Kanji Translation App (KTA)

Client: Keith Stevens
Team: AI-Network

Members:
Cody Hubbard, Eric Yuan, Nikola Samardzic,
Weijia Yuan, Xinyu Wang, Zhao Weng

1. Introduction and Application Overview

1.1. Purpose

The purpose of this Software Requirement Specifications document is to provide a description of the features and use of the Kanji translation application. It will detail the functional and nonfunctional requirements of the application software, as well as its UI and technologies being used. This document also details the system constraints, interface and interactions with other external systems.

1.2. High level functions and objectives

The overall goal of the Kanji Translation App is to provide Kanji to Hiragana translations so users can easily see the phonetic spelling Kanji characters they may be having trouble understanding. The app is focused towards intermediate Japanese speakers as they need to know a decent amount of Japanese to make full use of the software. Additionally the KTA is focused on ease of use use with a focus on response time and readability.

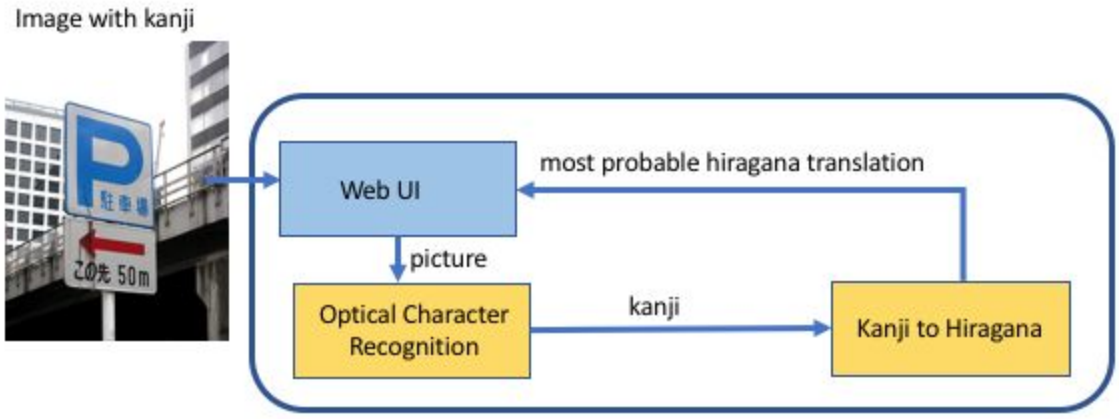
1.3. Roles and responsibilities of users

The only responsibilities of the user is to submit an image containing Kanji characters to be translated.

1.4. Dependencies

The KTA has several key dependencies that it will rely heavily on. The KTA will rely on an open source Japanese optical character recognition (OCR) library to identify the Japanese characters within the images submitted by the user. It will then rely on an an open source Kanji to

Hiragana translation library to compute and output the most likely Hiragana translation. Thus the KTA will have several attributes, namely accuracy and consistency which depend on these libraries.



1.5. Design and implementation constraints

1.5.1. The implementation of all back end functionality will rely on the specific open source libraries selected for OCR translation. The backend API's will have to be written so that it can communicate with both libraries without issue.

1.5.2. The KTA shall initially be implemented as a web application with all main functionality hosted on a web server on a cloud service system.

1.5.3. Teh KTA web application should be accessible from both personal computers and mobile devices.

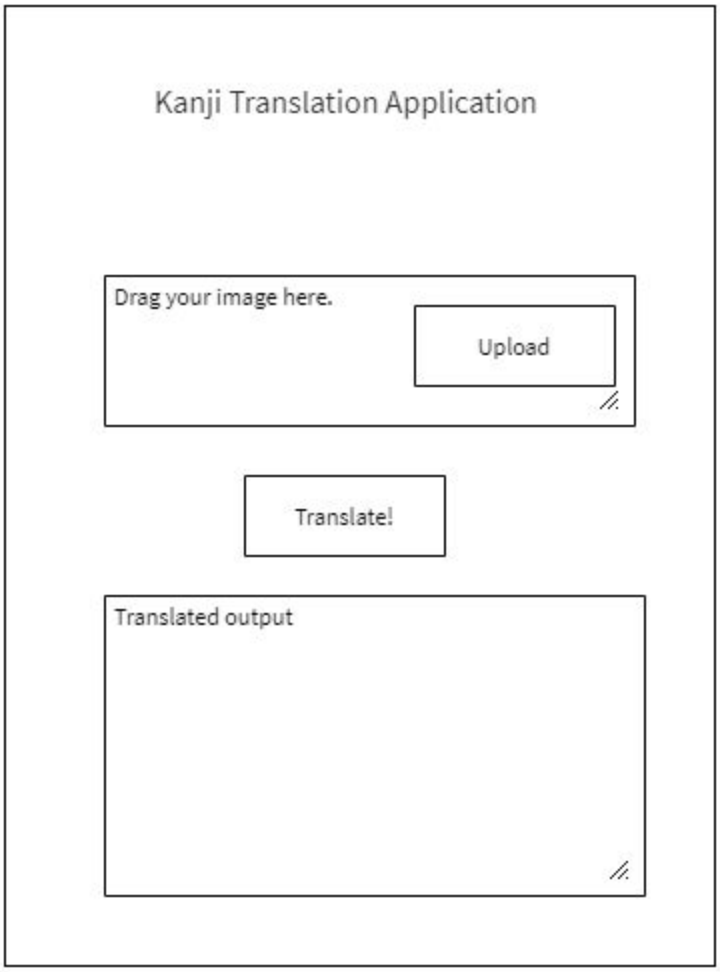
2. External Interface Requirements

2.1. User Interface

Since KTA will be first implemented on a web portal and later ported to mobile devices there are two user interfaces included in this project.

2.1.1. Web UI

