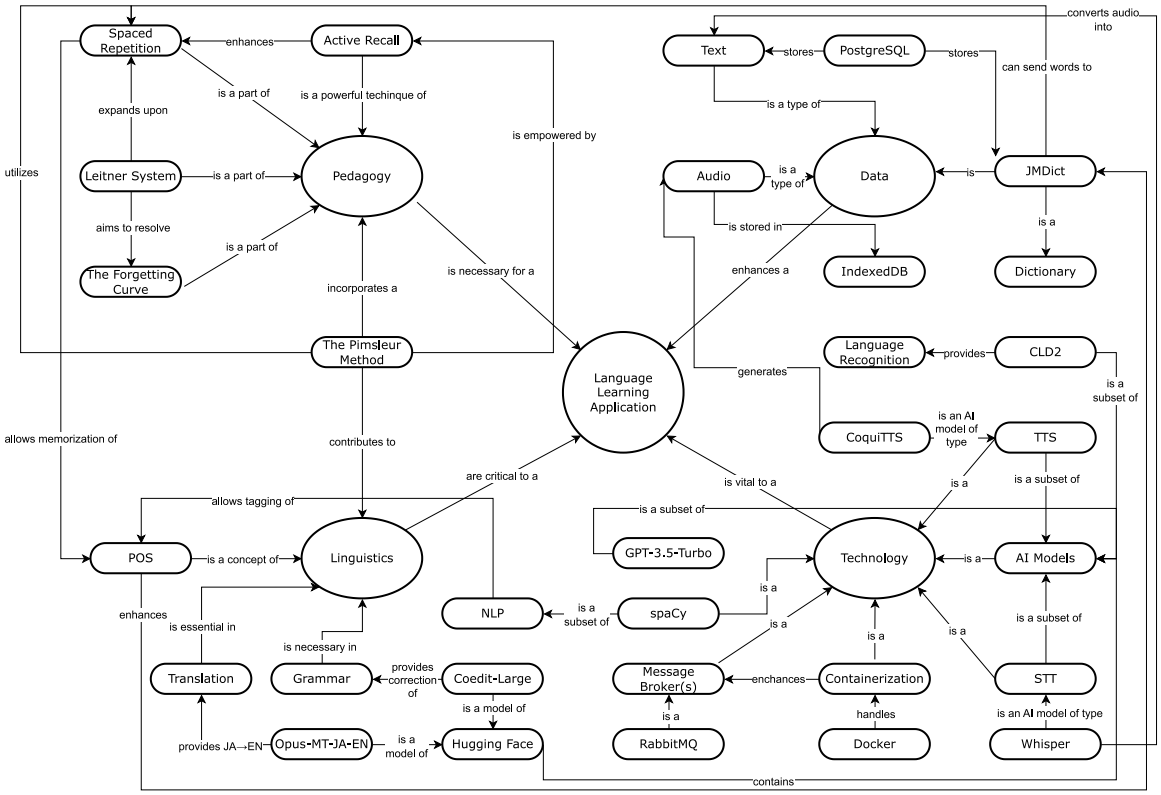

Figure 2.1: Concept map of topics explored in the theoretical framework

tion and language learning techniques. One concept that is pertinent to language learning applications is the concept known as the *forgetting curve*. The forgetting curve is an observation of human retention of information plotted over time as seen in Figure 2.2. The curve at the beginning of learning a specific piece of information is where time can be expressed as $t = 0$. At $t = 0$, the user has retained all of the information he or she already has. Thought of another way, the user has not lost or forgotten any information that they had just learned. However, as time goes on the amount of information retained is reduced logarithmically – in which a steep drop of loss of information is expected to occur before the curve smooths out into a gradual decline. This means that the slope of $|t = n| > |t = n + 1|$ can be formulated into a logarithmic function such that $b = \frac{100k}{(\log t)^c + k}$. Where k and

c are constants and can be represented as $k = 1.84$, $c = 1.25$.

The forgetting curve and its logarithmic function is credited to German psychologist Hermann Ebbinghaus who had observed the phenomenon in 1880 [3]. Despite the time period of creation, Ebbinghaus' work has continued to reinforce the spaced repetition methodology as well as conceived replicated studies which exhibit similar results [3].

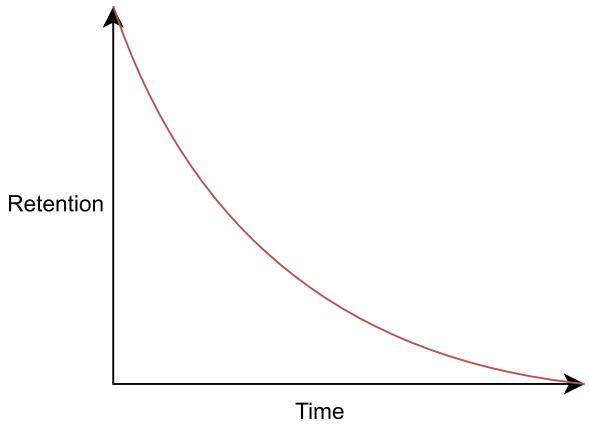

Figure 2.2: Ebbinghaus' Forgetting Curve [1]

Without the work of Ebbinghaus, works in spaced repetition may not have been possible. Spaced repetition is defined as a method of reviewing information at systematic intervals [4]. C.A. Mace was a British psychologist who expanded upon the works of Ebbinghaus. Specifically, he proposed whenever learning new information, "there is grinding work to do [... through] concentrated effort" and "[after] every mnemonic device has been exploited there remains much that can be acquired only by drudgery and systematic repetition" [5]. Mace provides a method for "learning a foreign language" by "parceling the material to be memorized into units which can be tabulated on postcards" and then memorized and "spaced in gradually increasing intervals, roughly intervals of one day, two days, four days, eight days, and so on" [5].

When overlaying a spaced repetition system over Ebbinghaus' forgetting curve a given user is shown to forget less information over time compared to without using a spaced repetition system as seen in Figure 2.3. For example, if a user decides to review information after some time, Ebbinghaus' forgetting curve tends to flatten or the logarithmic curve becomes less steep. Furthermore, if a user reviews a second time, the curve continues to become less steep as well as when the user does sequential reviews of the given information.

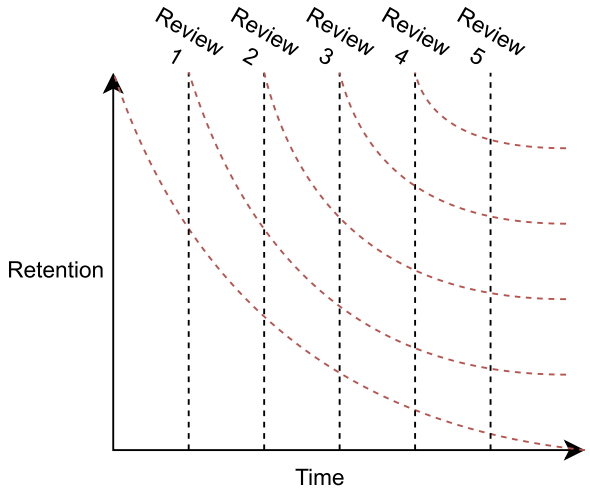

Figure 2.3: Ebbinghaus' Forgetting Curve with Spaced Repetition [1]

C.A. Mace explains that reviewing information can take place in two forms. The first of which is known as *passive repetition* which can constitute a user re-reading a list of items when reviewing. The second is known as *active repetition* or *active recall* in which a user may recall an "alternate" but synonymous meaning of the information being reviewed [5]. This information is vital to a language learning application. However, many language learning applications only use flashcards which do not enforce active recall effectively. This is especially an issue in applications that do not harness the power of artificial intelligence, which possesses the ability to detect whether a prompt is at least roughly equivalent

to the answer stored – typically stored in the backend in a database. Instead many solutions may fall into the fallacy of comparing a user's answer in a spaced repetition software directly to the stored answer in a database. If this occurs, the user's answer, despite being correct may be flagged as incorrect by the software. For instance, if a user types in, "Hello Earth" but the database has stored the correct answer as "Hello World" the user may be marked incorrect even if "World" and "Earth" can be used in the same context synonymously and still portray the same meaning. This pitfall can be mitigated by employing artificial intelligence which can interpret the meaning of words in context to check if they are synonymous and deem a user's answer to be correct if they are synonymous and incorrect if it is not.

A concept that extends from spaced repetition is known as the Leitner System which was developed by Sebastian Leitner [6]. Leitner proposed an implementation for a spaced repetition system by including three elements: flashcards, boxes, and review intervals. The boxes in his proposed implementation were mapped to a particular review interval, or the time a flashcard was supposed to stay in the box until the next review. Flashcards that have never been reviewed before would be placed in the first box, which would have the shortest review interval. At this point in time, the review interval lasts until a particular point in time in which the flashcards would be taken out to be reviewed. The flashcards may be in the form of questions with their associated answers. In such case, a review prompts a user to associate the answer of the flashcard to the question of a particular flashcard. This determines if the flashcard advances to a new box with a longer review interval, remains in its current box with the same review interval, or if it is placed in a previous box with a shorter review interval. Figure 2.4 provides an illustrated example of an implementation using the Leitner System. This figure shows that

new flashcards enter the box with the lowest review interval, designated in this figure as "1". After some time these flashcards are reviewed. Specifically, they may either be moved to the next box ("2" in this case) with a slightly higher review interval or remain in the first box to await another review. If the user associates the correct answer of a flashcard to the flashcard's question, the card will either be moved to a box with a slightly higher review interval, remain in the same box, or be removed from the system entirely – depending on the algorithm and whether or not the flashcard's current box has the highest review interval. If the user does not associate the answer of a flashcard to the flashcard's question, the flashcard will either be moved to the first box with the lowest review interval so that it is encountered quickly as per Leitner's implementation, moved back to a box with a slightly lower review interval as implementations seen in traditional language learning applications, or remain in its current box – depending on the algorithm and whether or not flashcard's current box has the lowest review interval.

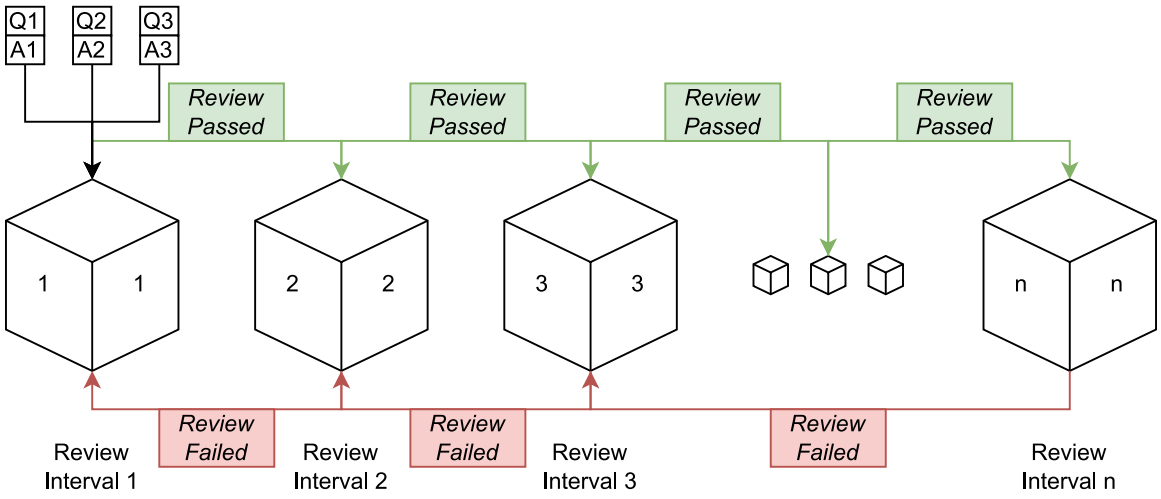

Figure 2.4: An example of the Leitner System [1]

The Leitner System has given rise to multiple algorithms designed to work in conjunction with the system as well as sparked research studies in fields of

psychology and medical sciences. One such study aimed to enhance human learning via spaced repetition by utilizing dynamic parameters rather than hard-coded constants in rule-based heuristic algorithms [7]. It is in research such as this that the Leitner system is valued for its exponential spacing which may be explicitly cast using arbitrary formulae. In fact, the Leitner System not only implements spaced repetition but it also accounts for Ebbinghaus' forgetting curve by mapping flashcards or review items to a particular review interval.

The Leitner System has also influenced the creation of multiple commercial flashcard and language learning software, which have implemented their own spaced repetition algorithms. For instance, SuperMemo has utilized spaced repetition algorithms such as SM-0, SM-2, SM-8 and SM-18. The creation of these algorithms is credited to Polish computer scientist Piotr A. Wozniak as proposed in his master's thesis [8]. Specifically, he presented the "problem of forgetting" and proposed his SuperMemo solution which consists of two restraints:

1. Intervals should be as long as possible to obtain the minimum frequency of repetitions

2. Intervals must be short enough that knowledge is still remembered

The SM-2 algorithm was developed, was modified slightly, and utilized by the Anki software [9]. As research in the field of spaced repetition and the Leitner System had progressed, open-source algorithms such as the Free Spaced Repetition Scheduler (FSRS) algorithm were made publicly available [10], [11]. The FSRS algorithm can be installed as an add-on with the Anki[1] software.

One method that integrates spaced repetition and the forgetting curve into a language learning technique is known as the Pimsleur method. The Pimsleur

[1] https://apps.ankiweb.net/

method was created by Paul Pimsleur, Ph.D. His work was instrumental for theories of language learning as well as the creation of the language learning application. Pimsleur is known for publishing books and papers based on learning a foreign language [12] and psychological experiments on second language learning [13]. Also, Pimsleur had taken interest in spaced repetition and Ebbinghaus' forgetting curve which he applied the forgetting curve to short-term memory [14]. He illustrates this with a language student who is taught the meaning of a modern Greek word and its pronunciation. The student is then instructed to recall the information a second later. At this point in time, Pimsleur claims that the probability that the student will remember the word is "approximately 100%." As time increases, in a matter of seconds, the probability of recalling the correct response decreases drastically. For instance, if the student waits six seconds to recall the meaning of the word that was taught, the student is more likely to forget the meaning of the word at this point in time compared to if the student had only waited one second to recall the word. Pimsleur expresses this concept in detail by supposing that at six seconds the probability of recalling the correct response is 35% – which is not likely to be recalled when prompted. Pimsleur claims that when a student is prompted to recall the word at a point in time with a decent probability of 60%, the teacher will remind the student of the correct answer. By reminding the student of the correct answer, the probability of recalling the word rises back to 100%. This point Pimsleur makes also relates to Pior Wozinak's SuperMemo solution in which, "Intervals should be as long as possible to obtain the minimum frequency of repetitions" [8]. By not relying on perfect probability of remembering an item an efficient spaced repetition algorithm can be formulated.

Additionally, Pimsleur explored his philosophy of recognizing a good teacher. He accomplished this by comparing two teachers who teach Japanese and Swahili

[12]. The Japanese teacher, on the first day, had only described the sounds of Japanese, the theory, and the possible syllables. However, on the first day for the Swahili teacher, the teacher prompted the students to stand and sit in Swahili. The Swahili teacher emphasized practice over theory and understood that repetition is key to learning a language. Pimsleur declared that "[g]rammar is best learned by using, not by talking about it." Pimsleur therefore encouraged language learning to be interactive, provide the students to think about responses to questions, and to capitalize on repeated instructions by limiting usage to the target language. This tenet is a driving force in this project.

2.1.2 Linguistics

Linguistics is the study of human languages and its structure, including morphology, syntax, phonetics, and semantics. Linguistics is important to Immersio because it determines how artificial intelligent models and other technologies can produce and understand Japanese. In the context of linguistics, Part-of-Speech or POS refers to a word, grammar point, or piece of text that has a particular function in a sentence. Specifically, POS may be a noun, pronoun, verb, adjective, adverb, conjunction, interjection, preposition or determiner. Because POS determines a function in a sentence or phrase, it becomes useful for both AI tools such as natural language processing or machine learning and humans to be able to understand its purpose in a sentence. POS is valuable for language learning tools and applications as shown in Indonesian POS tagging systems [15]. POS is also used in some dictionaries such as JMDict[2]. It is therefore vital to utilize tools and standards that take advantage of POS parsing. With such tools it is possible to create on-demand POS parsing on dynamically generated text from a conversational agent. This method

[2]https://www.edrdg.org/jmdict/j_jmdict.html

is expected to become superior than traditional usage of POS where POS data is stored statically in a dictionary, such as the Contemporary Chinese Dictionary (Xian Dai Han Yu Ci Dian) [16] [17] or a database such as JMDict [18]. Even when stored statically in a database, it is typical for language learning applications to not display POS in a concise or efficient manner. For instance, many applications may only allow an end-user to look up words in a dictionary and provide whether a word is a noun, verb, or a few other common POS types. However, POS types can be broken up into subtypes. For instance, a noun can be broken up into a person, place, thing, animal, quality, ideology, or an action. Additionally, a verb can be split into an auxiliary or linking verb. This phenomenon is known as POS subtypes. However, traditional or current language learning applications do not appear to take advantage of on-demand POS generation or POS subtypes.

Like POS, translation has been considered heavily in the field of linguistics [19]. Translation can vary depending on the source and target language. Additionally, translation can resolve to different meanings depending on the context. Parallel to translation, grammar is regarded as a vital component in linguistics.

2.1.3 Data

In terms of data, one project has taken on the form of a refined dictionary known as JMDict[3]. JMDict[3] is an online electronic dictionary and project overseen by Jim Breen [20] and managed by the Electronic Dictionary Research and Development Group [21]. The main objective of JMDict[3] project was to provide a multilingual dictionary in the form of a lexical database [22]. JMdictDB[3] contains various tables which can be used to query words regarding their definitions, the sense or tense of the word – such as if a word is used in passive or causative style, the POS if

[3]https://www.edrdg.org/jmdict/j$_j$*mdict.html*

applicable, the word itself, and the Kana reading (Chinese characters, or Kanji, can be mapped to a particular script such as Hiragana and Katakana where Hiragana and Katakana make up what is known as Kana).

An exceptional language learning application would not be complete without textual and audio data of both source and target languages[4]. On one hand, textual data from both source and target languages are used to teach typing and reading skills. On the other hand, audio data is used to teach speaking and listening skills in a language learning application. For these reasons, it is important to store both textual and audio data in a database.

There are two types of databases which are used in Immersio: PostgreSQL[5] and IndexedDB[6]. PostgreSQL[5] was used to store textual data for a user and IndexedDB[6] was used to store large audio files recorded by a user. PostgreSQL[5] is a reputable, open-source, SQL relational database. It is significant because it can be used with Django's[7] object relational mapper (ORM). An ORM is a tool that abstracts user interactions of a database by representing database relations as an object. This is typically done through a `class` in a programming language such as Python. Using this configuration allows easy interactions with PostgreSQL[5]. However, if a user needs to store large files, such as audio data, PostgreSQL[5] can quickly grow in size and cause multiple problems. One problem is that accessing the database can become slow. Another problem is that it can be expensive to contain large files in a database. For these reasons, a solution that can be used in conjunction with PostgreSQL needs to be used. This solution needs the ability to store large files, also known as blob data. There are multiple solutions for this problem but

[4]A source language refers to a language that a language learner is learning from while a target language refers to a language that a language learner is trying to learn.

[5]https://www.postgresql.org/

[6]https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB$_{A}PI/Using_{I}ndexedDB$

[7]https://www.djangoproject.com/

the one that was considered is IndexedDB[6]. IndexedDB[6] is a database that stores data in a client's browser. By using IndexedDB[6] to store audio data in a client's browser, privacy issues are eliminated as the audio data is not maintained on a machine that the user does not own. Because the audio is stored in a client's browser, a user can delete their own recordings without having to make a REST API request to the backend server which would otherwise incur network expenses when attempting to access a large file. However, one potential issue that may occur when using IndexedDB is that a user may accidentally delete their data. This can occur if a user clears their browser cache which also removes IndexedDB databases. Because audio data can be regenerated on Immersio, via CoquiTTS[8], it is not critical to ensure that audio data is preserved. However, this means that important textual data that cannot easily be regenerated needs to be stored with some perpetual solution. For this reason, all textual data is stored in PostgreSQL[5] to retain efficiency and not incur data access expenses that would otherwise be caused by storing large blob data.

2.1.4 Technology Used in Immersio

There are several AI models available both publicly and open source that yield interest for a language learning application. Because GPT-3.5-Turbo[9] is easily accessible with an API and it can be configured with LangChain[10] to act as a conversational agent it was selected as the conversational agent in this project. Also, we used two pre-trained Hugging Face models for translation and grammar correction. For POS tagging capabilities spaCy[11] was explored. For language de-

[8] https://github.com/coqui-ai/TTS
[9] https://platform.openai.com/docs/models
[10] https://www.langchain.com/
[11] https://spacy.io/

tection CLD2[12] was used. For STT and TTS models capable of speech transcription and speech generation we used Whisper[13] and CoquiTTS[14] respectively.

In order to support dynamic POS generation and POS subtypes various tools and standards were created. One such standard is the Universal Dependencies project, [23] which provides a set of Universal POS tags [24]. The standards provided by the Universal Dependencies extend and categorize POS into seventeen different categories. These categories can express POS in many if not most human languages. Although the Universal Dependencies POS tags cannot cover all lexical and grammatical properties of every language and therefore an additional standard called Universal features [25] was utilized. The Universal features may account for POS that represent numbers, have a particular tense or mood, or much more.

With standards such as the Universal POS tags and the Universal features defined, various tools utilizing this format, such as the Natural Language Toolkit (NLTK)[15] and spaCy[16], have emerged. Such tools have been traditionally used to parse large amounts of corpora but appear to find little usage in language learning applications. However, both of these tools are utilized in Python and are equipped to provide an array of applications for research and industrial usage.

spaCy[17] is a modern NLP tool that is capable of tokenization and tagging text into POS. spaCy[17], compared to NLTK[18], is a newer tool with early versions released as early as 2015[19]. spaCy[17] offers high-level management, training of models, and provides the most highly efficient NLP algorithm. spaCy[17] also

[12] https://github.com/CLD2Owners/cld2
[13] https://github.com/openai/whisper
[14] https://docs.coqui.ai/en/latest/implementing_a_new_language_frontend.html
[15] https://www.nltk.org/
[16] https://spacy.io/
[17] https://spacy.io/
[18] https://www.nltk.org/howto.html
[19] https://explosion.ai/blog/introducing-spacy

utilizes the Universal POS tags and Universal features defined by the Universal Dependencies.

In order to accommodate linguistical principles concerning translation and grammar correction, two Hugging Face pretrained models were considered. One suitable Japanese-to-English translation model from Hugging Face that was considered is `Opus-MT-JA-EN`[20] by Helsinki-NLP. An English grammar correction model that was used is `Coedit-Large`[21] by Grammarly.

Text-To-Speech (TTS) is the process of sending textual data through a TTS pipeline and generating humanlike speech – either through creating an audio file or through an audio stream. TTS typically works through a large language model (LLM), which are AI models trained on vast amounts of data such as publicly available online text or corpora. TTS LLMs typically consist of a voice synthesizer, such as eSpeak[22], which is responsible for generating the speech with a given set of linguistic sequences [26]. Additionally, a TTS LLM typically consists of a phonemizer – especially with support for various human languages. A phonemizer takes the text of a certain language as input and returns phonemes and graphemes as output[23]. A phoneme represents a distinct sound in a human language or human alphabet. A grapheme is the smallest meaningful contrastive unit in a writing system. An English phonemizer can take English text and return a phonetic representation. One phonetic representation that a phonemizer can output is the IPA format by the International Phonetic Association.

Another tool that is utilized in TTS LLMs is called a morphological analyzer. "Morphological analysis may be defined as the process of obtaining grammatical

[20] https://huggingface.co/Helsinki-NLP/opus-mt-ja-en
[21] https://huggingface.co/grammarly/coedit-large
[22] https://espeak.sourceforge.net/
[23] https://docs.coqui.ai/en/latest/implementing_a_new_language_frontend.html

information from tokens, given their suffix information."[24]. One such tool that does this is MeCab[25]. MeCab[25] is a POS and morphological analyzer which is capable of parsing Japanese text into POS. For instance, MeCab[25] can determine nouns, auxiliary verbs, punctuation, and much more[26].

Another tool that is utilized in TTS LLMs are Romanizers. The job of a Romanizer is to convert non-Roman or non-Latin characters into a form that can be read and written in Roman or Latin characters. For example, Nepali, Chinese, Japanese, and Korean words (and to some degree characters) can be represented with Roman characters used in languages such as English ("a", "b", "...", up to "z"). One such Japanese Romanizer used by TTS LLMs is called Cutlet[27] which converts Japanese text to a romanized form known as Romaji.

All of these tools are typically used within a TTS LLM or TTS library such as CoquiTTS[28]. CoquiTTS[28] is a TTS library which offers various tools, pretrained models, and "utilities for dataset analysis and curation" under the MPL-2.0 license[29]. Like most machine learning tools CoquiTTS[28] is written and developed in Python[30]. CoquiTTS[28] offers fine-grain control to engineers by providing tools to help models understand various human languages.

CoquiTTS[28] does an excellent job at producing Japanese speech from provided Japanese text. However, Immersio also needs to support converting Japanese speech to Japanese text. This is done through what is known as Speech-To-Text (STT). STT is the procedure of transforming human speech, usually in the form of an audio file or audio stream, into human readable text. Like TTS, STT typically

[24] https://www.oreilly.com/library/view/natural-language-processing/9781787285101/ch22s05.html
[25] https://github.com/taku910/mecab
[26] https://taku910.github.io/mecab/
[27] https://github.com/polm/cutlet
[28] https://github.com/coqui-ai/TTS
[29] https://github.com/coqui-ai/TTS
[30] https://www.python.org/

consists of an LLM and a pipeline of data transformation where Python data libraries such as pandas[31] and or NumPy[32] are used to handle various datatypes – especially those concerning audio. These underlying tools provide the ability to take an audio clip and convert it into a mel spectrogram, pass it through a convolutional neural network, a transformer pipeline consisting of encoding and decoding, and token prediction. This type of model is known as a sequence-to-sequence (seq2seq) model [27]. There are various open-source tools to create a STT LLM such as CoquiSTT[33] and Whisper[34]. However, CoquiSTT[33] is no longer actively maintained in favor of more popular and potentially more robust STT models such as Whisper[34].

Whisper[34] is a STT tool and Python[35] library developed and maintained from OpenAI[36], which is registered under the MIT license[34]. Whisper[34] is capable of transforming many human languages recorded in the form of audio files into text of a desired language. For instance, the audio file could be a sentence spoken in Spanish but the output could be set to return English text [27]. This is because Whisper is trained on at least 680,000 hours via multitask training. This effectively has made Whisper[34] multi-lingual and is very appealing for language learning applications and tools. Additionally, background noise and interference should not generate garbled textual data and instead will return nothing [27].

Containerization is the process of breaking an application into smaller pieces which are decoupled but can still communicate with each other [28]. Historically, an application's architecture would be monolithic and would not be able to scale

[31] https://pandas.pydata.org/
[32] https://numpy.org/
[33] https://github.com/coqui-ai/STT
[34] https://github.com/openai/whisper
[35] https://www.python.org/
[36] https://openai.com/

efficiently as the application updated its requirements. This traditional architecture has often led to the rigidity of an application where it is difficult to make changes without breaking another piece of dependent code. However, as time progressed, the concept of micro-services has mitigated the effects of a traditional monolithic architecture by reducing coupling, allowing developers to more safely deprecate functionality, and improving scalability. This new micro-service architecture has led to the advent of containerization software such as Docker[37] and Podman[38]. While both of these technologies utilize containerization, Docker[37] appears to have a lot of popularity and has been on the market longer than Podman[38] [29].

With volatile changes in code and the fast-paced advances in AI technology, it may be favorable to contain functionality or AI models as a service. By placing code or AI models in a container a piece of depreciating technology can be removed without too much rework so long as there is a clear interface or API that all the services are to follow. For instance, before Microsoft's GODEL[39] model – a conversational agent – was DialoGPT[40] which preceded it. However, DialoGPT[40] was superseded by GODEL[40] as GODEL[39] had become more performant than DialoGPT [30]. In a situation where a developer decided, first, to utilize the DialoGPT[40] model but found that it would be better to replace it with GODEL[39] then the developer could remove the container containing DialoGPT[40] and replace it with a container containing GODEL[39]. The developer may have to update or create new lines of code but the process may not be as arduous as updating this functionality in a traditional monolithic application.

A message broker is a tool that allows services or micro-services to commu-

[37] https://www.docker.com/
[38] https://podman.io/
[39] https://huggingface.co/microsoft/GODELv1_1largeseq2seq
[40] https://github.com/microsoft/DialoGPT

nicate information with each other [31]. Typically a message broker consists of at least two pieces: a consumer (or subscriber) and a producer (or publisher). This paradigm is what is known as the publish-subscribe (pub-sub) pattern. A producer (or publisher) would send a message to an exchange (or topic) in which the message gets sent to a particular queue (or directly to the publisher in some architectures) in which is consumed by a consumer. What makes communication of a message broker unique is that messages of data can be handled asynchronously. This is to say that clients do not have to wait for a response from a server. Moreover, a message broker allows undelivered messages to be stored on disk when the message broker or a service goes offline. A message broker may have its protocol such as AMPQ[41] or it may implement its own such as gRPC. When paired with containerization tools such as Docker[37], message brokers can decouple a tightly-coupled architecture.

One of the most popular message brokers on the market is RabbitMQ[42] which employs the publish-subscribe pattern. RabbitMQ[42] has what is known as a publisher which sends a message to an exchange within an AMQP broker. The exchange looks at the message label and determines which queue the message should be sent to. Finally, a consumer observes a queue with a particular label and consumes messages from it until the queue is empty or until the consumer goes offline.

2.2 State of the Art

This section discusses the current implementations on the language learning application market. Within this section we explore the usage of the Leitner

[41] https://www.amqp.org/
[42] https://www.rabbitmq.com/

system and other SRS tools. Additionally, the utilization of artificial intelligence in language learning applications are investigated. Furthermore, common techniques used by these applications are explained. Finally, we reveal the limitations of these applications and what could be improved.

2.2.1 Anki

Anki[43] is a free and open-source flashcard system that integrates spaced repetition and the Leitner system. Anki employs a derivative algorithm based on the SM-2 algorithm [32]. However, other algorithms can be utilized instead such as the FSRS algorithm. Anki comes from the Japanese word for memorization (”暗記”). The way Anki works is that a user can create a deck in which they place cards in. Cards can be made within the Anki software or simply imported from another source – typically as file in the SQLite format. A card usually contains a front and a back side in which information such as text, images, and even audio can be placed on both sides. Once a deck full of cards is created a user may review the deck in which a review session starts. During the review session, a user is presented with one card at a time which only shows one side of the card. The user must attempt to actively recall how the information on the side they are presented relates to the information of the side not presented. This is purely a manual cognitive process. The user may choose to flip the card over revealing both sides of the card (so long as the setting to show both sides is enabled). Additionally, the user is prompted to declare how well they recalled the item with several options: ”Again”, ”Hard”, ”Good”, and ”Easy”. Each option determines when the item will be reviewed again where choosing ”Again” shows the card to the user sooner than if they chose ”Easy”. The interface for this is illustrated in Figure 2.5. Once an option is selected

[43] https://apps.ankiweb.net/

the user repeats the process until all items in the deck are reviewed and the items that need to be reviewed again are reviewed until the user selects an option to review it at a later time. This process can be summarized by the diagram of the Leitner system in Figure 2.4.

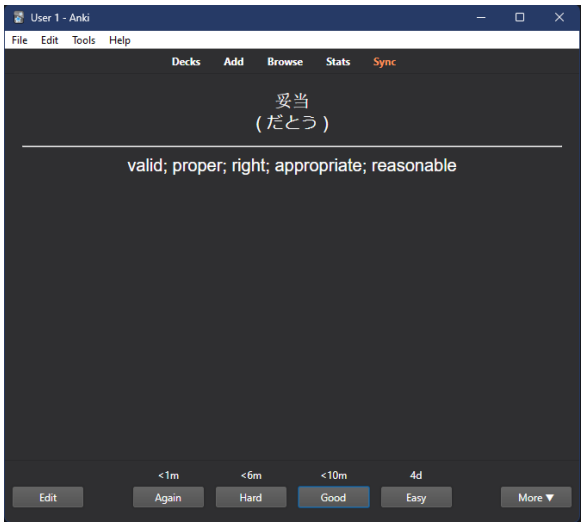

Figure 2.5: The interface as seen in Anki

Anki is a very versatile tool however it comes with several limitations and disadvantages in terms of a language learning application. As Anki is not explicitly a language learning application, Anki does not have native support for dictionary lookup with dictionaries such as JMDict. This can slow the process of both creating and reviewing flashcards. For instance, a user must manually look up words in a reputable dictionary to map their source and target language, then they may manually place their source and target language on the front and back of an Anki card respectively, and finally, they would review the items. Although, if a user would like to look up a definition of a particular word of sentence in the review item then they must look it up manually. A feature to hover or click on a word of the sentence would be very beneficial and efficient for a user during reviews.

Another limitation of Anki is that by default Anki utilizes the SM-2 algorithm although there are algorithms that improve the efficiency of spaced repetition such as the the SM-18 algorithm utilized by SuperMemo. Yet another disadvantage of Anki is that there is no native feature for a conversational agent, any AI tool, or machine learning to assist in making review information dynamic. Such a tool can be quite valuable for organic encounters found when engaging in a foreign target language. Finally, Anki does not have POS highlighting, POS subtypes, or POS lookup for individual words unless a user ”hardcodes” the information into the flashcards via HTML and CSS.

2.2.2 HouHouSRS

HouHouSRS[44] is a free and open source Japanese dictionary that integrates spaced repetition with the JMDict dictionary. It is programmed in C# using Windows Presentation Foundation (WPF) for the user interface with a Model-View-ViewModel (MVVM) architectural pattern. HouHouSRS allows the user to search for a vocabulary item or a Chinese character – known as kanji – formulated by radicals, or pieces of a Chinese character. After searching for a particular word, a user can see the vocabulary meaning, its rarity, its rating on the JLPT[45], its reading, and its definition. The user can also choose to add the search item or items to their reviews which shows up as an item in the SRS tab. The dictionary interface of HouHouSRS can be seen in Figure 2.6.

Once an item is added to a user's reviews a user may click the SRS tab in which the user can iterate over all of the reviews via a Leitner system just as with Anki. Different from Anki, however, HouHouSRS prompts the user to user

[44] https://github.com/Doublevil/Houhou-SRS
[45] https://www.jlpt.jp/e/

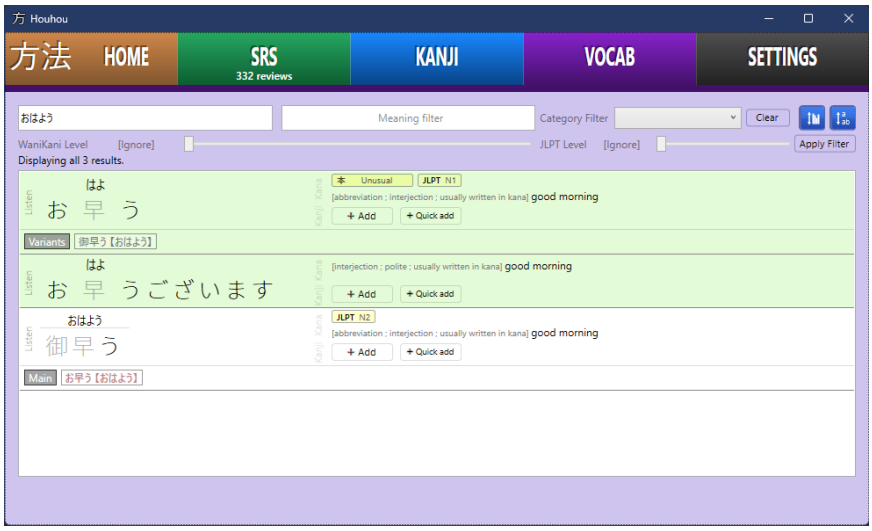

Figure 2.6: The dictionary lookup as seen in HouHouSRS

to type both the meaning and the reading of an item. This is a more powerful form of active recall in that a user must physically and actively type an answer rather than conceptually visualize one. Once the user answers the question one of two interfaces will be shown to the user depending on whether the user was correct or not. The first interface shows the typed answer highlighted in green if the user was correct. Additionally, two other options exist on this interface such as ”Ignore answer” and ”Edit this item”. Selecting ”Ignore answer” skips the card and shuffles it back into the deck. Pressing ”Edit this item” allows the user to manually adjust review item criteria such as the SRS level, add notes, and much more. The second interface appears when a user answers the question incorrectly and displays the user’s typed answer highlighted in red. This second interface will include the ”Ignore answer” and ”Edit this item” buttons with the same functionality as the first interface but also adds an additional option which is either, ”Add to readings” or ”Add to meanings”. The ”Add to readings” option, which only appears if the review item prompted the user for the item reading,

when selected, adds whatever the user typed as an accepted answer, marks their answer as correct, and goes to the next review item in the queue. The "Add to meanings" option provides a similar function with the only difference being that the review item to be answered prompted the user for a meaning instead. The interface in which a user answers a reading item incorrectly can be seen in Figure 2.7.

HouHouSRS is an incredible software inspired by Anki and WaniKani (a tool we explored later) which allows a user to look up an item in a dictionary, review the item via a spaced repetition system, and understand both readings and meaning of a word through a powerful form of active recall. However, HouHouSRS is not without its limitations. One such limitation is that HouHouSRS is not a cross-platform tool as it is programmed in WPF which is only supported on Windows machines. Another limitation is that HouHouSRS does not employ artifically intelligent tools such as a conversational agent.

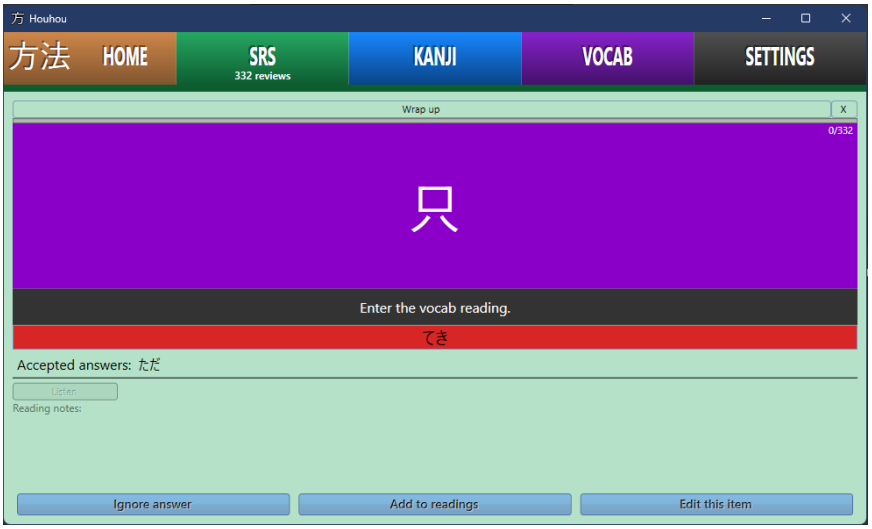

Figure 2.7: The spaced repetition feature as seen in HouHouSRS

2.2.3 WaniKani

WaniKani[46] is a spaced repetition tool inspired by Anki and developed by Tofugu LLC which allows a user to review the most common characters, character radicals, and vocabulary in Japanese. WaniKani utilizes the Leitner system by having various SRS levels. There are 5 ”boxes” in WaniKani’s Leitner system which include ”Apprentice”, ”Guru”, ”Master”, ”Enlightened”, and ”Burned”. ”Apprentice” is the first box a review item will enter and ”Burned” is the box a review item can enter. A review item is said to be ”Burned” when a user understands a review item and no longer needs to review the item. Each of these boxes or SRS levels each have their review intervals calculated by a formula [47]. Whenever a user answers an item correctly the user may advance to the next SRS level. However, if the user answers incorrectly a formula is used to determine how many SRS levels to decrement a review item. The formula can be expressed as: $n = c - (a * p)$ Where n represents the new SRS level, c denotes the current SRS level, a holds the value of the incorrect adjustment count, and a constant, p, which is the SRS penalty factor. The penalty factor, p, is 2 if the SRS level is at 5 or above – otherwise, it is 1.

With a provided SRS algorithm to determine what SRS level a review item is on, another component is needed to determine the review interval, or the time spent waiting for the next time the review item is reviewed. In addition to the five boxes discussed previously, there are sub-boxes which allow fine grain control over review intervals. For instance, ”Apprentice” is sub-categorized into ”Apprentice 1”, ”Apprentice 2”, ”Apprentice 3”, and ”Apprentice 4” while ”Guru” is sub-categorized into ”Guru 1” and ”Guru 2”. The rest of the boxes are not sub-categorized. With this system in place, a review item at ”Apprentice 1”,

[46] https://www.wanikani.com/
[47] https://knowledge.wanikani.com/wanikani/srs-stages/

for example, would wait 4 hours until the next review, and a review item at "Apprentice 2" would wait 8 hours, and so on. This algorithm is implemented in what is known as "Reviews" in WaniKani. During reviews, a user is prompted to provide the meaning and reading of a review item. Figure 2.8 shows what the reviews interface looks like.

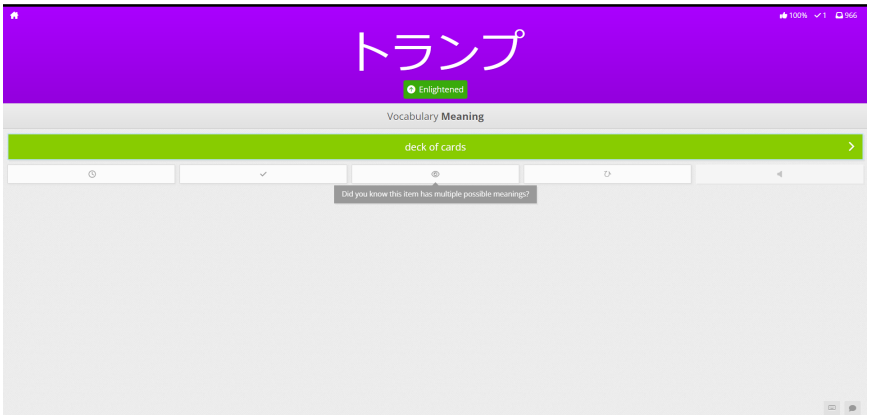

Figure 2.8: The user interface of "Reviews" within the WaniKani web application

WaniKani is a useful application which provides a directed approach to learning Japanese although it is not without limitations. One of WaniKani's limitations is that there is no flexibility to add custom review items. This is to say that it is impossible to add items a user wants to add willingly. Additionally this also means that a dictionary cannot be used to look up an item and add it to a user's reviews. One other limitation of WaniKani is that it does not utilize artificial intelligent tools such as a conversational agent. Additionally, definitions or meanings of review items are hardcoded which means that determining whether a review item was answered correctly or not would mean that a user would have to remember the exact meaning or synonyms associated with the review item – even if the answer they provided is a synonym or is a valid alternative.

2.2.4 Bunpro

Bunpro[48] is a Japanese language learning software which is based in Osaka, Japan. Also like WaniKani, it is based on spaced repetition and the Leitner system. It also contains five boxes which include ”Beginner”, ”Adept”, ”Seasoned”, ”Expert”, and ”Master”. It also has the concept of ”Ghosts” which contains all review items that are missed most frequently. One advantage Bunpro has over WaniKani is that Bunpro adds the flexibility to add custom review items but still allows a user to follow a curriculum. For instance, a user may choose to learn review items that are rated at a higher difficulty or frequency before they learn an item of a lower difficulty or frequency and vice-versa. We can therefore say that Bunpro is difficulty agnostic and not opinionated on a particular curriculum. Another difference from WaniKani is that it is designed to assist a Japanese learner with grammar points – not vocabulary meanings or readings (although recently it has some functionality to achieve this).

Reviews are done by prompting an answer from a user for a fill-in-the-blank question. If the answer to the grammar point is correct, the user can continue to the next item in the review queue or undo their answer and try again (such a functionality does not exist in WaniKani). If the answer is correct and the item is not a ”Ghost”, the item will not be reshuffled in the review queue. A ”Ghost” is a review item that is commonly answered incorrectly during a review session. If the user’s answer was incorrect, they do not choose to undo, and continues, the user will be reshuffled in the review queue to be reviewed again during the same review session. Figure 2.9 encapsulates the functionality of the Bunrpo reviews interface.

[48] https://bunpro.jp/

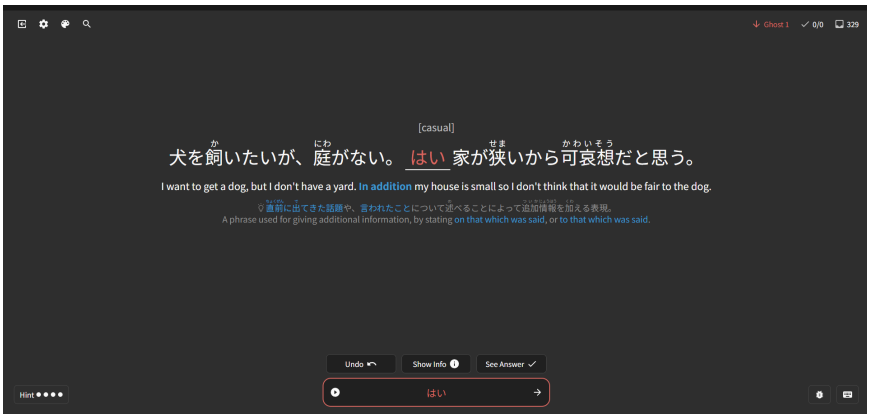

Figure 2.9: The user interface of Bunpro reviews where the user had answered incorrectly

Bunpro is an exciting grammar language learning application for Japanese with several advantages over the previous examples explored in Section 2.2. However, Bunpro also has limitations. One of these limitations is that Bunpro, like WaniKani, has hardcoded examples and POS for each of its grammar points. This means that an administrator would have to add these items manually. A solution to this problem is to add generated examples and POS on demand with tools such as a conversational agent and spaCy. One other disadvantage of Bunpro is that it is designed specifically for grammar that is added manually. One final pitfall that is found within Bunpro is the lack of artificial intelligent tools such as a conversational agent or tools to assist with grading reviews.

2.2.5 Pimsleur

Pimsleur[49] is a language learning application that employs Paul Pimsleur's memory schedule technique, spaced repetition and active recall, and audio lessons. This application is available as a mobile application as well as a web application[50].

[49] https://www.pimsleur.com/
[50] https://www.pimsleur.com/

Pimsleur lessons are designed to be done every day, not skipped, and where the language learner is actively repeating what is spoken to them (shadowing). A narrator guides the language learner through scenarios and prompts them to repeat after the native actor of a target language. For this reason, Pimsleur transfers memory from short-term memory to long term memory. Unlike other language learning applications, which typically assist a user with typing and reading, Pimsleur emphasizes language learning skills such as speaking and listening (although there are also readings lessons).

Pimsleur is a brilliant language learning application that allows a user to rapidly learn a language through consistent practice and spaced repetition. However, there are some limitations to Pimsleur. In essence, the Pimsleur method spaces its lessons into small 30 minute audio clips, which are designed to be completed once per day. There are various challenges that come with how this method is currently implemented in the official Pimsleur application:

- The application must have a team create each audio lesson manually
- An audio lesson must have a hired voice actor to narrate the lesson
- Each audio lesson must be reviewed by an expert in the the end-user's target language
- It is expensive to manage such a large team to generate the lessons
- The application is limited mainly to speaking and listening with few exceptions

However many of these challenges may be resolved by incorporating modern artificial intelligent tools such as Whisper from OpenAI which can transcribe

human speech-to-text (STT). Conversely, CoquiTTS provides the ability to transcribe text-to-speech (TTS). Through the utilization of these tools, it is possible to eliminate the need for hired voice actors, manual labor for review creation, and possibly large expenses incurred from managing a large team. Both of these tools are explained in section 2.1.

2.2.6 Univerbal (Quazel)

Quazel[51] was the name of a multi-language learning application that is now known as Univerbal[52]. It is a relatively new application for both web and mobile devices. Univerbal utilizes AI by including a conversational agent a user can interact with through a messaging interface. In order to access the messaging interface a user must select one of many lessons which each come with a conversation topic. Each topic is generated on-demand by purportedly using ChatGPT.

Once a user selects a lesson from the lessons page (see Figure 2.10), an AI tool will generate a conversation relating to the lesson topic. The user is then greeted with the messaging interface which allows a user to both type and speak to the conversational agent – ostensibly with STT technology. The user may also translate the messages and play the audio of the message with what appears to be TTS technology. In fact, the user may translate each word individually in context. The user may also choose to do tasks which are suggested topics a user can choose to message the conversational agent about during the conversation. The user may optionally speak about these tasks but it is not required to complete the lesson. The user can also over over words to get a translation of a word in context. Finally, when a user exchanges a few messages with the conversational agent, the

[51] https://www.quazel.com/
[52] https://www.univerbal.app/

user may ask the conversational agent to act as a tutor to summarize in detail the conversation as well as provide suggestions on how to use certain grammar points and words. At this point, a user may choose to go over the words from their conversations by reviewing them. However, Univerbal's review feature is not based on spaced repetition but it does prompt the user to answer fill-in-the-blank questions. The messaging interface can be seen in Figure 2.11.

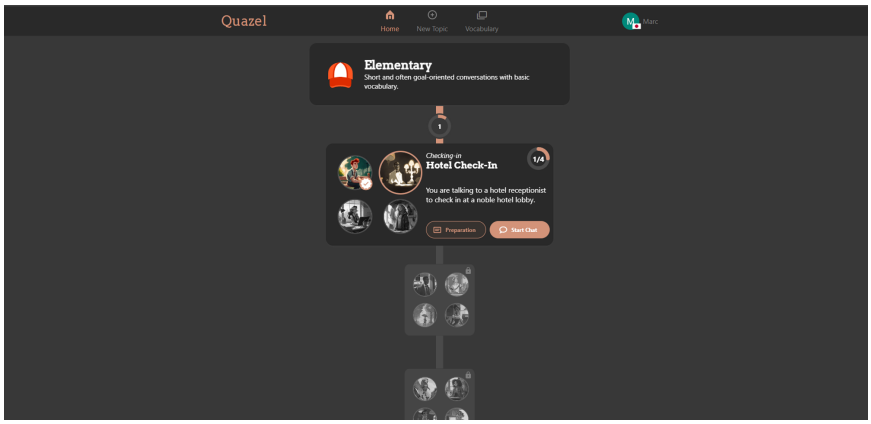

Figure 2.10: The lessons page as seen in Quazel (now Univerbal)

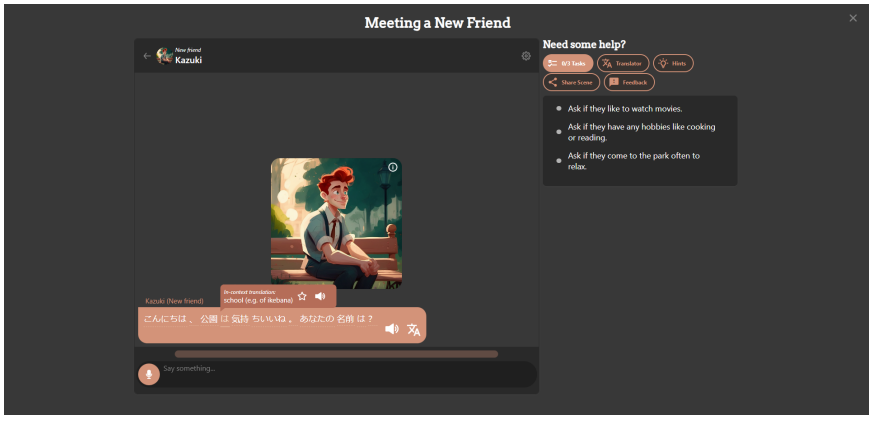

Figure 2.11: The messaging interface for a particular lesson on Quazel (now Univerbal)

Univerbal is a useful language learning tool that utilizes a conversational agent and AI. However, the tool is not without its limitations as a language learning

application. Perhaps the greatest limitation of Univerbal is that reviews do not implement spaced repetition. Instead, reviews are done arbitrarily by the user and how well a user understands or remembers a review item is not implemented well. Another limitation of Univerbal is that it allows a user to speak in any language the user wants – even if the language is not the target language. Such a pitfall inhibits language learning as discussed by Paul Pimsleur in his examples of what make a good language teacher [12].

2.2.7 SuperMemo

SuperMemo[53] is by far the most comprehensive language learning software. SuperMemo is a powerful language learning tool which employs AI tools and conversational agents, power algorithms, such as the SM-18 algorithm, and spaced repetition. Additionally, SuperMemo has years of experience by adding features such as translation and a dictionary. Based on the work of Piotr A. Wozniak, SuperMemo utilizes a flashcard system to learn a language. Furthermore, flashcards are automatically generated for the user to review after a conversation with a conversational agent. Aside from the flashcard feature and spaced repetition integration, SuperMemo shares many of the features Univerbal has such as the ability to both speak and type when responding to the conversational agent, translation, and listening to audio. One difference from Univerbal is that SuperMemo allows the user to manually select words from the conversation that they would like to add to their reviews. Univerbal, on the other hand, picks words for the user automatically. The messaging interface is shown in Figure 2.12.

In order to review words selected in a conversation, the user selects a deck of cards in which a review session would start. During the review session, a

[53] https://www.supermemo.com/en

Figure 2.12: The conversation interface on SuperMemo

user encounters the words they had added from a previous conversation with the context of the word provided below. The user is prompted to pronounce the word in their target language in which they may grade manually based on how they think they did. The grading system is similar to Anki's except that it contains a green smiling face, a yellow indifferent face, and a red frowning face. Each option determines when the next review occurs. A user may also add notes to each of the flash cards if necessary. Additionally, the user may utilize a built in dictionary and request aid from an AI assistant which, like Univerbal, appears to be utilizing ChatGPT. SuperMemo's flash card reviews can be seen in Figure 2.13.

SuperMemo utilizes both spaced repetition, AI, and dictionaries. One of the greatest limitations of SuperMemo is that there is no POS highlighting when conversing with the conversational agent. A lack of such a feature inhibits a user from determining the usage of the word in the sentence. Additionally, SuperMemo does not break up difficult sentences until after a conversation is completed, which is inefficient. Another limitation of SuperMemo is that during a conversation

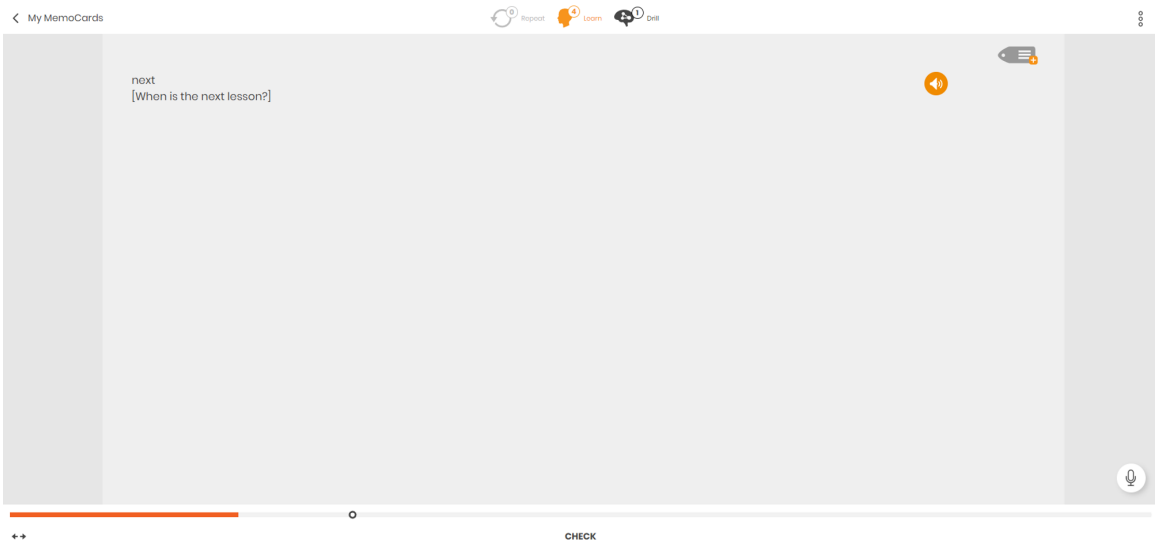

Figure 2.13: SuperMemo "MemoCards" review which provide context for the word seen in a previous conversation

the user can freely message in a language other than the language he or she has intended to learn.

2.2.8 Discussion

One of the most common limitations of the tools presented in this section is the lack of artificial intelligence – specifically conversational agents. Without such functionality, a language learning application can quickly become rigid and only allow what is added by a systems administrator to be learned. However, by providing a dictionary and a conversational agent, a user can have more flexibility on what to learn. One of the greatest limitations that all of the implementations presented in this section has is a lack of POS highlighting with dictionary lookup. Even tools such as Univerbal and SuperMemo – which utilize a conversational agent – fail to provide an easy POS lookup for words with a reputable dictionary. Instead, applications such as these use a translator for words in context, which

relies on the accuracy of the AI tool rather than user intuition. Such intuition is critical to a user's understanding of their target language. However, the most critical limitation in the explored applications is that spaced repetition is not utilized.

Chapter 3

Methodology

In order to resolve the limitations of current language learning tools, an immersive language learning application, Immersio, was created. This tool is composed of three different pages to encapsulate spaced repetition functionality, a conversational agent, and a dictionary. It also contains a review section, a chat page, and a dashboard page.

3.1 Create the Review Page

The review section is the core of the spaced repetition system which has the role of keeping track of all of the user review items. A review item may be a sentence, a word, or a POS. The user is also able to create and review a custom review list consisting of unique items they choose to add. A user may also choose to review all of the user's sentence review items, all of a user's word review items, or all of the review items in general. This functionality can be selected on the landing page of the review page as seen in the Figma mockup in Figure 3.1.

Once a user chooses one of these options the user is greeted with a review item

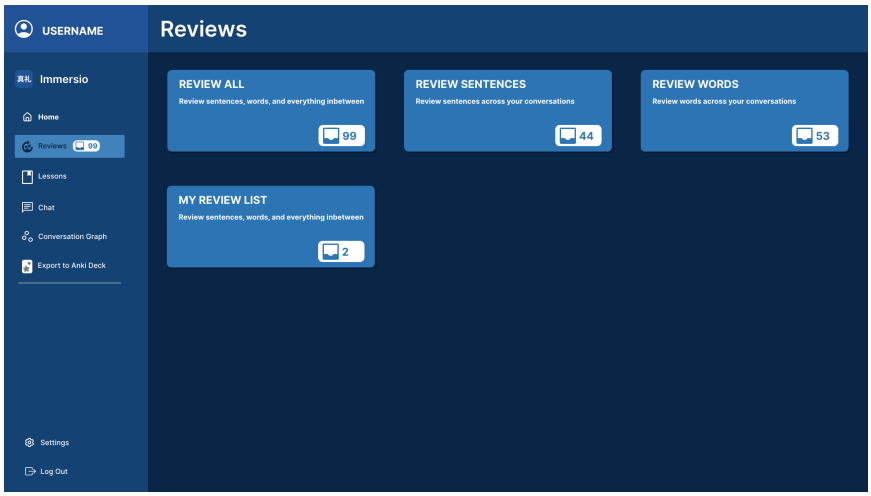

Figure 3.1: The landing page for the reviews

confirmation page which allows them to individually select what items will be placed in their review queue. The review queue is a priority queue that orders items first by the earliest due date and with a lower SRS level. A SRS level is a property of a review item that determines the review frequency and ultimately the next due date. The user also has the option to delete the review item, reset its SRS level, and delete all of their review items. Additionally, the amount of reviews that are due are displayed on this page as well as to the left on the navigation drawer. The end results are similar to the Figma mock-up in Figure 3.2.

Once the user starts the review session, the user is prompted to translate the first item that is popped from the review queue. The user then submits their translation in which CLD2[1] compares a user's translation to its own translation. The mockup for this description is seen in Figure 3.3.

Once the backend compares the user's translation with its own translation, the conversation context is displayed. This conversation context shows whether or not the user was correct in their translation. Specifically, the conversation context

[1] https://github.com/CLD2Owners/cld2

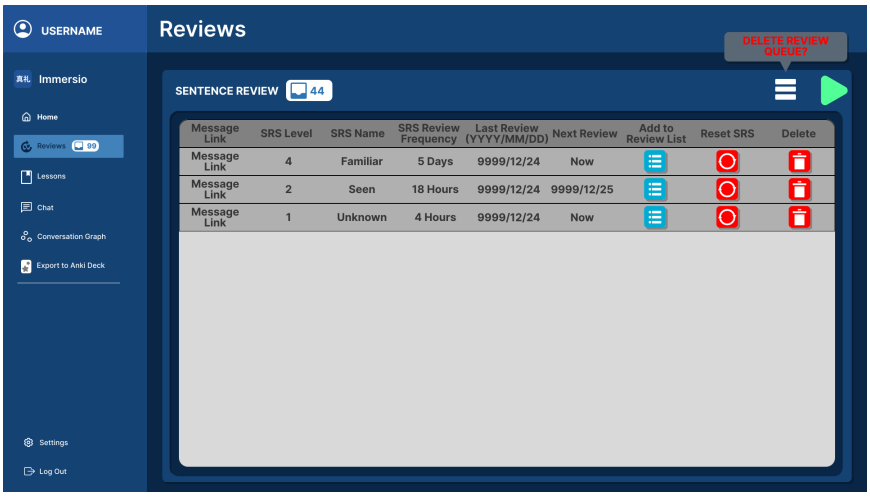

Figure 3.2: The confirmation page for selecting reviews via a table

Figure 3.3: The translation prompt a user sees when doing one item of their reviews

holds the purpose of showing the user's message they added in context to the conversation they had with a conversational agent. In case of suspicion of an artificial grading failure, the user may also choose to assert that their translation was correct or incorrect. The user also has the option to toggle showing the corrections made by the artificial intelligent agent. Finally, the user can press a button to continue to the next prompt (see Figure 3.4).

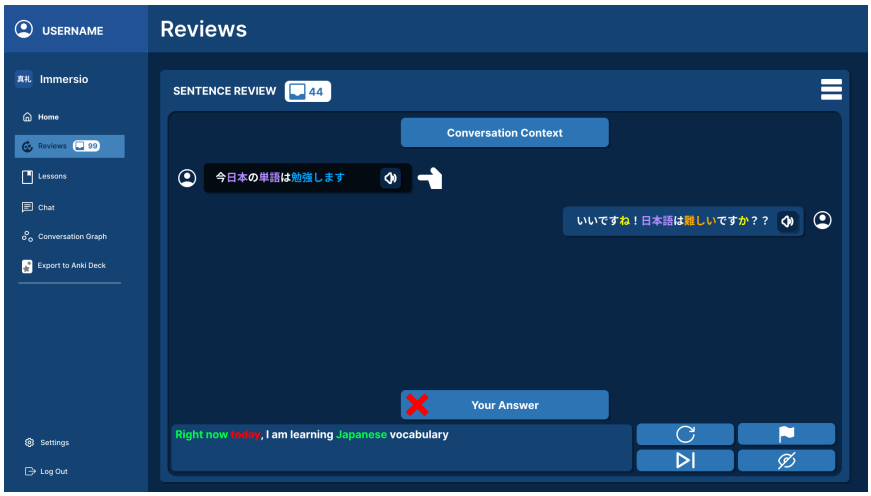

Figure 3.4: The conversation context that is shown to a user

Finally, after several interactions between translation prompts and conversation contexts, the user lands on the report page which shows what review items they had answered correctly or incorrectly. This iterative process can be summarized in the illustration of a finite state diagram in Figure 1.1. On this report page, the user sees the percentage of questions answered incorrectly over the total amount of questions. The user is also able to select individual review items that were either correct or incorrect. The user can also return to the dashboard by clicking on the home icon. The diagram for the reports is shown in Figure 3.5.

The frontend of Immersio was constructed utilizing React[2] and TailWindCSS[3] for rapidly designing UI elements. The React frontend communicates with the Django backend via authenticated REST API. Listing 3.1 provides an example of the REST API JavaScript code for retrieving a user's review items by review item type, via a path parameter, denoted by the `reviewVariant` variable. Once the backend server responds with a JSON response of the user review items, the items get set to a React state that is defined as `reviewItems`. React state allows the

[2]https://react.dev/
[3]https://tailwindcss.com/

Figure 3.5: The report that is displayed once a user completes all of the reviews for their review session

HTML document object model (DOM) to change reactively if the state changes. This is to say that the user interface changes and re-renders in the browser. In this case, the `reviewItems` state was updated with `setReviewItems`.

```
// Retrieve the data from the server
const getReviewItems = async (reviewVariant: ReviewVariantType) => {

    const reviewVariantString: string = reviewVariant.toString()

    // Get reviews
    const getReviewsResponse = await fetch(`${__VITE_SERVER_URL__}/
        api/v1/review/${reviewVariantString}/`,
    {
        method: 'GET',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${user.access}`
        }
    })
```

```
15      const getReviewsJsonResponse = await getReviewsResponse.json()
16
17      setReviewItems(getReviewsJsonResponse)
18
19      // Primarily for selection boxes
20      setReviewItemsSelected(new Array(getReviewsJsonResponse.length).
            fill(true))
21  }
```

Listing 3.1: JavaScript code that utilizes REST API to retrieve user review items

3.2 Create the Chat Page

The chat page implements a conversational agent with both TTS and STT technology, POS parsing, and translation capabilities with grammar correction. In order to do this, we created an information hierarchy where a user can have many conversations and a conversation can have many messages. This hierarchy is described in PostgreSQL tables. Unlike language learning applications, which do not enforce conversing in the user's target language, we prevented the urge for the user to use another language. We accomplished this by having an AI tool, that resides in the Django backend[4], to check the language of the user and determine if they had been speaking in their target language. The AI tool that was used is Compact Language Detector 2 (CLD2)[5]. If the user is found to not speak in their target language, no message will appear and the frontend displays a toast message that tells the user that they must speak in their target language. At the chat page's most basic level, the user is to send messages (in the user's target language) to

[4]https://www.djangoproject.com/
[5]https://github.com/CLD2Owners/cld2

the Django backend where OpenAI's GPT-3.5-Turbo model[6] generates a response message in the user's target language and is then returned to the client. The interface for this can be seen in Figure 3.6. Communication with GPT-3.5-Turbo is possible via a generated API key which is provided as a service by OpenAI. However, GPT-3.5-Turbo alone is not capable of acting as a conversational agent because the model does not retain information by default. For instance, if a user attempts to ask the conversational agent to recall their name later in a conversation, GPT-3.5-Turbo responds by stating that they do not know or forgot. For this reason, we used LangChain[7] to provide memory for the conversational agent. This is accomplished through a `ConversationBufferMemory`, as provided by LangChain[7], and PostgreSQL[8]. The database has a dedicated table which stores the `session_id`, a unique identifier of a conversation, and the message text, and time. We also generated audio via CoquiTTS[9] and stored the audio locally in IndexedDB[10] to save server resources. The audio resembles an elderly Japanese man who speaks a Japanese sentence whenever the user presses the audio button near a message (see Figure 3.6).

The user, additionally, has the ability to translate messages on the interface by hovering over a translate button as seen in Figure 3.7. The translation shows up as a tool-tip in this case. If the user desires to continually display the translation on the screen without hovering, the user may press the translation button as seen in Figure 3.8. Pressing the translation button toggles displaying the translation below the actual message. This translation feature is implemented with the conjoined work of pre-trained hugging face models. Whenever a user presses the translation

[6] https://platform.openai.com/docs/models/overview
[7] https://www.langchain.com/
[8] https://www.postgresql.org/
[9] https://github.com/coqui-ai/TTS
[10] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

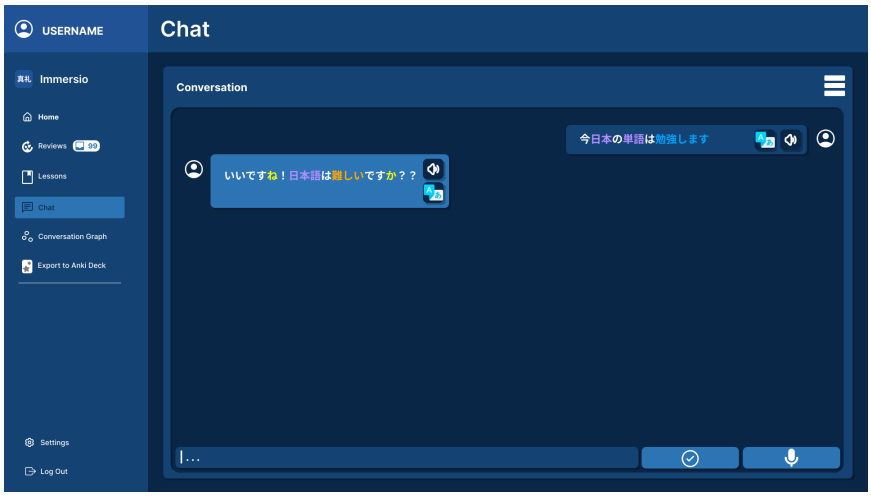

Figure 3.6: The chat interface a user is able to use to communicate with a conversational agent

On Hover:

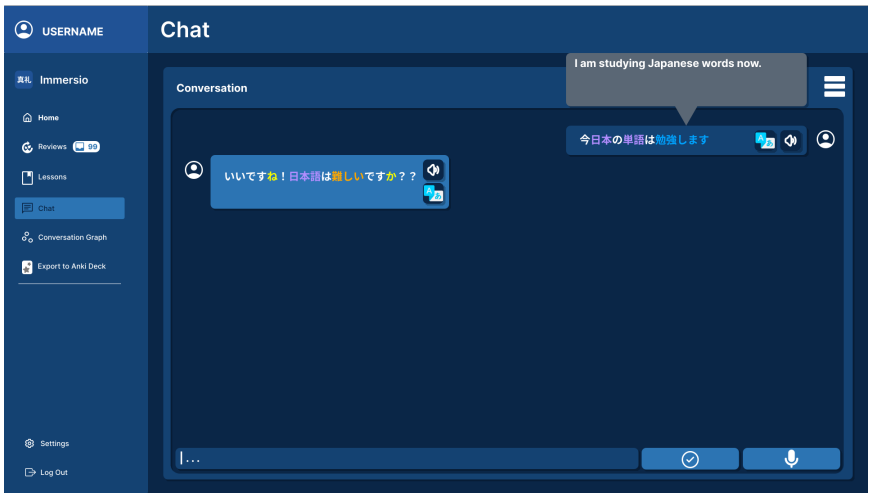

Figure 3.7: Translation occurs when a user hovers over the translation button of a message

button a REST API is invoked in which the Japanese message gets passed to Django. Once Django receives the translation API request, it sends it to a Docker container via a RabbitMQ Remote Procedure Call (RPC) that contains `Opus-Mt-JA-EN` which translates the Japanese text into English. Once the text is translated, it is pipelined

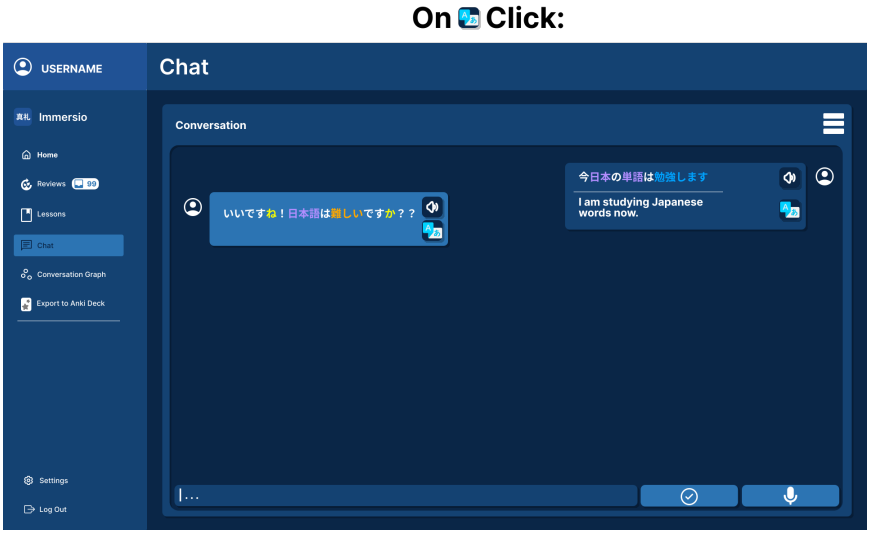

Figure 3.8: Translation may also be continually displayed on the window when a user presses the translation button

into another model to fix any grammar mistakes. This model is `Coedit-Large` by Grammarly. Once the text has gone through this grammar correction phase it is sent back to the Django Backend via a constructed RabbitMQ RPC. Finally, the translated and grammatical text is returned to the React frontend where it can be displayed and styled using TailwindCSS.

spaCy is utilized to provide dynamic POS tagging of generated words by GPT-3.5-Turbo. Like the other pre-trained models in Immersio, spaCy is placed in Docker. Communication happens through RabbitMQ RPC. With this configuration, generated Japanese text is highlightable by individual POS. For example, a generated GPT-3.5-Turbo Japanese message is split up into verbs, nouns, determiners, and punctuation to name a few. By hovering over these highlighted POS terms, the text is enlarged to draw focus on that particular POS. Additionally, the user is able to click on a POS item in which a modal, or popup, is displayed. This modal shows the user the type of POS the item is along with providing definitions from the JMDict dictionary using JMDictDB. In order to communicate with the spaced

repetition ability of the reviews page, a button on the modal can be pressed in order to add the POS item as a review item. This review item shows up in the reviews page whenever its due date has passed.

Another feature on the chat page is the ability to send audio messages to the backend which Whisper is to convert audio into text. IndexedDB saves the audio locally to preserve server resources. So long as the user enables microphone access for Immersio, the interface starts the recording state. This functionality is illustrated in Figure 3.9.

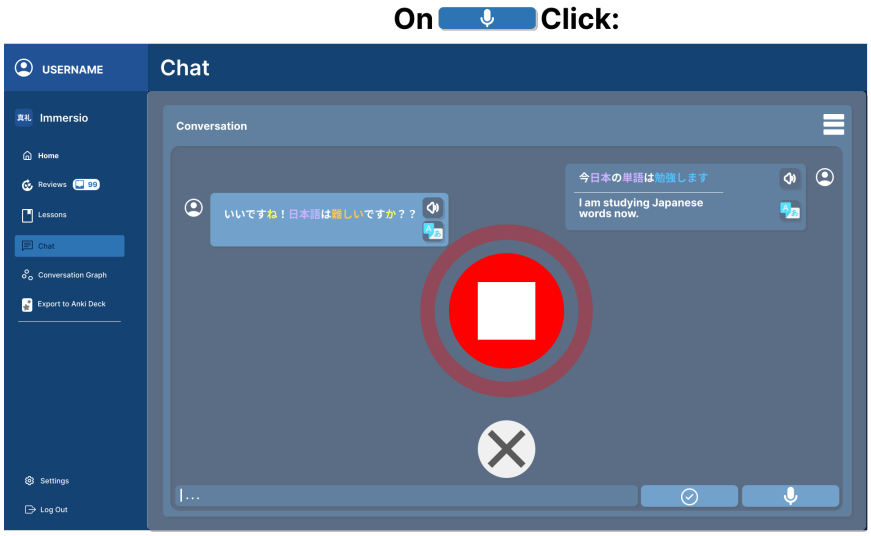

Figure 3.9: The audio interface when the audio button is pressed

When the stop button and the send button is pressed, the resulting audio is represented by a base64 string which is designed to be sent over REST API to where it is handled by Django, pipelined through a RabbitMQ RPC, and transcribed with Whisper. The transcribed text, which is in Japanese, then goes back to Django where it is used by GPT-3.5-Turbo for Japanese text generation. After GPT-3.5-Turbo generates its AI response, Django creates two messages in a PostgreSQL table: one for the user's message and one for the AI's response. Then, Django

publishes to two different RabbitMQ queues to handle POS parsing and translation. However, unlike the previous RabbitMQ RPC, Django does not wait for a response from the translation and POS service but instead packages a response to send to the React frontend. The React frontend then renders the messages once Django responds – regardless of whether or not the POS or translation data has been created. The user audio is also saved locally in IndexedDB so that the user does not need to record again so long as the user does not clear their IndexedDB data. The audio button is designed to record the audio in the case that the user would like to re-record their message.

The audio overlay that is displayed when the stop button is pressed is illustrated in the Figma diagram in Figure 3.10. At this point, the audio data is recorded and the user can choose either to re-record, close the audio overlay, or press the send button which sends their audio to the backend in which two messages are generated. If the user presses the microphone button, the user can re-record their audio. If the user presses the "X" button at the bottom left of the audio overlay, the audio overlay will close. If the user presses the send button at the bottom right of the audio overlay, the user's audio data will be sent to the backend to generate two messages and the audio overlay will eventually close. If either the "X" or send button is pressed the audio overlay will close as seen in Figure 3.11.

In order to make rapid progress in development working with IndexedDB, Dexie.js[11], a high-level IndexedDB API wrapper, was used. Dexie.js is available as a yarn[12] or npm[13] package and therefore can be imported as shown in `line 1` in Listing 3.2. In order to create an IndexedDB datastore, we create a class that extends from Dexie and created a constructor in which we call `super()` with the

[11] https://dexie.org/
[12] https://yarnpkg.com/
[13] https://www.npmjs.com/

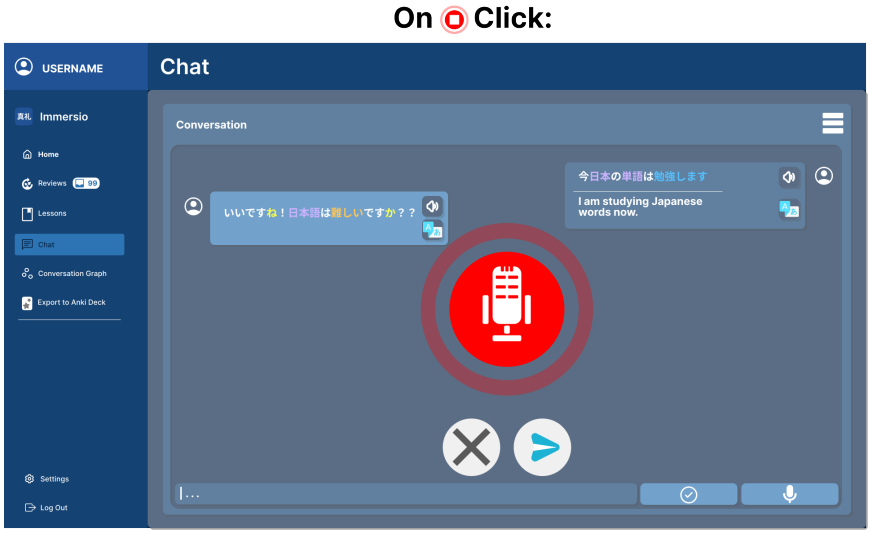

Figure 3.10: The audio interface when the audio button is stopped

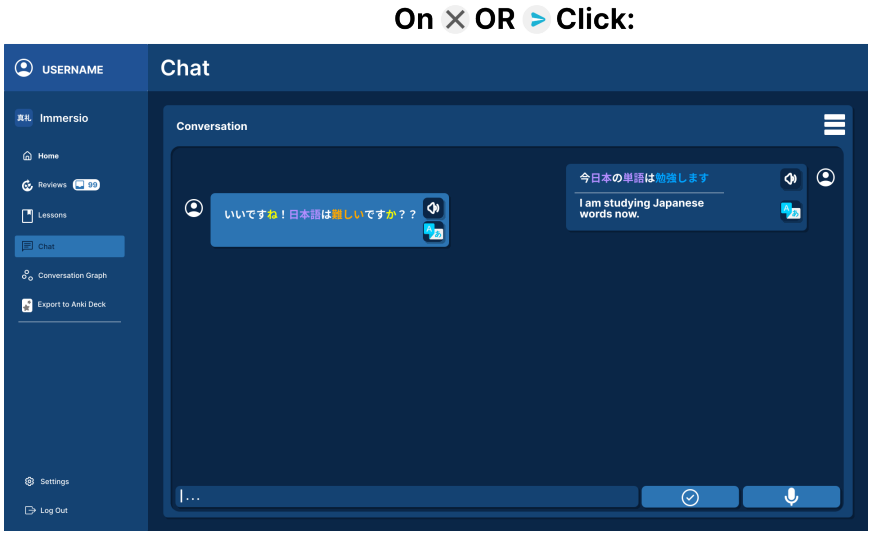

Figure 3.11: Whenever the cancel or send button is pressed the user will return to their chat

name of the database passed ('UserData' in this case). Finally, we created a version of the datastore along with fields or columns. A datastore acts similarly to a table in traditional relational databases. Because the datastore stores audio data, the `AudioData` datastore contains the associated `messageID` and `base64Audio` along with an auto-incrementing `id`. Because accessing IndexedDB is an asynchronous

operation it is necessary to use JavaScript `await` and `async` syntax when adding new base64 audio data. Otherwise, adding data to a IndexedDB datastore is as simple as calling `IndexedDB.AudioData.add()` where data store field values are passed as a JSON object.

```
1  import Dexie, { Table } from 'dexie'
2
3  interface IAudioData {
4      id?: number
5      messageID: number
6      messageAudio: string
7  }
8
9  /**
10  * Add audio data to IndexedDB
11  */
12 export const addAudioData = async (messageID: number, messageAudio:
       string) => {
13     let successfullyAddedToIndexedDB = false
14
15     try {
16         // Add the new audio data
17         const id = await IndexedDB.AudioData.add({
18             messageID,
19             messageAudio
20         })
21
22         successfullyAddedToIndexedDB = true
23
24         console.log("added audio. . . ")
25     }
```

```
26      catch (error) {
27          successfullyAddedToIndexedDB = false
28          console.log(`Error: ${error}`)
29      }
30      return successfullyAddedToIndexedDB
31  }
32
33  // Create a database
34  class DexieIndexedDB extends Dexie {
35      // Tell the typing system to use our interface for declaring our
            store
36      AudioData!: Table<IAudioData>
37
38      constructor() {
39        // Create a datastore called UserData
40        super('UserData')
41        this.version(1).stores({
42          // Primary key and indexed props
43          AudioData: '++id, messageID, base64Audio'
44        })
45      }
46  }
47
48  const IndexedDB = new DexieIndexedDB()
49  export default IndexedDB
```

Listing 3.2: Example of creating a datastore in IndexedDB using Dexie.js

3.3 Create the Dashboard Page

The dashboard page is a page that provides quick links to the reviews and chat page. The dashboard also shows high-level information of the user's progress, a heat map, and recent messages. The mock-up for the dashboard can be seen in Figure 3.12.

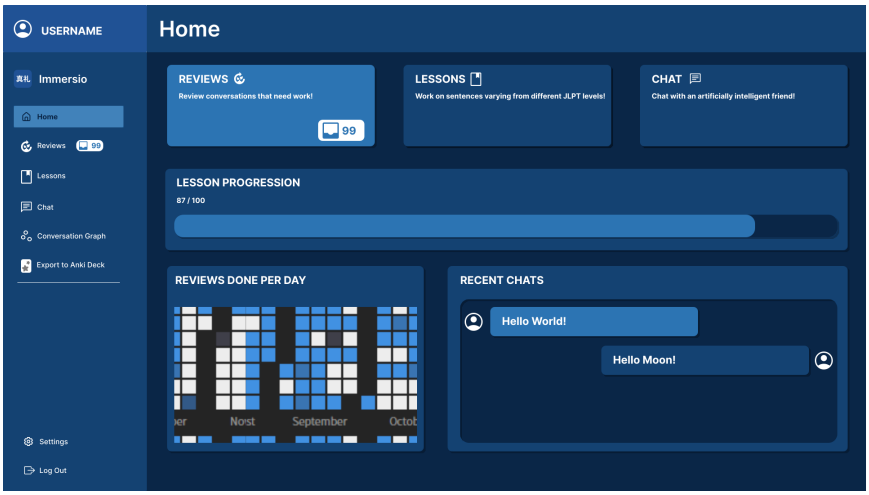

Figure 3.12: The dashboard page

Chapter 4

Testing/Evaluation Plan

Immersio was tested with automated and manual testing to ensure optimal quality assurance. Immersio's automated testing consisted of using frontend testing tools that can be used in React. Immersio's manual testing consisted of a Japanese evaluator who navigated through the application with guided instructions and filled out an evaluation form.

4.1 Automated Testing

In order to perform automated testing on Immersio, we utilized integration testing and end-to-end testing. Testing was done primarily on the frontend as testing user experience is what we have determined to be most important. However, backend tests were also employed. We used testing tools such as PyTest[1], Mock Service Worker[2], Django's tools for automated testing[3], Vitest[4], React Testing Library[5],

[1] https://docs.pytest.org/en/7.4.x/
[2] https://mswjs.io/
[3] https://docs.djangoproject.com/en/5.0/topics/testing/
[4] https://vitest.dev/
[5] https://testing-library.com/docs/react-testing-library/intro/

and Cypress[6]. We performed black-box testing where we intended to test only what the user can see or interact with. The majority of our automated tests did not comprise of unit testing but integration and end-to-end testing.

4.2 Manual Testing

In order to test the accuracy of Japanese used by the AI models, Immersio was tested by an evaluator who is fluent in Japanese via a three part evaluation plan including reading an instruction document, interacting with the application, and filling out an evaluation form. The instruction document was created via Google Docs[7] as it makes distributing a read-only document online feasible. The application was accessed during the evaluation period for interaction at a publicly available link. Finally, the evaluation form was a survey publicly available on Typeform[8] designed to be used in conjunction to Google Docs[9].

The instructions document that was created on Google Docs was intended to explain the basic functionality of Immersio, to provide evaluators of expectations of how to evaluate the project, and most importantly, instructions on how to navigate throughout the application. The instruction document also made references to the evaluation form and made suggestions to the evaluator on what to look for during the interaction with Immersio. The instruction document was organized into various sections including: a description of what Immersio is and what it is supposed to do, an overview detailing what to expect in the instruction document as well as suggestions for the evaluator, a "Getting Started" section to inform the

[6] https://www.cypress.io/
[7] https://www.google.com/docs/about/
[8] https://www.typeform.com/
[9] https://docs.google.com/document/d/1SI1Yc4etJZwPkRpsekWY7ywi6KSBSOfzPVTz4wKN6$_M$/*edit?usp = sharing*

evaluator how to navigate to Immersio and how to sign up, a section to inform the evaluator how to evaluate the chat page, a section to inform the evaluator how to evaluate the review page, and finally a "Wrapping Up" section to ensure that the evaluation form was completed and to thank the user for evaluating Immersio.

Immersio is publicly available on Railway[10] for the evaluator to interact with the Japanese spoken by the AI models. The evaluator had read and typed Japanese text, spoke Japanese, and listened to Japanese audio. The application was deployed with the intention that the evaluator signs up to learn Japanese as an English speaker. By doing so, the evaluator was expected to only speak Japanese when conversing with the AI models on the chat page and expected to translate in English during a review session on the review page. By navigating throughout the application, the evaluator had been expected to take the translation accuracy, grammatical correctness, and proper register or tone of the Japanese used by the AI models into account in order to complete the evaluation form.

There were four main sections on the evaluation form. The first section only consisted of a single question that asks the evaluator how well they understand Japanese on a scale from one to five and can be seen in Table 4.1. One being the most fluent and five being a beginner or a new Japanese learner. The scale is based on the Japanese Language Proficiency Test (JLPT)[11] metrics that rank Japanese language proficiency from levels N1 to N5. The second section asked the evaluator to evaluate Immersio's chat page by asking questions in Table 4.2. The third section contains questions concerning how the reviews page performed in terms of Japanese accuracy and can be seen in Table 4.3. The fourth section asks open-ended questions that allow the evaluator to provide feedback on the

[10] https://immersio.up.railway.app/
[11] https://www.jlpt.jp/e/about/levelsummary.html

application's abilities as noted in Table 4.4.

Question in English	Question in Japanese	Scale
How well do you speak Japanese?	日本語をどの程度お話しできますか？	1-5

Table 4.1: The evaluation form's first section's question

Question in English	Question in Japanese	Scale
How accurate/natural was the application's chat page's AI's Japanese response?	アプリのチャットページのAIの日本語応答はどれくらい正確/自然でしたか？	1-5
How well has the AI adhered to the proper register, tone, or mood? (Did the AI respond casually or formally; For example: Did the AI use です or ます when appropriate?)	AIが適切なレジスター、トーン、またはムードにどれくらい適合していましたか？（AIはカジュアルにまたはフォーマルに応答しましたか？ 例：AIは適切な場面で「です」または「ます」を使用しましたか？）	1-5

continues on next page

How accurate was the AI's translation?	AIの翻訳の正確さはどの程度でしたか？	1-5
How accurate was the AI's translation at being grammatically correct?	AIの翻訳が文法的に正確であったかどうか、どの程度でしたか？	1-5
How accurate were the dictionary definition(s) for a given word? (The dictionary is activated by clicking on a word in a message)	特定の単語の辞書の定義は、どの程度正確でしたか？（メッセージ内の単語をクリックして辞書を表示します）	1-5
How accurate was the AI generated audio when speaking Japanese?	AIが生成した日本語音声の正確さはどの程度でしたか？	1-5
When speaking Japanese using the microphone button, how well did the AI transcribe the text into Japanese?	マイクボタンを使用して日本語で話した場合、AIがテキストをどの程度正確に日本語に転写しましたか？	1-5
When speaking Japanese using the microphone button, how grammatically correct was the AI as it transcribed the text into Japanese?	マイクボタンを使用して日本語で話した場合、AIがテキストを日本語に転写する際の文法の正確さはどの程度でしたか？	1-5

continues on next page

How well has the AI remembered information previously mentioned in a conversation?	AIが以前の会話で言及された情報をどの程度覚えていましたか？	1-5
How well did the AI prevent talking/messaging in English and only allowed conversing in Japanese?	AIは、英語での話しやメッセージの防止をどの程度うまく行い、日本語での会話のみを許可しましたか？	1-5

Table 4.2: The evaluation form's second section's questions for the chat page

Question in English	**Question in Japanese**	**Scale**
During a review session, how well was the AI able to grade the user's translated sentence?	レビューセッション中、AIはユーザーの翻訳された文章を評価するのにどの程度うまく対応しましたか？	1-5
During a review session, how well was the AI able to grade the user's translated word?	レビューセッション中、AIがユーザーの翻訳した単語を評価するのにどの程度うまく対応しましたか？	1-5

continues on next page

During a review session, how accurate was the AI able to generate the translation of a sentence/message?	レビューセッション中、AIが文やメッセージの翻訳を生成する能力においてどれくらい正確でしたか？	1-5
During a review session, how accurate was the AI able to generate the translation of a word?	レビューセッション中、AIが単語の翻訳を生成する能力においてどれくらい正確でしたか？	1-5
How well did the AI prevent translating the Japanese text into a language other than English? For instance, the AI should ideally not allow the usage of Japanese when translating and should only allow English text during the translation of a review item.	AIが日本語のテキストを英語以外の言語に翻訳するのをどれくらい防止しましたか？たとえば、AIはレビューアイテムの翻訳時に日本語の使用を許可せず、英語のテキストのみを許可すべきです。	1-5

continues on next page

During a review session, how well was the corrected answer annotated?	レビューセッション中、修正された回答が適切に注釈付けされていたかどうか、どの程度でしたか？	1-5
How correct was the accuracy report after the reviews were finished? Ideally, the accuracy report should mark a message as incorrect if a review item was marked incorrect at least once and the accuracy report should count a message as correct if it a review item was marked correct the first time.	レビューが終了した後の正確性レポートはどの程度正確でしたか？理想的には、正確性レポートは、少なくとも1回のレビューアイテムが誤ってマークされた場合にメッセージを不正確としてカウントし、レビューアイテムが最初に正しくマークされた場合にメッセージを正確としてカウントすべきです。	1-5

Table 4.3: The evaluation form's third section's questions for the reviews page

Question in English	Question in Japanese	Scale
How well did the chat page perform? What did you like or dislike about the chat page?	チャットページ全体の性能はどの程度でしたか？チャットページについて好きな点や気に入らなかった点は何ですか？	Open-Ended
What do you think could be improved on the chat page?	チャットページを改善するためには、どのような点が改善されるべきだと思いますか？	Open-Ended
How well did the review page perform? What did you like or dislike about the review page?	レビューページ全体の性能はどの程度でしたか？レビューページについて好きな点や気に入らなかった点は何ですか？	Open-Ended
What do you think could be improved on the review page?	レビューページを改善するためには、どのような点が改善されるべきだと思いますか？	Open-Ended
How you think this application could be improved?	このアプリケーションをどのように改善できると思いますか？	Open-Ended

Table 4.4: The evaluation form's fourth section's questions for open-ended responses on the application as a whole

Chapter 5

Results

This chapter discusses the finalized application, Immersio, and the evaluation results of a Japanese speaking person who evaluated the tool. Section 5.1 explains the process of navigating Immersio from the login and registration page to the chat and review page which are the core components of the application. Sequential instructions are provided as well as explanations to events that occur due to user input. Additionally, the evaluator's evaluation form responses are shown in Section 5.2.

5.1 Description of Immersio

Figure 5.1 shows the architecture of Immersio. The frontend is comprised of a Single Page Application (SPA), built upon React, which contains static HTML, CSS, and JavaScript files. IndexedDB is also present on the frontend to store audio data. The frontend communicates with the Django backend through REST API in which JSON requests and responses are sent and received. Django acts as the intermediary between the frontend and the rest of the backend services. Django

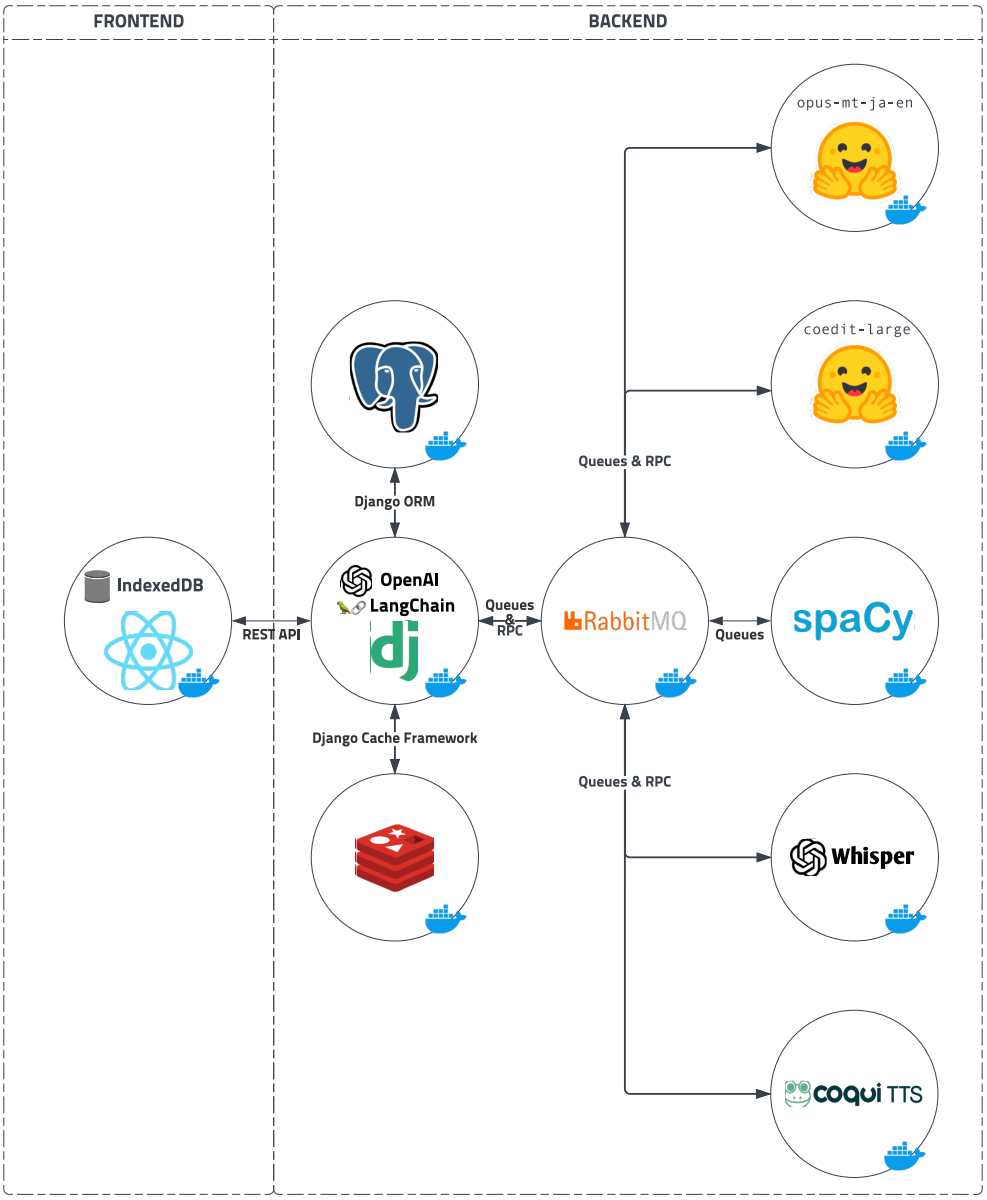

Figure 5.1: The architecture of Immersio

is also responsible for handling access to user data in PostgreSQL and Redis. Django communicates with the backend AI services through RabbitMQ queues and RabbitMQ RPCs. In the case that RabbitMQ RPCs are not used as a means for communication between the services and Django, Django uses Python threads to handle consuming from the various services. The main Django thread only needs

to publish to RabbitMQ queues. In terms of user data, Django communicates with PostgreSQL through its ORM. Django's ORM performs high-level Create, Read, Update, and Delete (CRUD) actions on user data without writing raw SQL queries. Django also uses its ORM to retrieve dictionary data from JMDictDB which resides in the PostgreSQL database. In order to hold temporary information that does not need to be perpetually stored, such as a user's review session, Redis and Django's cache framework is employed. Anything else requested by the frontend, such as the text generation from OpenAI's GPT-3.5-Turbo model and access to conversation memory by LangChain, is handled by Django directly

When a given user first attempts to use Immersio they are greeted with a landing page as seen in Figure 5.2 followed by a registration page, by clicking on the "Try Immersio" button, in the top right corner, in which they can choose a source language. The source language is the language the user chooses to learn a target language from. In Immersio's case, English is to be selected for the source language as seen in Figure 5.3.

After selecting a source language, the user is able to select a target language. The target language is the language the user intends to learn. Because Immersio is designed to cultivate Japanese language learning among English speakers, Japanese is the language a user will select as a target language as seen in Figure 5.4.

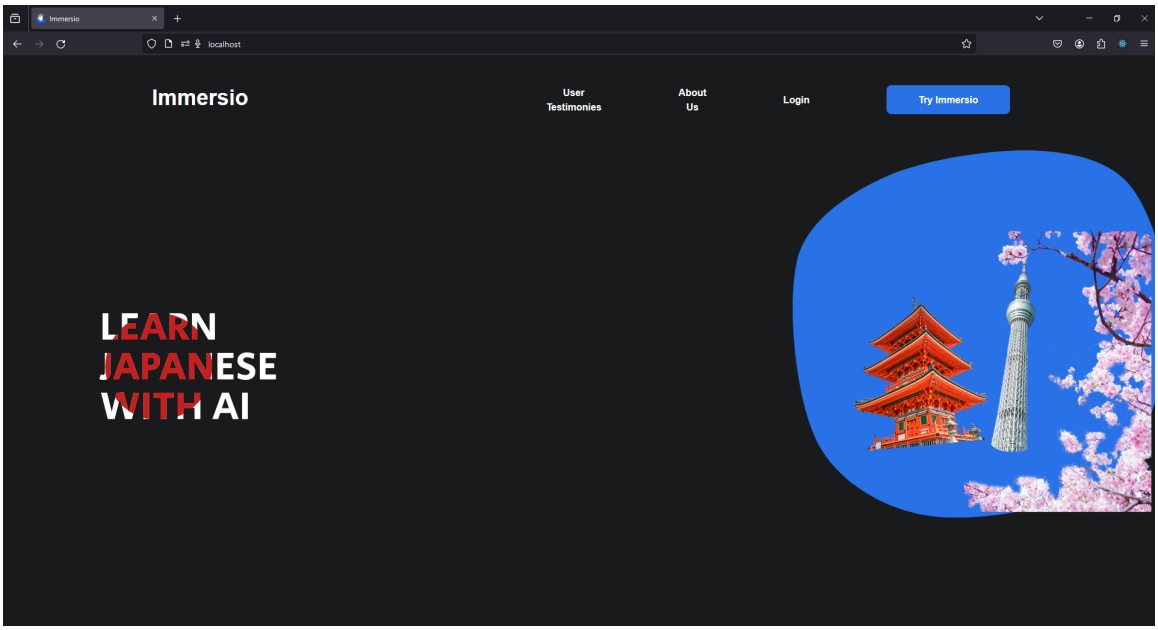

Figure 5.2: The landing page of Immersio

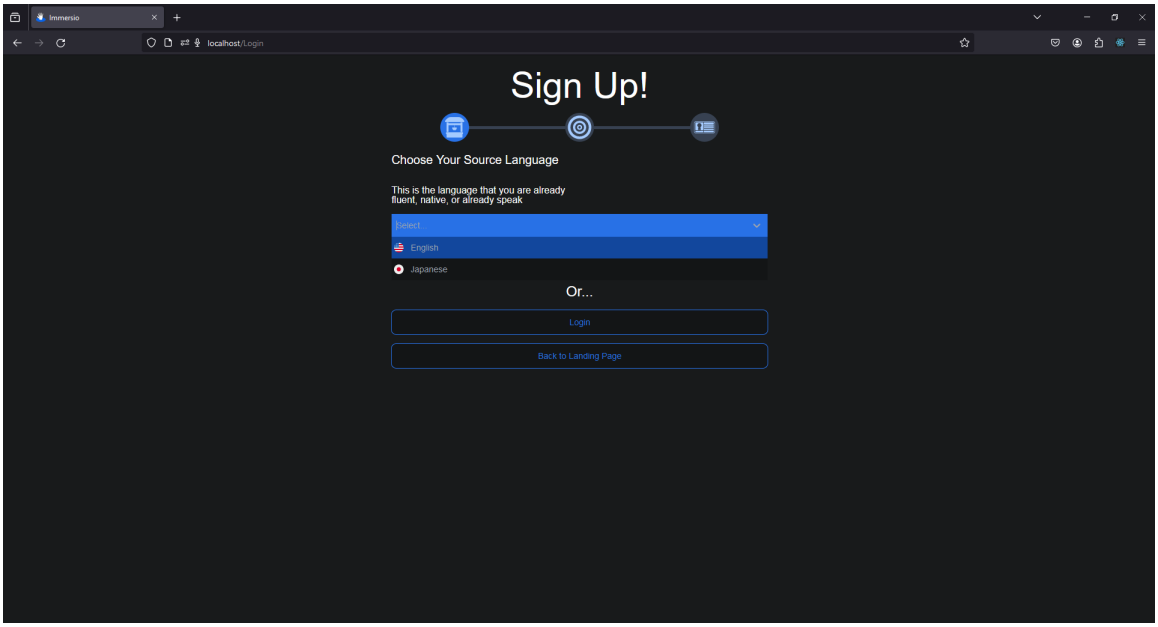

Figure 5.3: During registration of Immersio, the user is prompted to select a source language

After the user has selected both source and target language, the user is prompted to enter a username, email, password, and to retype their password as seen in Figure 5.5. If a user has previously registered for Immersio, the user simply has to login as seen in Figure 5.6.

Upon logging into Immersio, the user is greeted with a dashboard page that is intended to provide a high-level overview of the application. The dashboard consists of a heat map of a user's review's done per day as well as the recent conversations that a user has engaged in with the conversational agent. Additionally, the user can quickly go to their reviews or start a chat conversation by clicking on the link blocks. Link blocks contain a title and a short description of what the user can expect when pressing the link. Additionally, a user's review goal is displayed, along with its associated progress bar, which is defaulted at fifty review items per day (i.e., a user must review fifty review items every day). The dashboard can be

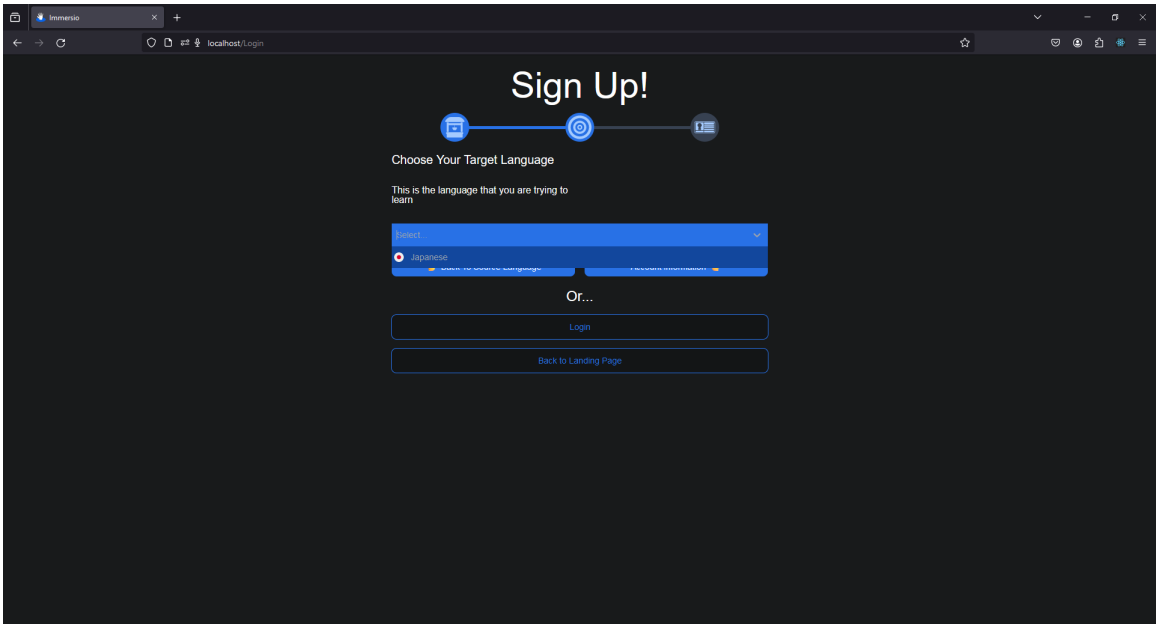

Figure 5.4: After selecting a source language a user selects a target language – Japanese in Immersio's case

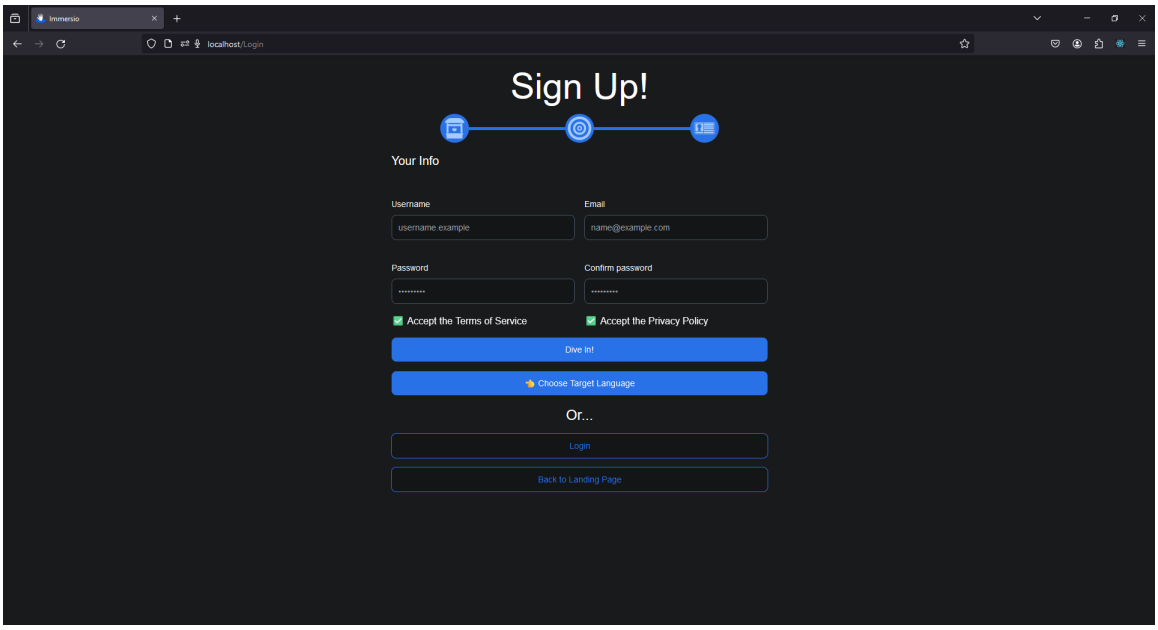

Figure 5.5: During the final stages of registration, a user is prompted to enter username, email, password, and to retype their password

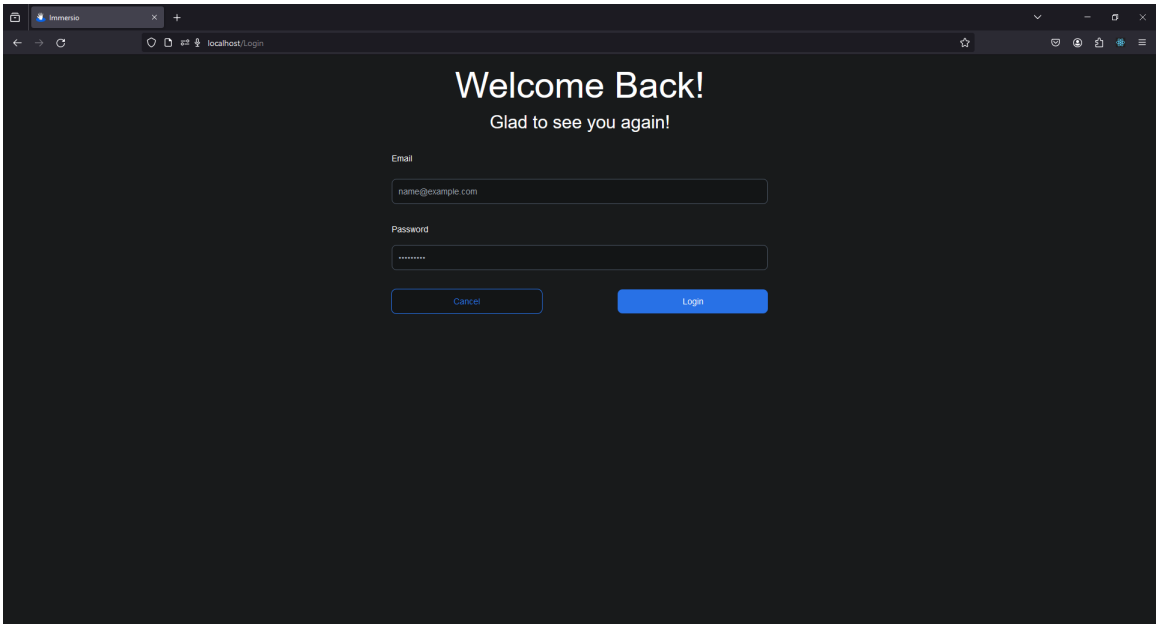

Figure 5.6: A user can login if they have previously registered before

seen in Figure 5.7 which contains the heat map, the recent conversations, and the link blocks.

The user can click on the chat link block to go directly to the chat page where conversation creation is possible. The user can also navigate there using the side navigation panel on the left hand side of the screen. A user can create a new conversation by typing in an input box an pressing the "enter" key on the keyboard. The user can alternatively click on the send button to the right of the input box. If the user types in a language other than their target language, Japanese, the user will receive an error message back from the server in the form of a toast message. The error message automatically disappears after a few seconds but can manually be closed via its close button. A summary of these features can be seen in Figure 5.8.

If the user sends a message in their target language, in this case, Japanese,

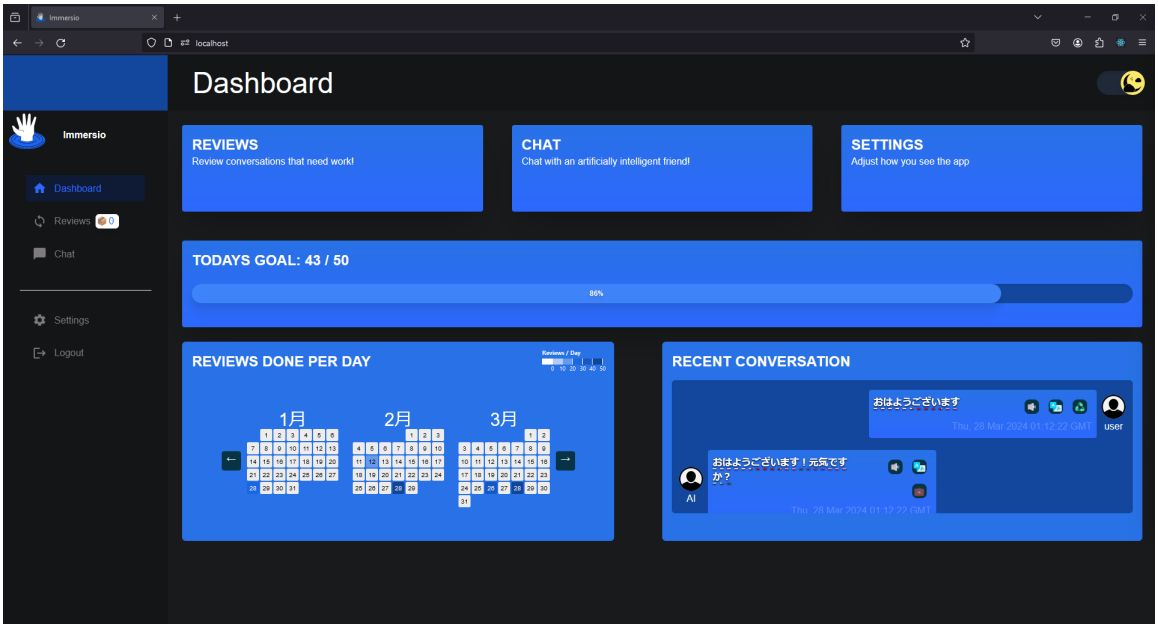

Figure 5.7: The dashboard page complete with a reviews heat map, a recent conversation block, and link blocks to the reviews and chat page

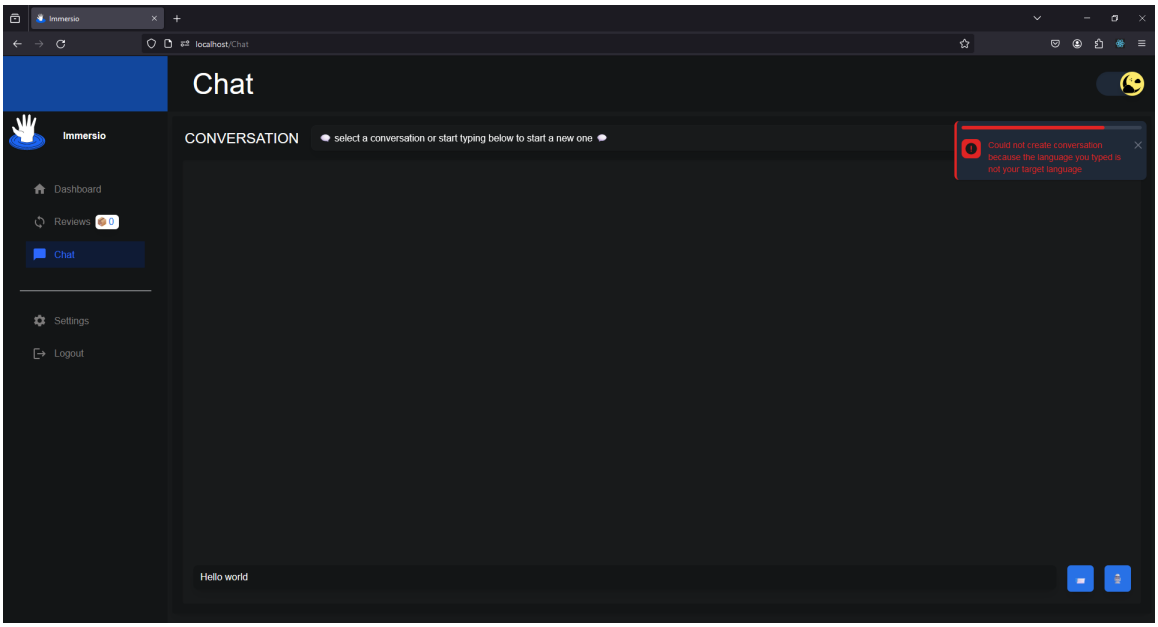

Figure 5.8: The chat page where a user is attempting to speak in English when the langauge expected was in Japanese

the server will respond with a response from the conversational agent produced by LangChain[1] and GPT-3.5-Turbo[2]. During this time, RabbitMQ also receives the AI's response as well as the user's response in which two additional actions are performed. First, RabbitMQ pipelines the response to a service in a Docker container which handles POS parsing which is where spaCy resides. Second, RabbitMQ sends the data to another Docker container which contains the Hugging Face translation model, `opus-mt-ja-en`[3], which translates the message from Japanese to English. `opus-mt-ja-en` also sends its translated output to Grammarly's Hugging Face model, `coedit-large`[4], in its own Docker container. Both the POS and translation data is sent back to Django over RabbitMQ and stored in a Message table in a PostgreSQL database. Because the POS and translation data

[1] https://www.langchain.com/
[2] https://platform.openai.com/docs/models
[3] https://huggingface.co/Helsinki-NLP/opus-mt-ja-en
[4] https://huggingface.co/grammarly/coedit-large

is generated asynchronously, that is without slowing the response time from the server to back to the client, the user does not have to wait a long time to view the generated messages. This allows the messages to be rendered nearly immediately after the data is given to the client. After a few seconds on the frontend, the POS and translation data are automatically requested via a REST API request in a React `useEffect` hook to retrieve the data. The POS items are highlighted and color-coded according to POS type and the translation button shows the English translation on hover and continuously shows the English translation below the message if the button is pressed. Messages also include time and date creation stamps. A visual representation of this description can be seen in Figure 5.9.

A user may also use their microphone to record audio so that it may be sent to the backend where it is transcribed to Japanese text using WhisperTTS[5]. This

[5]https://github.com/openai/whisper

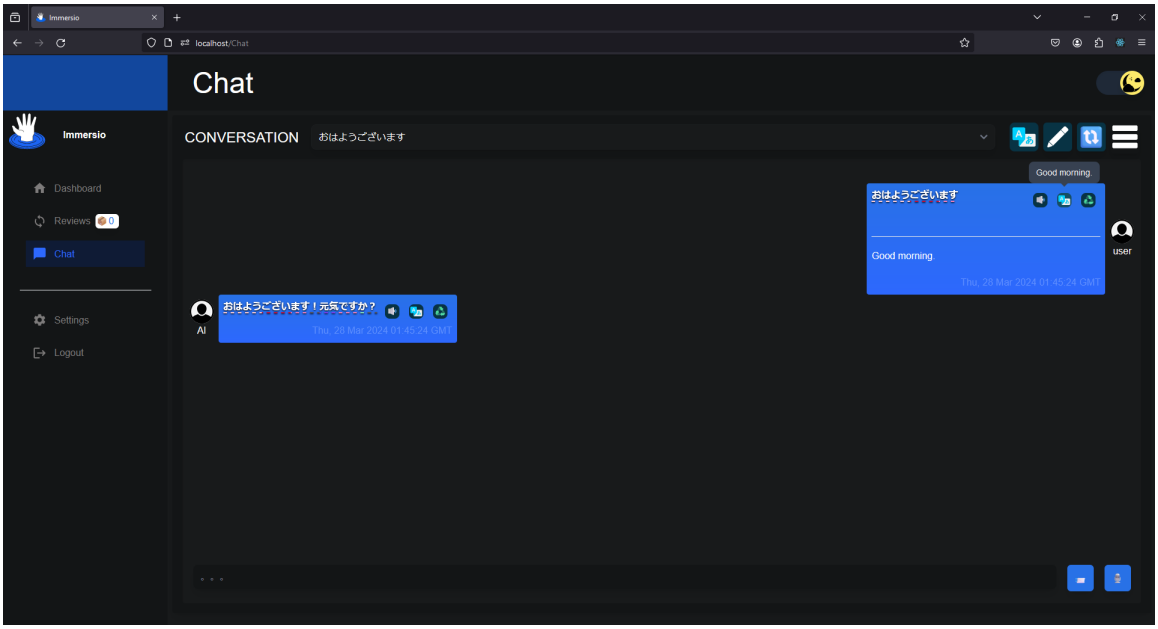

Figure 5.9: A Japanese conversation between a user and a conversational agent where POS items are highlighted and the translation button is being used

occurs whenever a user presses the record audio button on the bottom right of the chat page and chooses to allow microphone access to the browser. At this point, the audio overlay appears and allows recording audio by pressing the microphone button and can be stopped by pressing the stop button where the stop button can be seen in Figure 5.10. Afterwords the user can choose to send the audio message, to the backend so Japanese text gets transcribed, by pressing the send audio button as seen in Figure 5.11.

Upon hovering over a POS item in a message the POS item's text size increases slightly as seen in Figure 5.12. If the user clicks on the POS item, a modal will appear. A modal is a term for a popup that allows the user to see that they are still on the same page while allowing the user to perform other actions. The modal displays POS information using a dictionary stored in a PostgreSQL database. The dictionary in question is the JMDict database, which is an English-to-Japanese

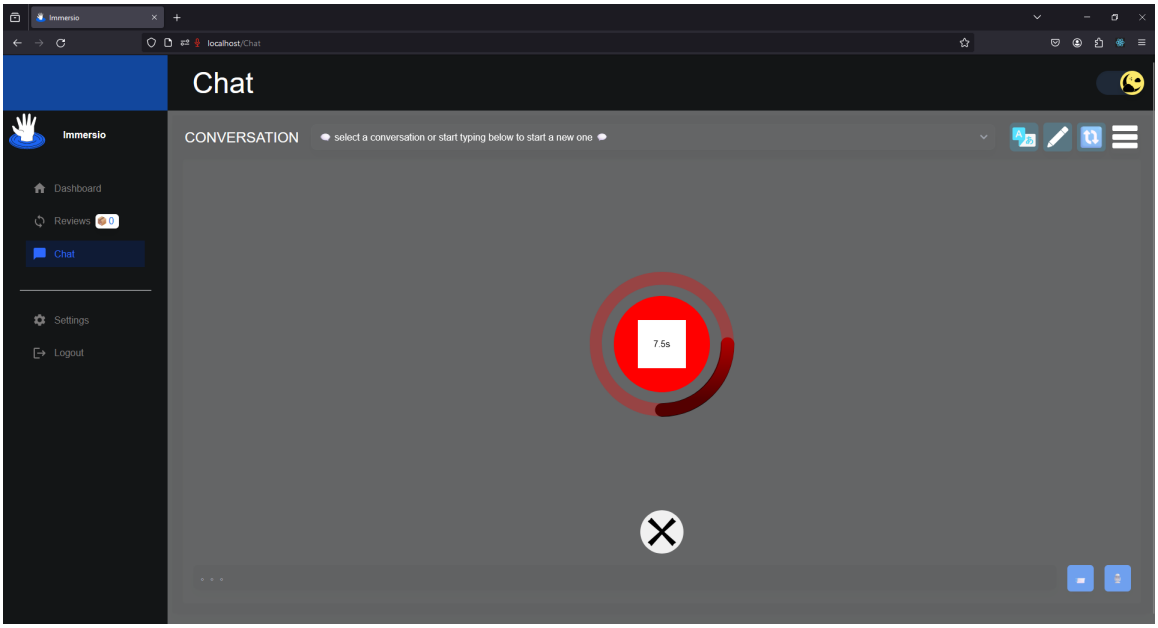

Figure 5.10: The stop button on the audio overlay that a user will see while recording audio

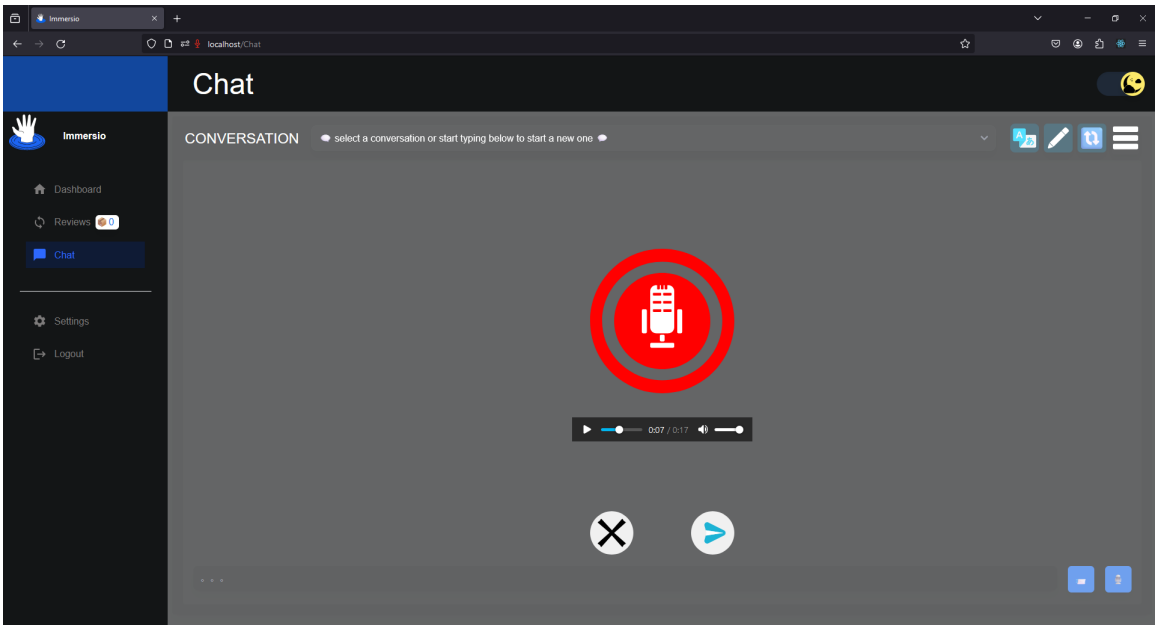

Figure 5.11: The audio overlay that shows the record button as well as a cancel and send audio button

dictionary. The modal displays JMDict's definitions depending on the type of word. The user can also add the word to their reviews by pressing the review button. An illustrated example can be seen in Figure 5.13. Once the user presses the review button, the review button is replaced with a manage review button item as seen in Figure 5.14. Similar to when a user presses the review item for a word, a user may also review a complete sentence by pressing the review sentence button on a message bubble. By doing this, the review sentence button updates its appearance as seen in Figure 5.15.

In order to review added review items the user may navigate to the reviews page by clicking on "Reviews" on the navigation panel. The user should see three UI elements that are shaped like blocks which are links. This interface is called the reviews landing page. Assuming that the user has already added review items previously, the total amount of due review items, counting all words and sentences,

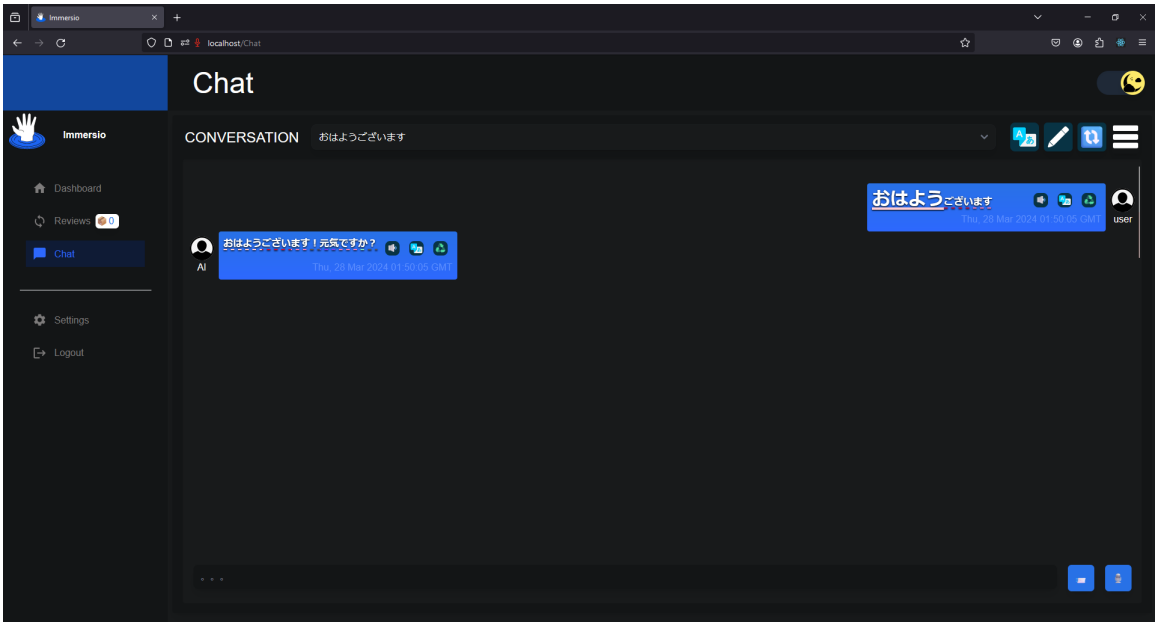

Figure 5.12: A POS item englargening as a user is hovering over it

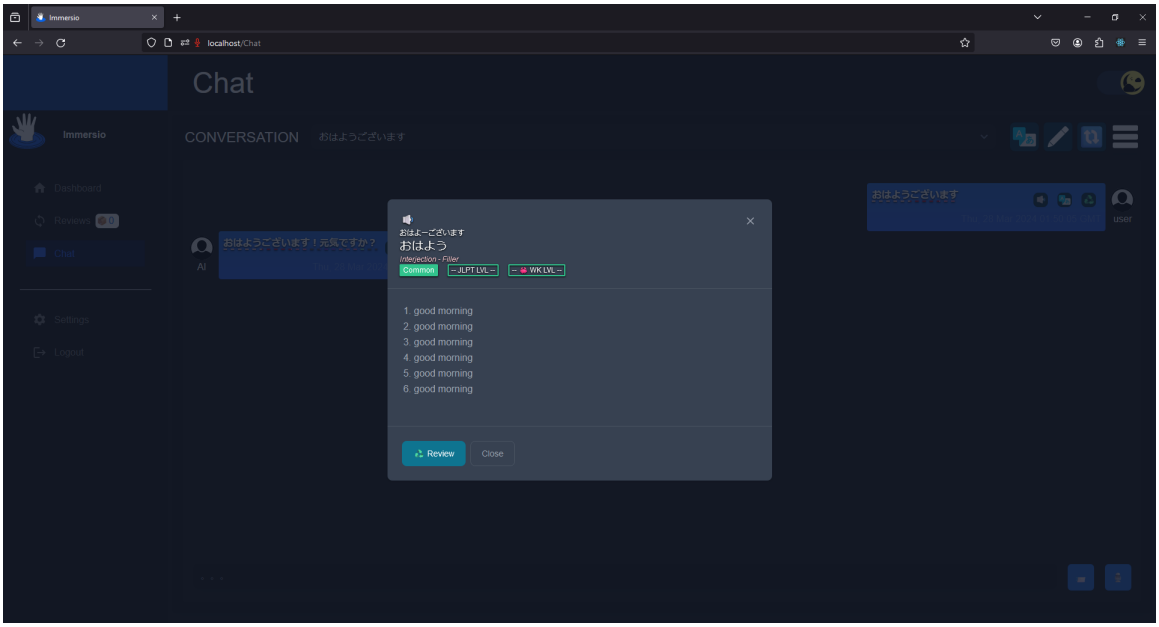

Figure 5.13: The POS modal that displays the definitions of a POS item as well as the POS type and a way to add the POS item to the reviews

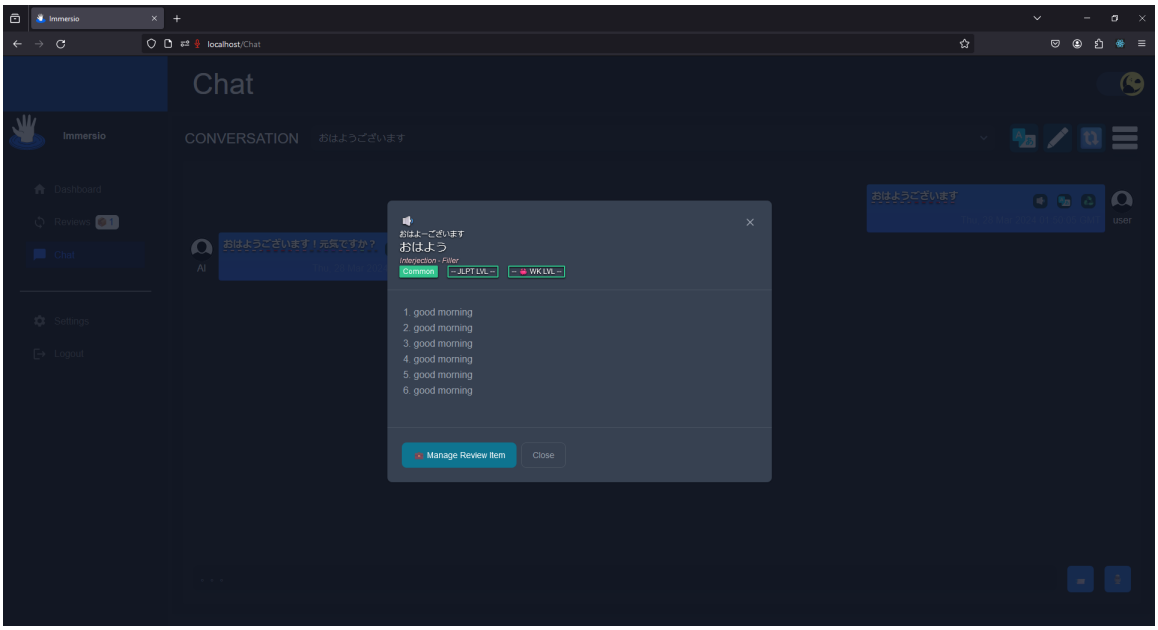

Figure 5.14: After the Review button has been pressed

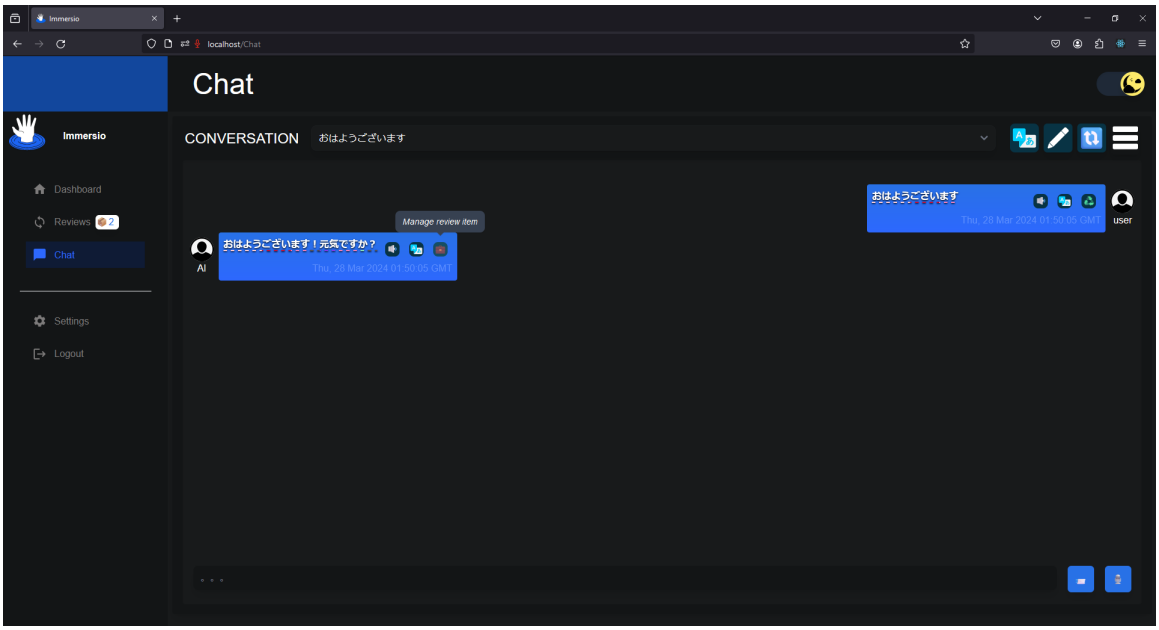

Figure 5.15: After pressing on the review sentence button on a message bubble

is displayed on this page under the ”REVIEW ALL” link block. Additionally, the amount of due review items can also be seen on the left-hand side on the navigation panel. If the user presses the ”REVIEW ALL” link block, a confirmation window shows up listing all the review items a user will review during a review session. Similarly, if the user presses the ”REVIEW SENTENCES” or ”REVIEW WORDS” link block a confirmation window will show up but only for review items of those review types. A diagram for this description can be seen in Figure 5.16.

Regardless of the type of link blocks a user had pressed on the reviews landing page, the user is greeted with a review confirmation page which provides detailed information of the review items that a user chooses to review. The review confirmation page organizes review items in a tabular format with information of each review item including the content, the translation, the SRS level, the SRS name, the

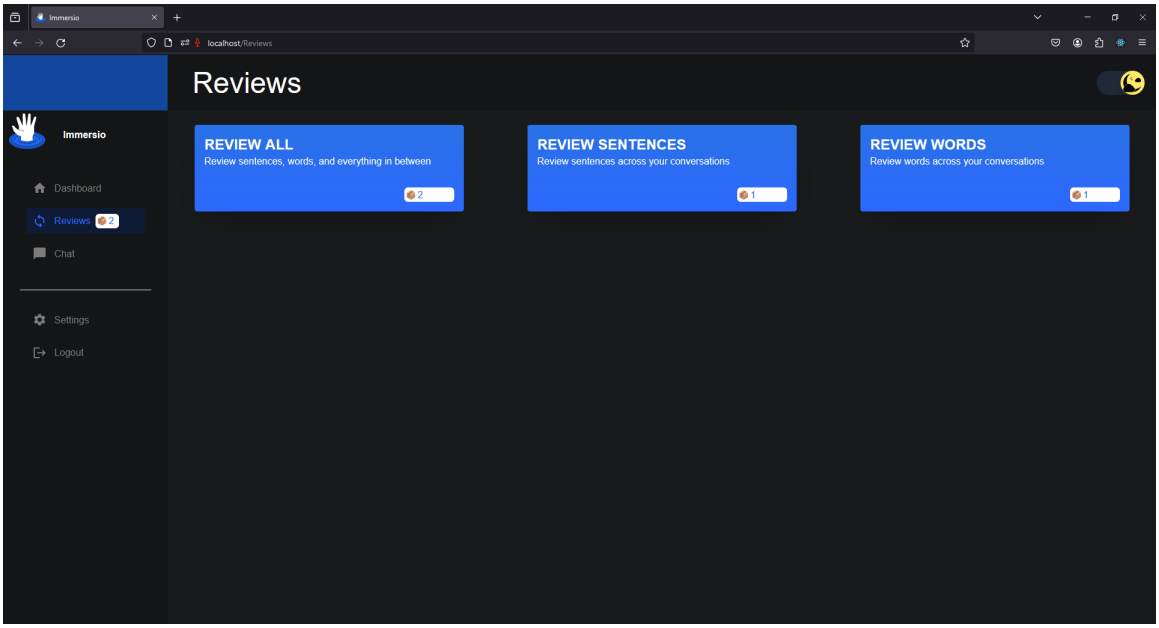

Figure 5.16: The reviews landing page with various options to review information via link blocks after the user navigates to the reviews

review frequency, the last review date, and the next review date. All due review items by default will say ”Now” for the next review date whereas the last review date depends on when the item reviewed during a review session. The SRS level and SRS name are Immersio’s implementation of Leitner system boxes. The SRS level determines the SRS name and review frequency. The SRS level, SRS name, and review frequency are color-coded and will change color depending on the SRS level. The user also has the ability to edit a review item by pressing the edit button denoted by the pencil icon. These details can be seen in Figure 5.17.

If the user clicks on the edit button of a review item, a modal is displayed in which the user can choose to edit a review item. Within this modal, a user has the ability to delete a review item by pressing the delete button which can be seen in Figure 5.18. Once this button is pressed, another modal, which acts as a confirmation window, displays. This modal is responsible for confirming with a

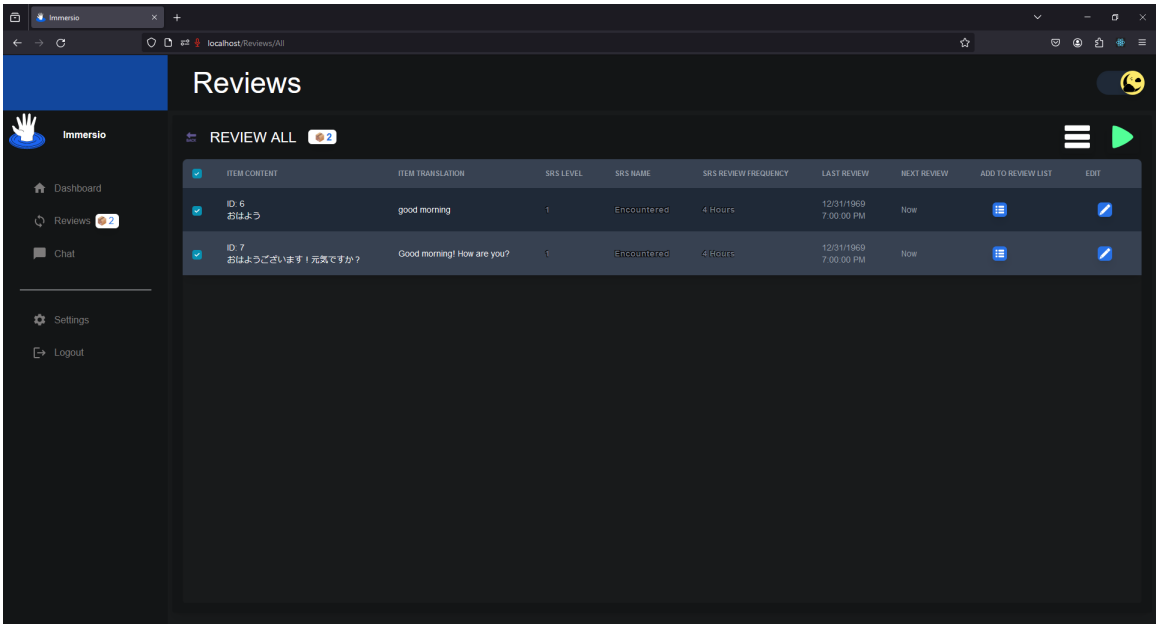

Figure 5.17: The review confirmation page where a user can see details of the review items they will review

user if they would like to delete a review item. The modal informs the user that deleting the review item is irreversible and cannot be undone if they proceed. This confirmation modal can be seen in Figure 5.19. If a user does delete a review item, the due reviews count is updated accordingly as seen in Figure 5.20.

By default, all review items will be reviewed when the user presses the start review session button on the top right. However, the user can choose to select or unselect certain review items by clicking on the review item's check box or by clicking on the review item in the table. If a review item's checkbox is unchecked it will not show up during the review session. Otherwise, it will be added to the review session. The user can toggle all review items at once by clicking on the checkbox at the top left of the reviews confirmation page. If the user does not have any review items selected, the start review session button cannot be pressed as no button will be displayed. Similarly, when there are no review items currently due

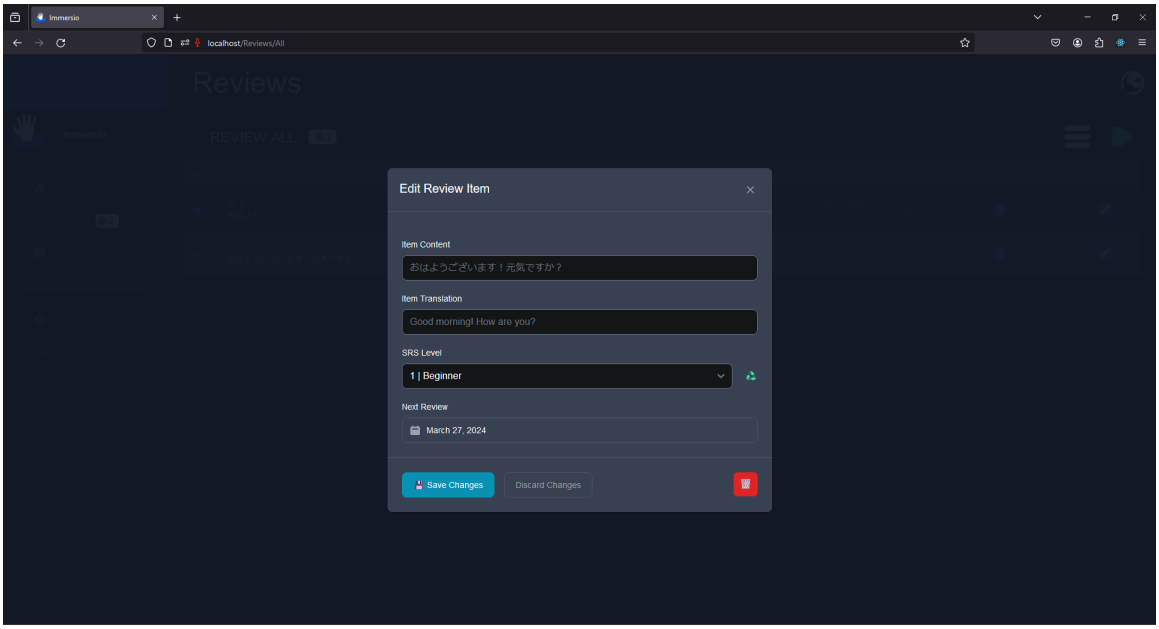

Figure 5.18: The edit review item modal which is necessary to open in order to delete a review item

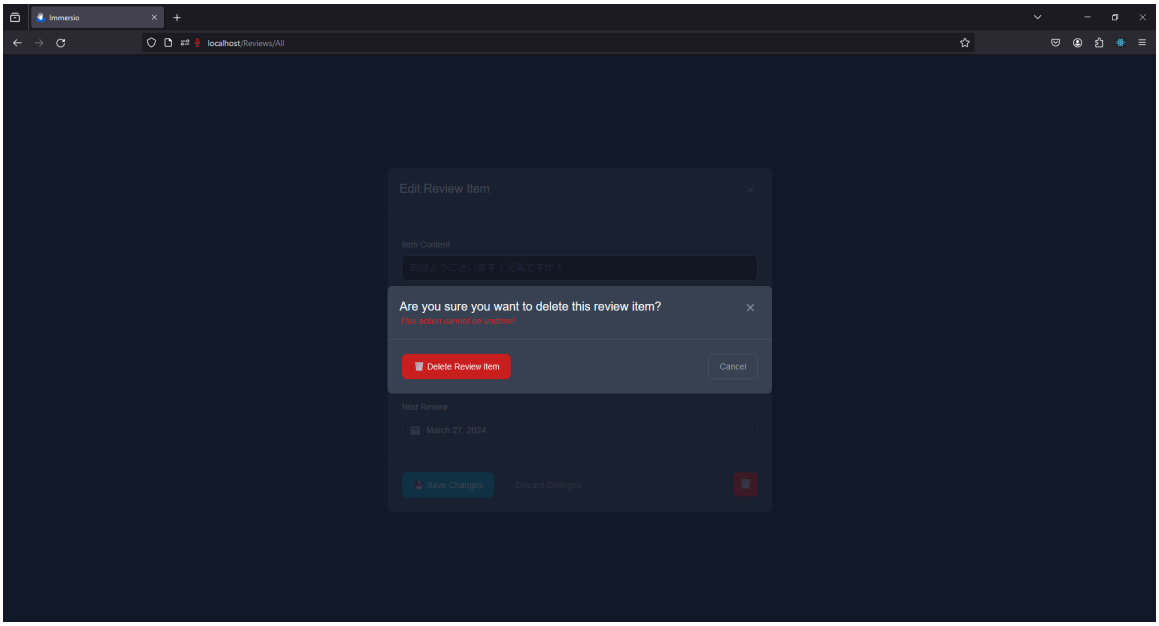

Figure 5.19: The delete review item confirmation modal which informs a users that deleting a review item is irreversible

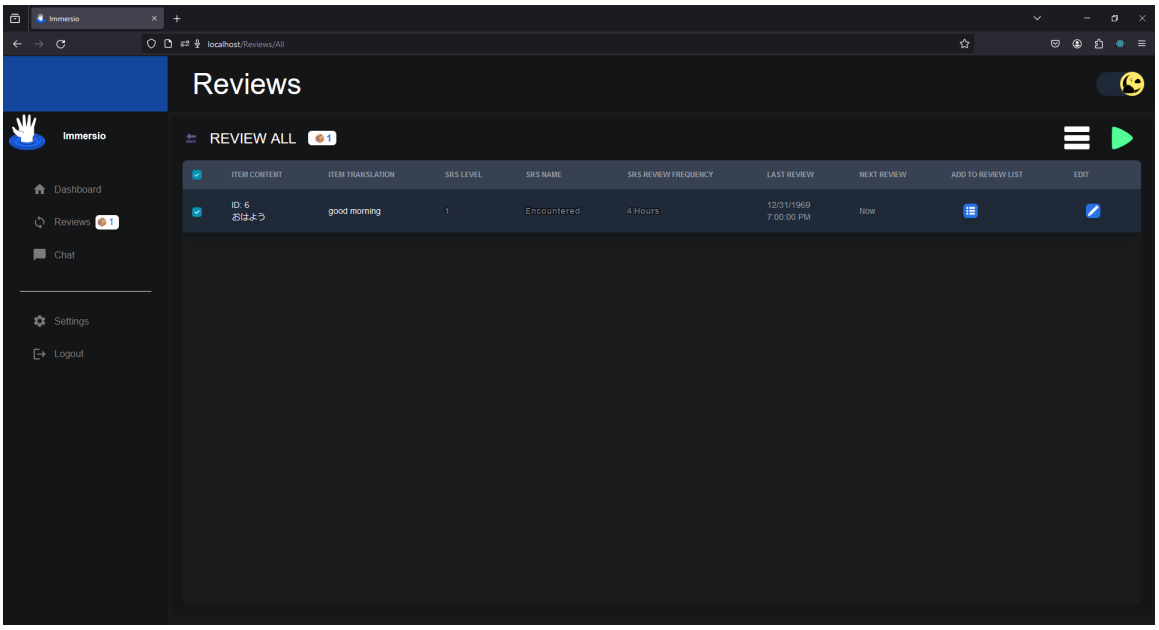

Figure 5.20: Deleting a review item results in decrementing the review item count and the review item will no longer be available for review

the start review session button will not display. This can be seen in Figure 5.21.

Once a review session has been created by pressing the start review session button in the top right corner of the review confirmation page, a review queue is generated that contains all selected due review items. A review session consists of three states: the translation state, the conversation context state, and the review report state. During the translation state the user is prompted to translate a word or sentence. Words are highlighted to denote what word to translate. Whenever the user presses the check answer button or presses the enter key on the keyboard, the backend evaluates whether or not the user's translation was correct. A loading animation will be displayed and input is disabled to prevent multiple REST API requests from being sent. An illustrated example can be seen in Figure 5.22.

The backend evaluates a user's answer by generating a translation of the original message itself and comparing the similarity of the generated translation

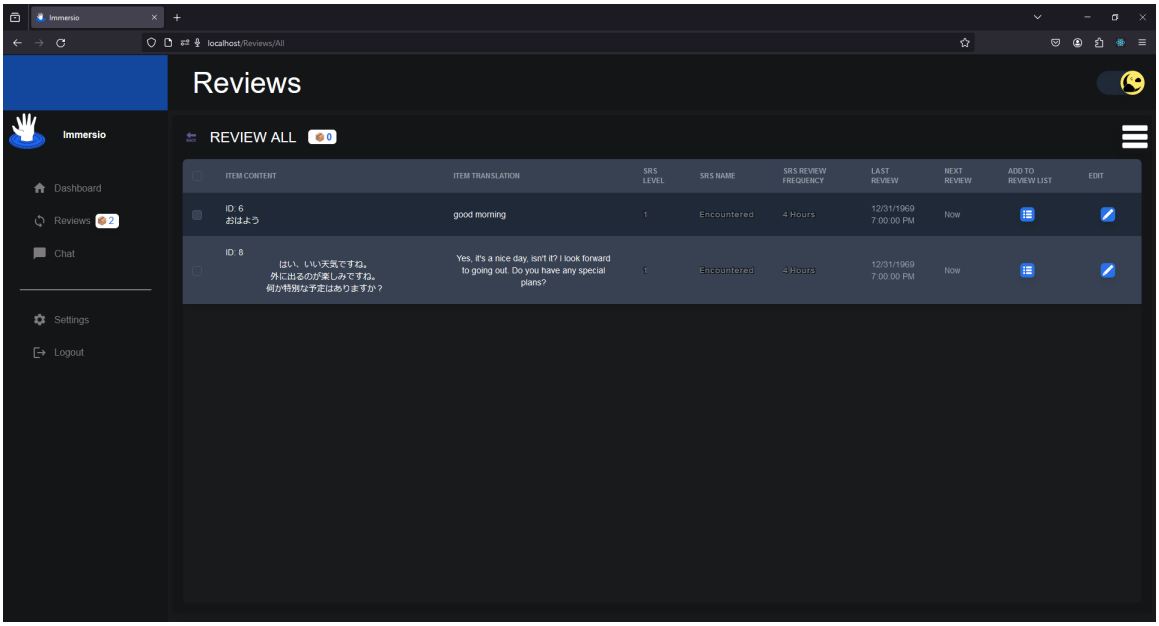

Figure 5.21: A review confirmation page that does not display the start review session button on the top right due to not containing selected review items

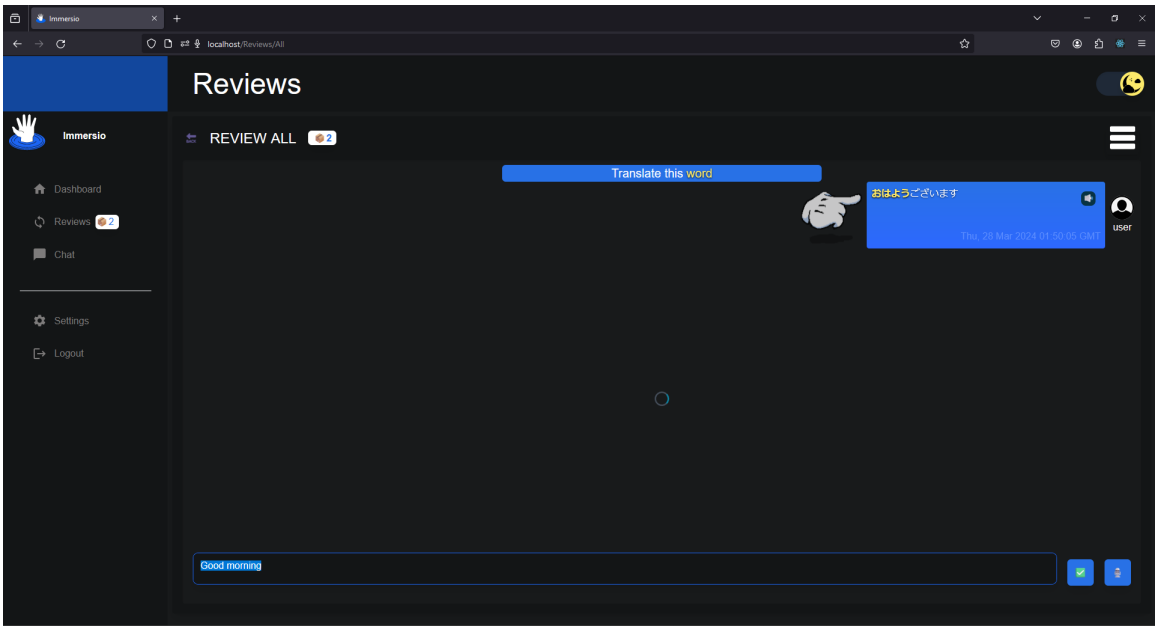

Figure 5.22: A word that a user is prompted to translate along with the loading animation

against a user's translation. If the similarity rating is above a certain threshold, the user's translation is marked as correct otherwise it is marked as incorrect. The information that gets sent back to the user includes a boolean value of whether the answer was correct or not along with the generated translation. Once this data is received by the client, the conversation context state starts and displays whether or not the answer is correct. The user also receives the conversation context that the word or sentence was in by performing another REST API request to the backend. During the conversation context state, a user has the ability to undo a submitted translation, the ability to toggle the corrected translation, and the ability to continue to the next sentence. This can be seen in Figure 5.23.

If the user presses the continue button a REST API PATCH request is made. This does not happen if a user presses the undo button. If a user presses the undo button on the bottom right during the conversation context state, the review

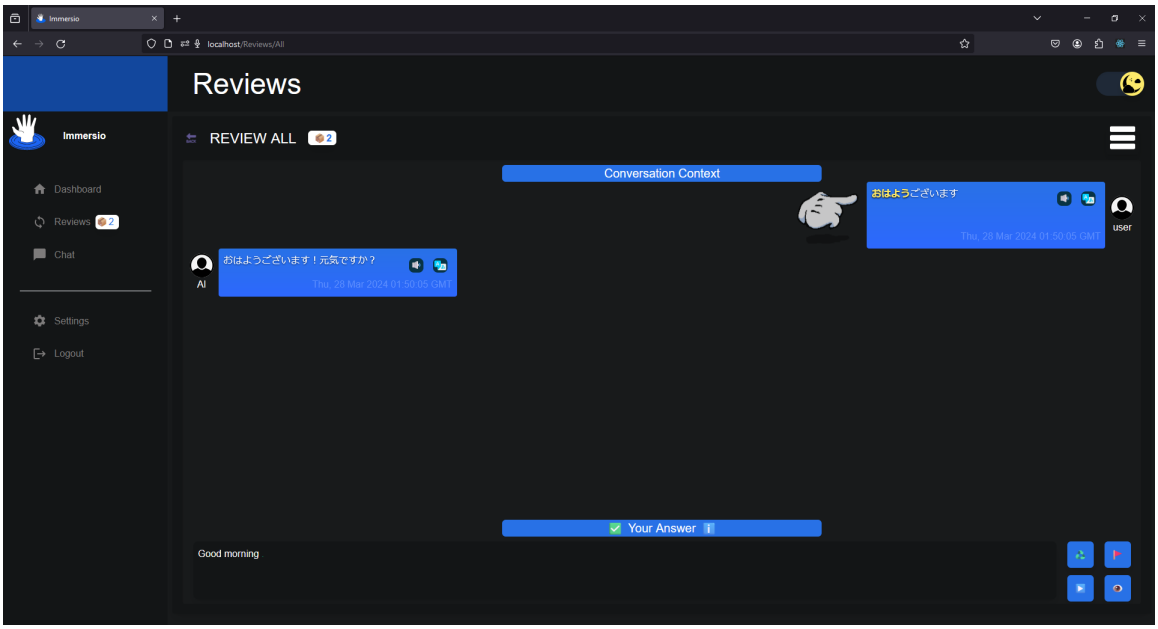

Figure 5.23: The conversation context that is displayed once the backend finishes evaluating a user's translation

item's SRS information is not modified. When the undo button is pressed, the user returns to the translation state to attempt to re-translate the word or sentence. The effects of clicking the undo button can be seen in Figure 5.24.

If the user translates a word or sentence and the backend evaluates that it is incorrect, the user's translation will be annotated with the correct answer. This is done by making words of the user's translated text, that are not in the generated translation, red and contain a strike-through. Words that are in the generated translation but not in the user's translation are green. Finally, words that are both in the user's translation and in the generated translation are white. This can be seen in Figure 5.25. If the user would like to see their original translation they may press the hide answer button to show show their original translation. This button can be toggled on and off to show the user's original translation along with the annotations. The result of pressing the hide answer button can be seen in Figure

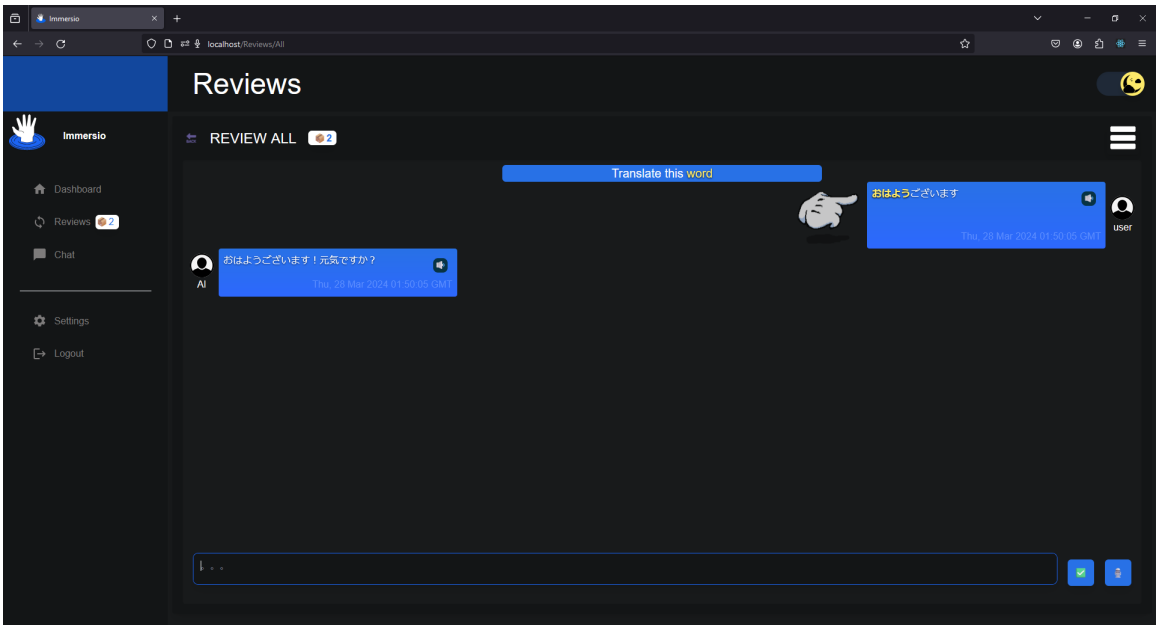

Figure 5.24: After pressing the undo button during the conversation context state the state changes to the translation state

5.26.

In the event that a user has a sentence review item in their review queue, during the translation state an arrow points to the message to be translated. Unlike a word review item that is highlighted during the translation state, no part of the sentence is highlighted. Both word and sentence review items will iterate between the translation state and conversation context state until the review items in the review queue are empty. If a review item is marked as incorrect it gets reshuffled in the review queue to be encountered during the same review session. A sentence review item can be seen in Figure 5.27.

Even if the generated translation and the user's translation are not completely the same, the user can still get a translation correct. The idea is that the user gets the idea of what the translation means but does not have to provide a word-for-word or literal translation to the item. In cases such as these, the review item can

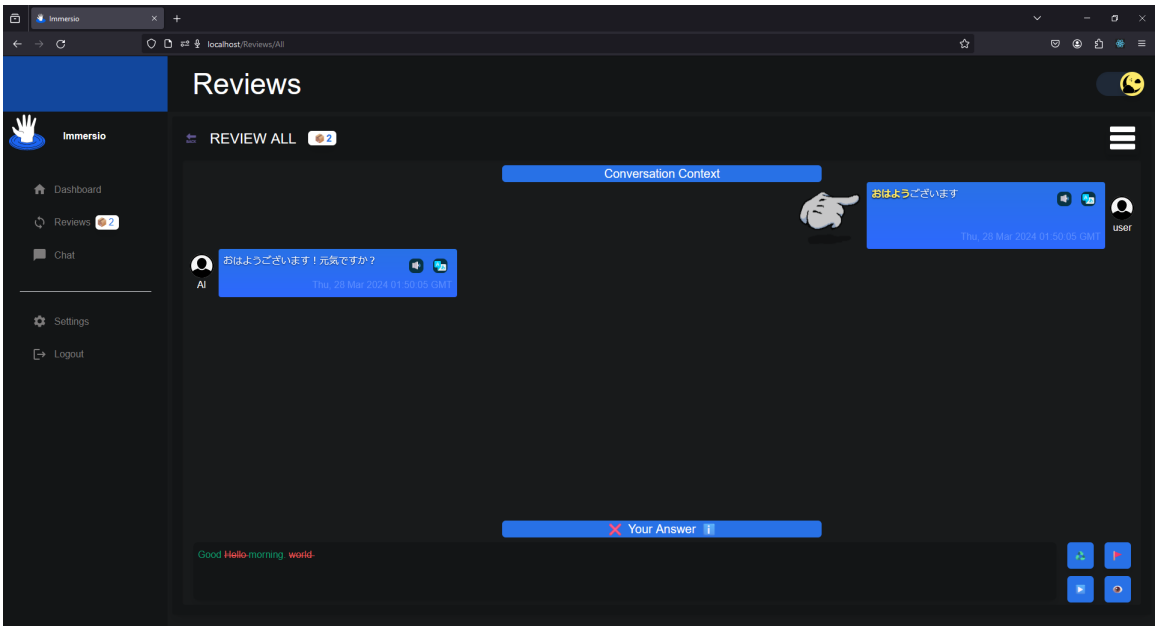

Figure 5.25: A user's translation that was deemed incorrect and annotated according to the generated translation

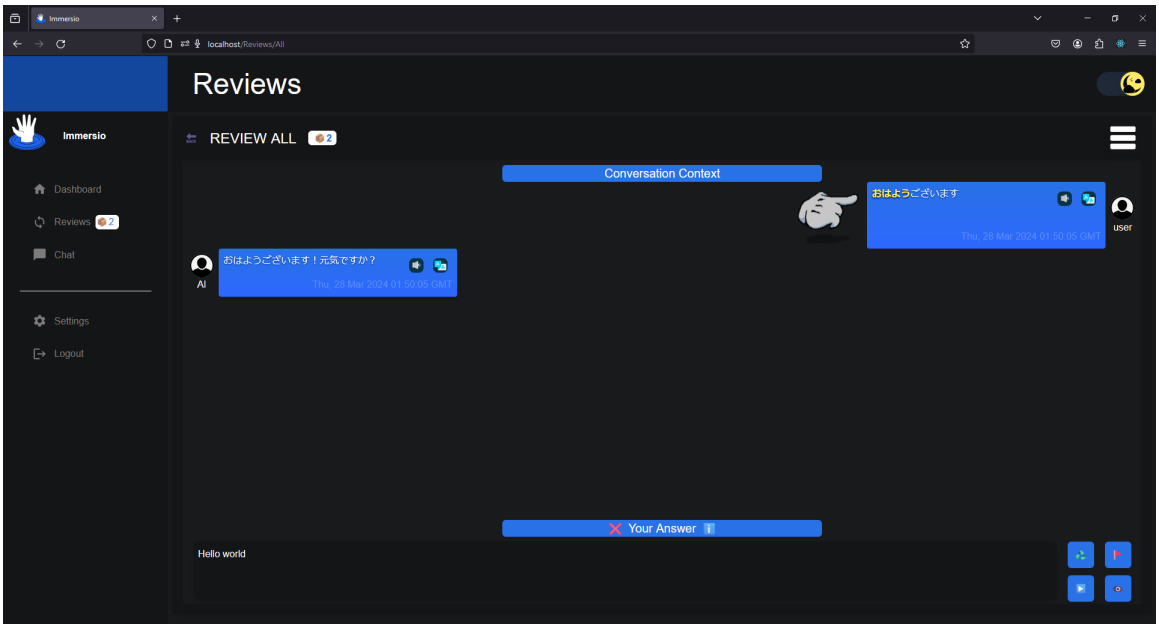

Figure 5.26: The result of pressing the hide answer button

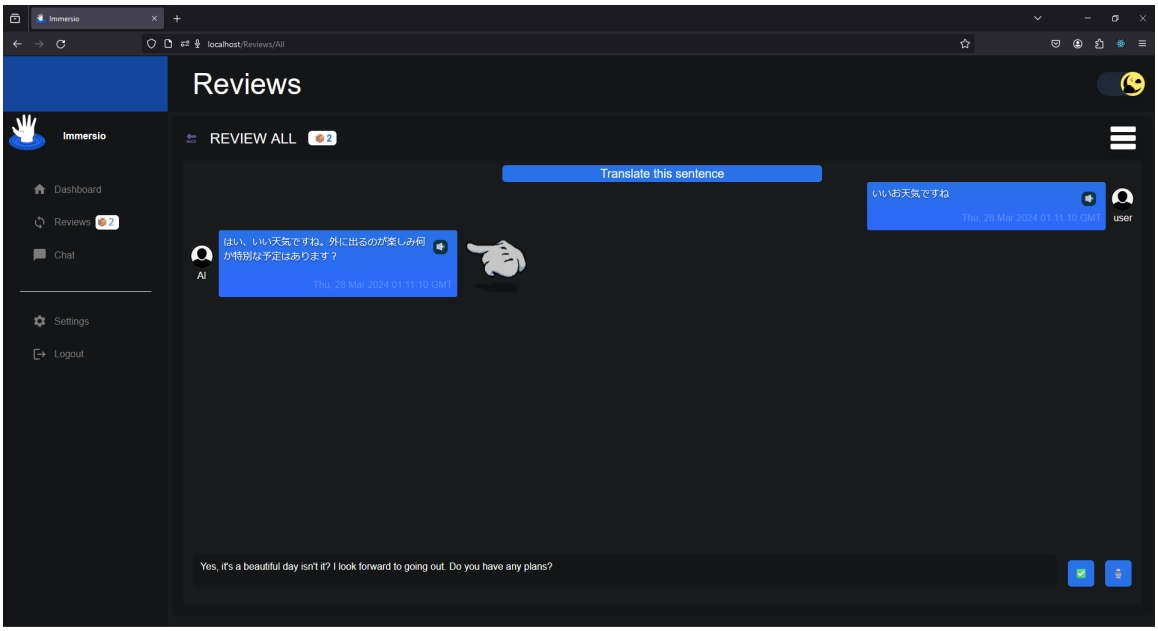

Figure 5.27: A sentence review item as seen during a review session

be marked as correct but have some words that are annotated to show that it was not the literal translation. This can be seen in Figure 5.28.

In some rare cases, a user may wish to assert that they were correct or incorrect regardless of the evaluation from the backend or when pressing the undo button does not seem reasonable or is time consuming. For this reason, a user may also press on the exclamation icon near the "Your Answer" title to assert whether or not the user was correct or incorrect. If the evaluation from the backend claims that a user was correct a user may assert that they were incorrect as seen in Figure 5.29. Similarly, if a user was incorrect, he or she may assert that they were correct as seen in Figure 5.30.

The review queue becomes empty when a user answers all review items in the review queue correct at least once. Once the review queue is empty, a user is transitioned to the review report state in which a report of the review session will be generated. The review report consists of an accordion to display user messages

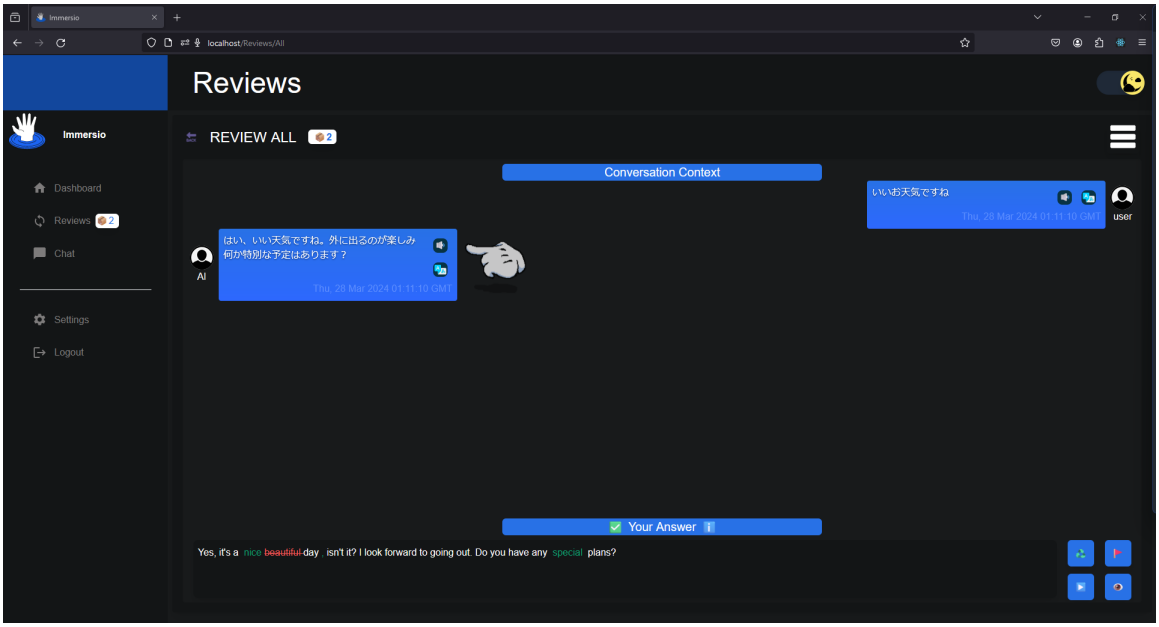

Figure 5.28: A user translation marked as correct even though not every word was the same as the generated translation

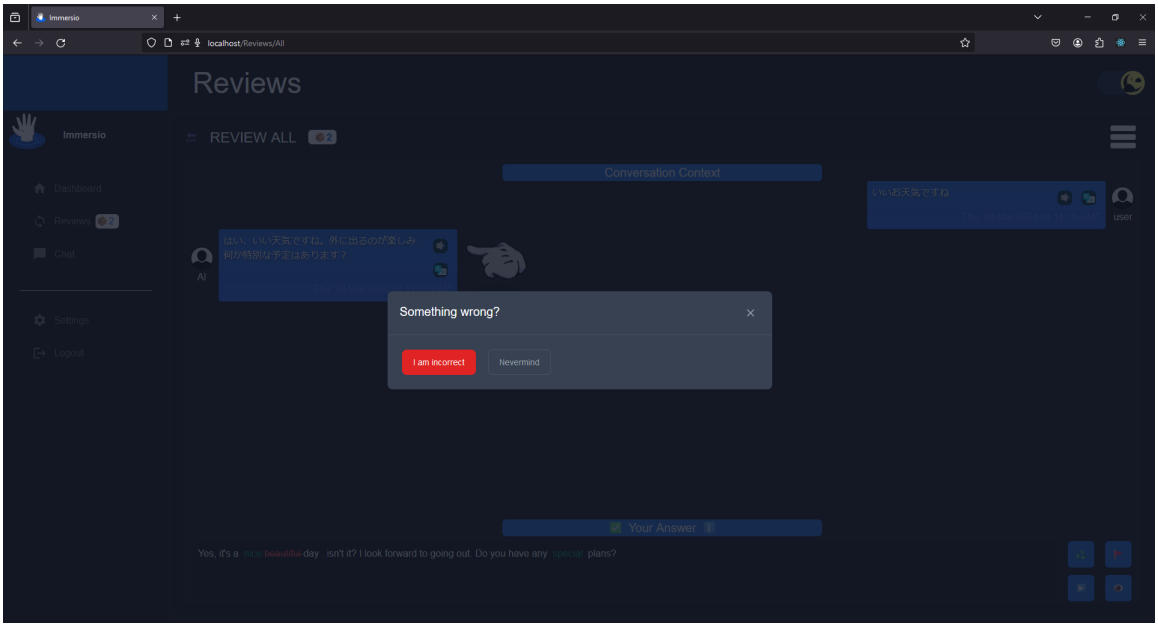

Figure 5.29: The ability to assert that the user was incorrect

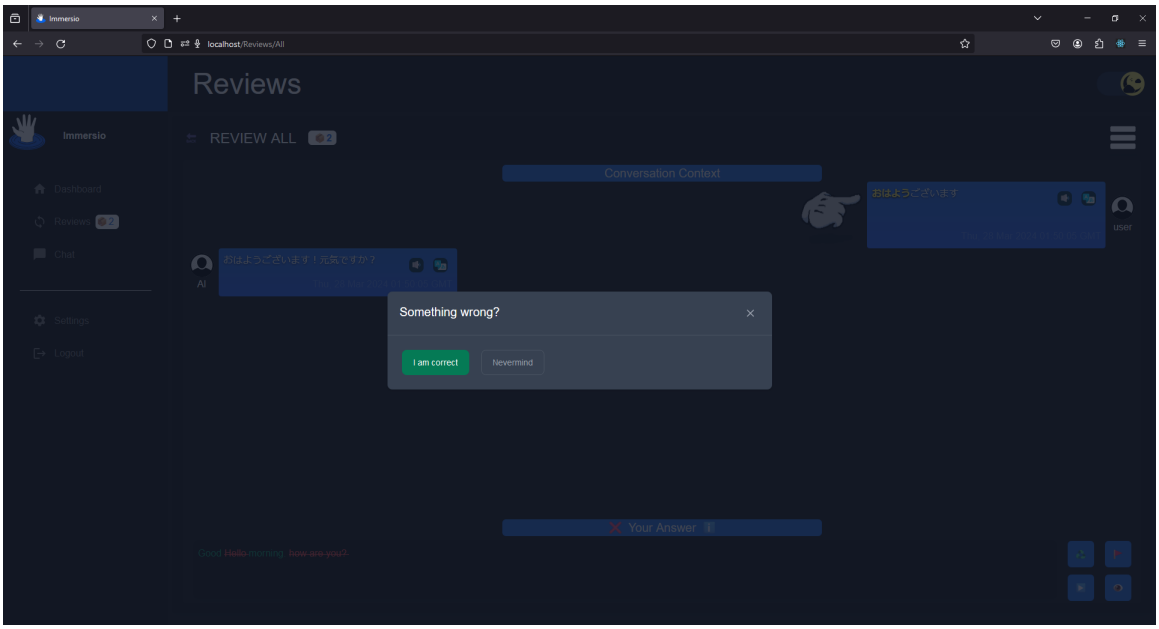

Figure 5.30: The ability to assert that the user was correct

that appeared in the review session. An accordion is a type of UI component that can show a drop down of information when a header is pressed. An accordion can have several drop downs with its associated header. The review report's accordion along with an accuracy of the number of correct review items over the total review items can be displayed in Figure 5.31. What is deemed a correct review item according to the translation accuracy is whether or not a review item has been ever been marked as incorrect. If a user gets a review item correct on the first try of translating a review item, or asserts that they were correct the first time they encounter the review item, the answer will be marked as correct in the review report. Figure 5.32 shows a review report with both correct and incorrect review items. The user may finally choose navigate to the dashboard by clicking on the home button provided on the review report state.

During the translation state, a user must translate a review item in their own source language. If the user does not translate in their source language, say English,

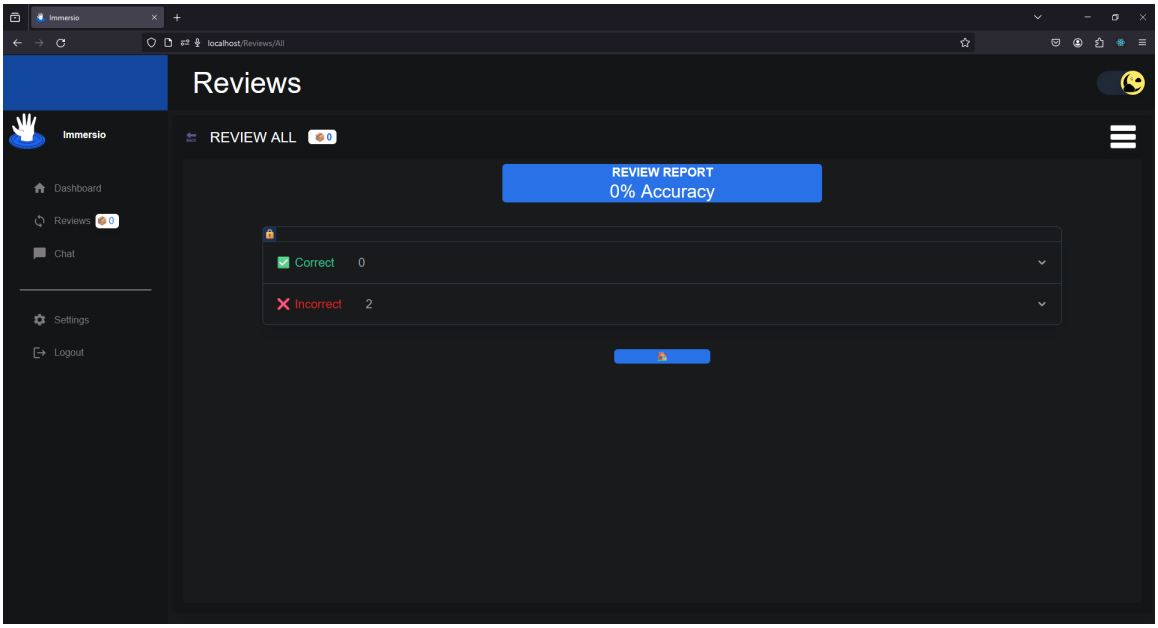

Figure 5.31: The review report state that shows an accordion of correct and incorrect items as well as the translation accuracy

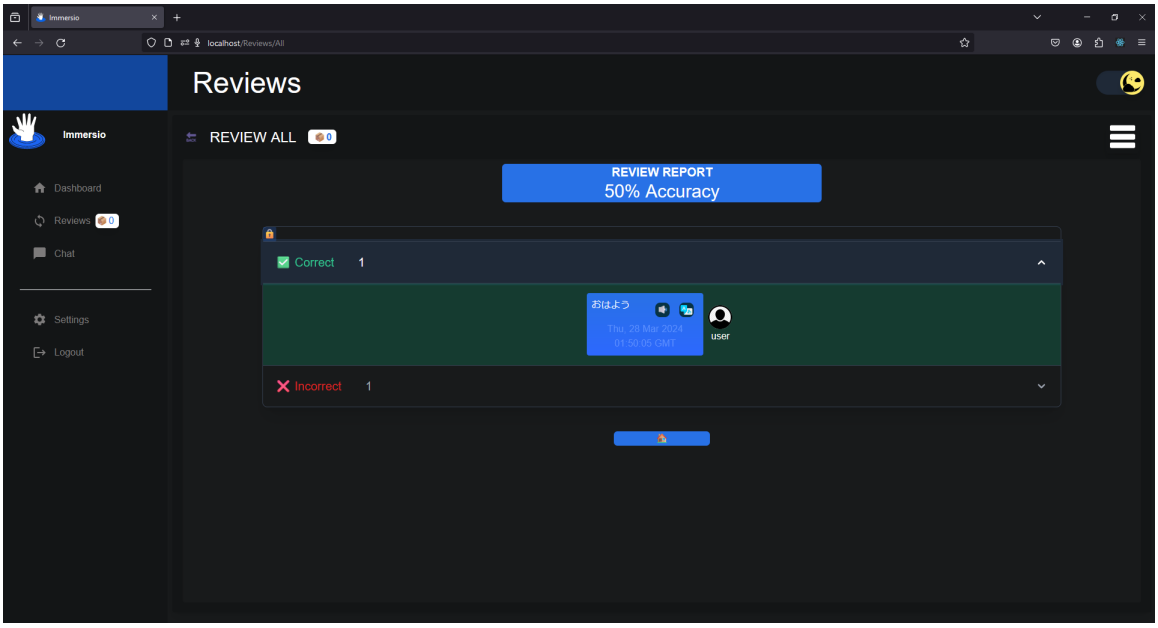

Figure 5.32: The review report page which displays both a correct and incorrect review item with a 50% accuracy

he or she will be prompted with a toast message stating that they must translate in the source language. This is done to prevent a user from accidentally retyping the sentence or word they are to translate and accidentally getting the word incorrect. This is similar to the message that is displayed when a user attempts to converse with a chat agent on the chat page in a language that is not their target language. This toast message can be seen in Figure 5.33.

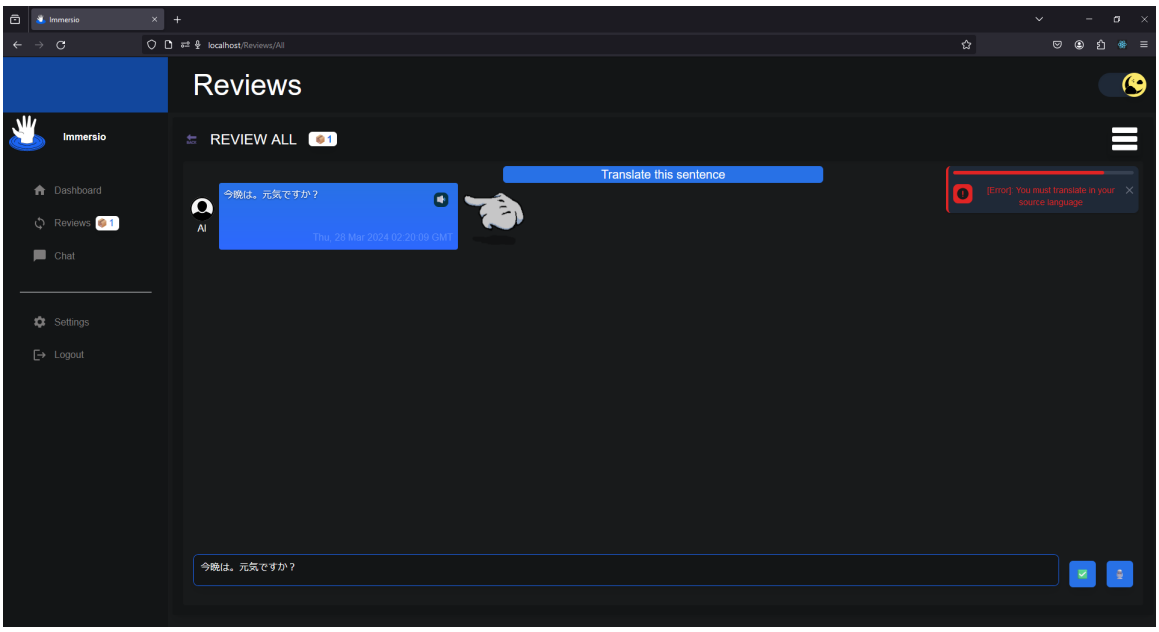

Figure 5.33: The toast message that is displayed when a user attempts to converse with a chat agent on the chat page in a language that is not their target language

5.2 Testing Results

The Japanese efficacy of Immersio's AI models was evaluated by having a native Japanese speaker evaluate the tool. The Japanese evaluator performed the role of a user who is learning Japanese as their target language and their source language as English. For this reason, the evaluator was fluent in both English and Japanese. On average, the evaluator rated the tool 3.24 out of 5. This average is calculated from the seventeen opinion scale questions which are rated one to five.

The evaluator began the evaluation process by accessing an instructions document[6] in order to understand the purpose, capabilities, and necessary evaluation metrics of Immersio. The instructions document provided links to the Immersio web application[7] as well as to the Typeform evaluation form[8]. After adhering to the information in the instruction document and interacting with Immersio, the evaluator was asked to fill out the evaluation form.

The evaluation form consists of twenty-three questions including one question asking the evaluator of their Japanese language proficiency, seventeen opinion scale questions, and five open-ended questions. Both the Japanese proficiency question and the fourteen opinion scale questions are rated from one to five. The Japanese proficiency question asked the evaluator, "How well do you speak Japanese?", and is scored based on the JLPT proficiency levels where a selection of "1" denotes an N5 proficiency or a basic understanding of Japanese up to a "5" which resembles an N1 proficiency or a native proficiency. The opinion scale questions are rated from one to five where "1" entails poor quality while a "5" entails excellent quality. The five remaining open-ended questions concerned the

[6] https://docs.google.com/document/d/
[7] https://immersio.up.railway.app/
[8] https://evaluation-form.typeform.com/report/

overall quality of a particular page or the application as a whole as well as the suggestions for improvements of the application.

The evaluator scored a ”5” on the Japanese proficiency question with a native proficiency. Ten of the opinion scale questions concerned the chat page by asking the user to rate the Japanese accuracy of the AI models behind speech and text generation and comprehension of user text and user speech. Additionally, the evaluator evaluated the dictionary, how well the AI remembered information, how well the AI had prevented the user from speaking in a language that was not their target language, grammar accuracy, mood, and register of the Japanese generated by the AI models. The other six remaining opinion scale questions concerned the review page in which the evaluator evaluated the translation accuracy of the AI, the ability to generate both words and sentences, how well the corrected answer was annotated, and how well the AI prevented translating Japanese text in a language other than the source language English.

Immersio had performed well in Japanese in some aspects while failed in others. On the chat page, Immersio performed the worst in its ability to generate accurate dictionary definitions and was unable to transcribe audio from the user into Japanese text. However, Immersio performed best in its ability to generate textual Japanese and prevention of using English when it should have only allowed Japanese. On the review page, Immersio performed only slightly better than poorly in its ability to generate a translation of a word. On the other hand, Immersio performed best when the evaluator asked the AI to generate a translation of a sentence or message as well as its capacity to prevent translating Japanese in a language other than the source language, English.

The main issues with Immersio’s evaluation performance can be expressed in the evaluator’s comments to the open-ended section of the evaluation form.

The evaluator was unable to access the Part-of-Speech so the dictionary could not be opened. This problem is more than likely the result of the Part-of-Speech data entering the database asynchronously while the main message data is being sent back to the client. The Part-of-Speech data is intended to load around ten seconds after the REST API response for the client was returned because RabbitMQ may not have consumed or published quick enough between spaCy and Django. Perhaps for a similar reason, the evaluator had seen that the translations for the chat messages were blank. When the evaluator attempted to record audio, the page did not load – more than likely due to the Whisper STT service failing due to dependency, environment, or FFmpeg compatibility issues. The evaluator also reported minor grammar issues with the AI's text generation on the chat page. In terms of the reviews page, the evaluator had asserted that unless the answer is exactly the same as what is expected, it will be annotated erroneously by the AI. In terms of the spaced repetition abilities of Immersio, the evaluator claimed that it would be best to have the review items be available to retake after the review session. Overall, the evaluator had desired for the application to include all of the features described in the instruction document. A complete summary of all of the evaluator's responses, their associated questions, and ratings can be seen in Table 5.1.

Question in English	Question in Japanese	Rating
How well do you speak Japanese?	日本語をどの程度お話しできますか？	5
How accurate/natural was the application's chat page's AI's Japanese response?	アプリのチャットページのAIの日本語応答はどれくらい正確/自然でしたか？	5
How well has the AI adhered to the proper register, tone, or mood? (Did the AI respond casually or formally; For example: Did the AI use です or ます when appropriate?)	AIが適切なレジスター、トーン、またはムードにどれくらい適合していましたか？（AIはカジュアルにまたはフォーマルに応答しましたか？ 例：AIは適切な場面で「です」または「ます」を使用しましたか？）	4
How accurate was the AI's translation?	AIの翻訳の正確さはどの程度でしたか？	3
How accurate was the AI's translation at being grammatically correct?	AIの翻訳が文法的に正確であったかどうか、どの程度でしたか？	4

continues on next page

How accurate were the dictionary definition(s) for a given word? (The dictionary is activated by clicking on a word in a message)	特定の単語の辞書の定義は、どの程度正確でしたか？（メッセージ内の単語をクリックして辞書を表示します）	1
How accurate was the AI generated audio when speaking Japanese?	AIが生成した日本語音声の正確さはどの程度でしたか？	2
When speaking Japanese using the microphone button, how well did the AI transcribe the text into Japanese?	マイクボタンを使用して日本語で話した場合、AIがテキストをどの程度正確に日本語に転写しましたか？	1
When speaking Japanese using the microphone button, how grammatically correct was the AI as it transcribed the text into Japanese?	マイクボタンを使用して日本語で話した場合、AIがテキストを日本語に転写する際の文法の正確さはどの程度でしたか？	1

continues on next page

How well has the AI remembered information previously mentioned in a conversation?	AIが以前の会話で言及された情報をどの程度覚えていましたか？	4
How well did the AI prevent talking/messaging in English and only allowed conversing in Japanese?	AIは、英語での話しやメッセージの防止をどの程度うまく行い、日本語での会話のみを許可しましたか？	5
During a review session, how well was the AI able to grade the user's translated sentence?	レビューセッション中、AIはユーザーの翻訳された文章を評価するのにどの程度うまく対応しましたか？	3
During a review session, how well was the AI able to grade the user's translated word?	レビューセッション中、AIがユーザーの翻訳した単語を評価するのにどの程度うまく対応しましたか？	3

continues on next page

During a review session, how accurate was the AI able to generate the translation of a sentence/message?	レビューセッション中、AIが文やメッセージの翻訳を生成する能力においてどれくらい正確でしたか？	5
During a review session, how accurate was the AI able to generate the translation of a word?	レビューセッション中、AIが単語の翻訳を生成する能力においてどれくらい正確でしたか？	2
During a review session, how well was the corrected answer annotated?	レビューセッション中、修正された回答が適切に注釈付けされていたかどうか、どの程度でしたか？	3

continues on next page

How well did the AI prevent translating the Japanese text in a language other than English? For instance, the AI should ideally not allow the usage of Japanese when translating and should only allow English text during translation of a review item.	AIが日本語のテキストを英語以外の言語に翻訳するのをどれくらい防止しましたか？たとえば、AIはレビューアイテムの翻訳時に日本語の使用を許可せず、英語のテキストのみを許可すべきです。	5

continues on next page

How correct was the accuracy report after the reviews were finished? Ideally the accuracy report should mark a message as incorrect if a review item was marked incorrect at least once and the accuracy report should count a message as correct if it a review item was marked correct the first time.	レビューが終了した後の正確性レポートはどの程度正確でしたか？理想的には、正確性レポートは、少なくとも1回のレビューアイテムが誤ってマークされた場合にメッセージを不正確としてカウントし、レビューアイテムが最初に正しくマークされた場合にメッセージを正確としてカウントすべきです。	4

continues on next page

How well did the chat page perform? What did you like or dislike about the chat page?	チャットページ全体の性能はどの程度でしたか？チャットページについて好きな点や気に入らなかった点は何ですか？	”I was unable to access part-of-speech, so could not open a dictionary. Also translations for the chat was blank. I was able to record my audio, but could not upload so could not use it in the chat. Iy just continues to load forever. In the review page I think it is better make a line break when there is another answer. otherwise it can be confusing, if the answer is correct or not. overall I think it went successful with the chat, because grammer was correct and choose of words were natural. There were some grammer mistakes, but it is AI so I think it can be fixed within time. ”

continues on next page

What do you think could be improved on the chat page?	チャットページを改善するためには、どのような点が改善されるべきだと思いますか？	”Part-of-speech should be accessable to know each words. If the answer is correct but there is a another way to answer there sholud be line break between the phrase. Audio should be ready to upload after recording. Also I could not hear most audio voice from the AI chat. ”

continues on next page

How well did the review page perform? What did you like or dislike about the review page?	レビューページ全体の性能はどの程度でしたか？レビューページについて好きな点や気に入らなかった点は何ですか？	”page itself and its performance was perfect but the visuality I think it can be better. At first look it is difficukt to know if my answer is correct or not because green check and red x is small. Also it is hard to recoznize my answer is correct because unless each word is perfectly same it will be retyped by AI. ”
What do you think could be improved on the review page?	レビューページを改善するためには、どのような点が改善されるべきだと思いますか？	”The past review page and its resluts should be visable, so they would be able retake the test afterwards. ”
How do you think this application could be improved?	このアプリケーションをどのように改善できると思いますか？	”It needs all the features introduced in the instruction. ”

Table 5.1: The results from the Japanese evaluator

Chapter 6

Conclusions and Future Work

Language learning applications have advanced in the last two decades, especially because of the modern capabilities of artificial intelligence and large language models. However, many language learning applications that use artificial intelligence fail to integrate POS highlighting, dictionary lookup, and an easy-to-use spaced repetition system. For these reasons, we proposed a wholistic Japanese learning application called Immersio. Specifically, Immersio integrates the following elements: spaced-repetition, active recall, and an artificial intelligent conversational chat agent both in voice and in text. Immersio achieves this through its easy-to-use chat interface that supports Speech-to-Text, Text-to-Speech, dictionary lookup, artificial intelligent memory, Japanese-to-English translation and English grammar correction abilities, and Part-of-Speech technologies. The chat page additionally keeps track of previous conversations and the individual messages associated with them for access at later a later time. Immersio also implements spaced repetition through the Leitner system through its efficient review page that allows a user to have flexibility in translating Japanese text into English, which is graded via artificial intelligence and annotated. Moreover, an arbitrary selection of review

items on the review page can be added to a review session manually and reviewed review items will re-emerge to be reviewed depending on their SRS review frequency. The dashboard page keeps track of the amount of review items done per day as well as the most recent conversation.

As future work, we intend to improve automated testing so that the review and chat page work optimally and can easily be monitored if new changes are added, which could potentially break some functionality. For this reason, we will use more automated end-to-end testing using Cypress and more integration testing with Vitest. Also, we plan to implement performance testing to ensure that the client does not wait long times to receive responses from the backend as experienced by the evaluator. For instance, the evaluator was able to record their audio but did not receive a response back from the backend. This is particularly problematic when using RabbitMQ RPCs to retrieve information from the translation and grammar services. It is particularly vital for Immersio to handle high loads and traffic when multiple users are using the site at once. Furthermore, we will experiment with making the POS items more readily available to the user so they do not have to wait a long time to see their messages highlighted. The evaluator had mentioned that they were unable to access the dictionary because there was no POS. This is likely because spaCy had not returned a response fast enough for Django to update the database. If there is no POS information in the database and Django responds to the frontend with an empty POS JSON object, React will simply render the text of the message and not construct a message with POS items. One area of research that will be explored to handle this is the JavaScript WebSocket API[1]. We will also research using Celery[2] with Django and RabbitMQ in order to

[1] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets$_{A}PI$
[2] https://docs.celeryq.dev/en/v5.3.6/getting-started/introduction.html

efficiently communicate with one another. Moreover, we will look into upgrading the conversational agent by replacing GPT-3.5-Turbo with GPT-4 in order to handle some grammatical errors mentioned by the evaluator.

Also, more native Japanese speakers will be asked to evaluate the tool. Evaluators should encompass multiple regions of Japan that cover multiple dialects such as the Kansai and Tokyo dialects. For this reason, the evaluation form will have to be updated with new questions to reflect greater diversity for Japanese evaluation.

Additionally, we intend for Immersio to become more secure and therefore will need to be properly modified for production. One security flaw is that the registration page allows any password to be used such as "1234". We intend to allow only reputable passwords of a specific length and with certain characters instead of allowing simple passwords. Additionally, we will look into encrypting the user messages so that user messages are kept private in the database.

Furthermore, we will improve the user interface so that it is more consistent across all devices and screens. For instance, during development, there were instances where the screen would show white blocks at the bottom due to how styling was done. On mobile browsers, there are issues with how the chat page renders. Because of this, Immersio cannot be used on mobile devices. Additionally, we will look for better ways to improve the review page so that it is more intuitive. For instance, we will experiment with adding a line break, increasing the size of the green check-mark, and increasing the size of the red "X" on the reviews page as per the evaluator's comments. Additionally, we will experiment with ways of being able to review previous items listed on the review page so that a user can re-review necessary information.

In order to make development easier over time, we intend to more effectively implement continuous integration and continuous deployment. This work would

entail incorporating GitHub actions to work effectively with automated tests. GitHub would also need to automatically deploy the new services to our given hosting provider depending on the branch. For instance, we will need to create branches specifically for production and staging.

Bibliography

[1] B. Settles and B. Meeder, "A trainable spaced repetition model for language learning," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 1848–1858. [Online]. Available: http://aclweb.org/anthology/P16-1174 (document), 2.2, 2.3, 2.4

[2] T. T. Goh, N. A. A. Jamaludin, H. Mohamed, M. N. Ismail, and H. S. Chua, "A comparative study on part-of-speech taggers' performance on examination questions classification according to bloom's taxonomy," vol. 2224, no. 1, p. 012001. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/2224/1/012001 1.1

[3] J. M. J. Murre and J. Dros, "Replication and analysis of Ebbinghaus' forgetting curve," vol. 10, no. 7, p. e0120644. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4492928/ 2.1.1

[4] Jingyong Su, Junyao Ye, Liqiang Nie, Yilong Cao, and Yongyong Chen. Optimizing spaced repetition schedule by capturing the dynamics of memory. [Online]. Available: https://www-computer-org.ezproxy.southern.edu/csdl/journal/tk/2023/10/10059206/1LiKKhcXw52 2.1.1

[5] C. A. Mace, *Psychology of study*. Baltimore,: Penguin Books. [Online]. Available: http://archive.org/details/psychologyofstudoomace 2.1.1, 2.1.1

[6] S. Leitner and R. Totter, *So lernt man lernen: angewandte Lernpsychologie - ein Weg zum Erfolg*, 20th ed. Weltbild. [Online]. Available: https://archive.org/details/solerntmanlernenoooleit/mode/2up 2.1.1

[7] B. Tabibian, U. Upadhyay, A. De, A. Zarezade, B. Schölkopf, and M. Gomez-Rodriguez, "Enhancing human learning via spaced repetition optimization," vol. 116, no. 10, pp. 3988–3993. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6410796/ 2.1.1

[8] Piotr A. Wozniak, "Optimization of learning." [Online]. Available: https://super-memory.com/english/ol.htm 2.1.1, 2.1.1

[9] J. Ye, J. Su, and Y. Cao, "A stochastic shortest path algorithm for optimizing spaced repetition scheduling," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, pp. 4381–4390. [Online]. Available: https://dl.acm.org/doi/10.1145/3534678.3539081 2.1.1

[10] J. Ye. Free spaced repetition scheduler algorithm. [Online]. Available: https://github.com/open-spaced-repetition/fsrs4anki/wiki/The-Algorithm 2.1.1

[11] J. Ye. Spaced repetition algorithm: A three-day journey from novice to expert. [Online]. Available: http://bit.ly/41SaaHE 2.1.1

[12] P. Pimsleur, *How to Learn a Foreign Language Hardcover – Unabridged*. [Online]. Available: https://www.amazon.com/-/es/Paul-Pimsleur-Ph-D/dp/1442369027 2.1.1, 2.2.6

[13] P. Pimsleur and T. Quinn, *The psychology of second language learning: papers from the Second International Congress of Applied Linguistics, Cambridge, 8-12 September 1969*. Cambridge University Press, meeting Name: International Congress of Applied Linguistics. 2.1.1

[14] P. Pimsleur, "A memory schedule," pp. 73–75. 2.1.1

[15] M. Muljono, U. Afini, C. Supriyanto, and R. A. Nugroho, "The development of Indonesian POS tagging system for computer-aided independent language learning," vol. 12, no. 11, p. 138. [Online]. Available: http://online-journals.org/index.php/i-jet/article/view/7383 2.1.2

[16] J. Zhang and H. Zhang, "A corpus-based case study on the POS tagging of self-referential lexemes in the Contemporary Chinese Dictionary," vol. 10, no. 8, p. 879. [Online]. Available: http://www.academypublication.com/issues2/tpls/vol10/08/05.pdf 2.1.2

[17] S. Lu, *Xian Dai Han Yu Ci Dian: 2002 Zeng Bu Ben*. Shang Wu Yin Shu Guan. 2.1.2

[18] S. McGraw, "JMDictDB." [Online]. Available: https://gitlab.com/yamagoya/jmdictdb/ 2.1.2

[19] R. Jakobson, "On linguistic aspects of translation." [Online]. Available: https://web.stanford.edu/~eckert/PDF/jakobson.pdf 2.1.2

[20] J. W. Breen. Building an electronic Japanese-English dictionary. [Online]. Available: http://nihongo.monash.edu/jsaa_paper/hpaper.html 2.1.3

[21] EDRDG. Electronic dictionary research and development group. [Online]. Available: https://www.edrdg.org/ 2.1.3

[22] J. W. Breen. JMdict: a Japanese-multilingual dictionary. [Online]. Available: https://www.edrdg.org/jmdict/jmdictart.html 2.1.3

[23] M.-C. De Marneffe, C. D. Manning, J. Nivre, and D. Zeman, "Universal Dependencies," pp. 1–54. [Online]. Available: https://direct.mit.edu/coli/article/doi/10.1162/coli_a_00402/98516/Universal-Dependencies 2.1.4

[24] Universal Dependencies. Universal POS tags. [Online]. Available: https://universaldependencies.org/u/pos/index.html 2.1.4

[25] Universal Dependencies. Universal features. [Online]. Available: https://universaldependencies.org/u/feat/index.html 2.1.4

[26] K. Kastner, J. F. Santos, Y. Bengio, and A. Courville, "Representation mixing for TTS synthesis." [Online]. Available: http://arxiv.org/abs/1811.07240 2.1.4

[27] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision." [Online]. Available: http://arxiv.org/abs/2212.04356 2.1.4

[28] K. Sharma, A. Verma, and P. Verma, "Analysis and implementation of microservices using docker," in *Advanced Network Technologies and Intelligent Computing*, I. Woungang, S. K. Dhurandher, K. K. Pattanaik, A. Verma, and P. Verma, Eds. Springer Nature Switzerland, vol. 1797, pp. 413–421, series Title: Communications in Computer and Information Science. [Online]. Available: https://link.springer.com/10.1007/978-3-031-28180-8_28 2.1.4

[29] K. Chołody and S. Przyłucki, "Analysis of the impact of using containerization techniques on application performance in Python," vol. 29, pp. 413–420.

[Online]. Available: https://ph.pollub.pl/index.php/jcsi/article/view/4589 2.1.4

[30] B. Peng, M. Galley, P. He, C. Brockett, L. Liden, E. Nouri, Z. Yu, B. Dolan, and J. Gao, "GODEL: Large-scale pre-training for goal-directed dialog." [Online]. Available: http://arxiv.org/abs/2206.11309 2.1.4

[31] A. Nugroho, Widyawan, and S. S. Kusumawardani, "Distributed classifier for SDGs topics in online news using RabbitMQ message broker," vol. 1577, no. 1, p. 012026. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1577/1/012026 2.1.4

[32] AnkiWeb. What spaced repetition algorithm does Anki use? - frequently asked questions. [Online]. Available: https://faqs.ankiweb.net/what-spaced-repetition-algorithm.html 2.2.1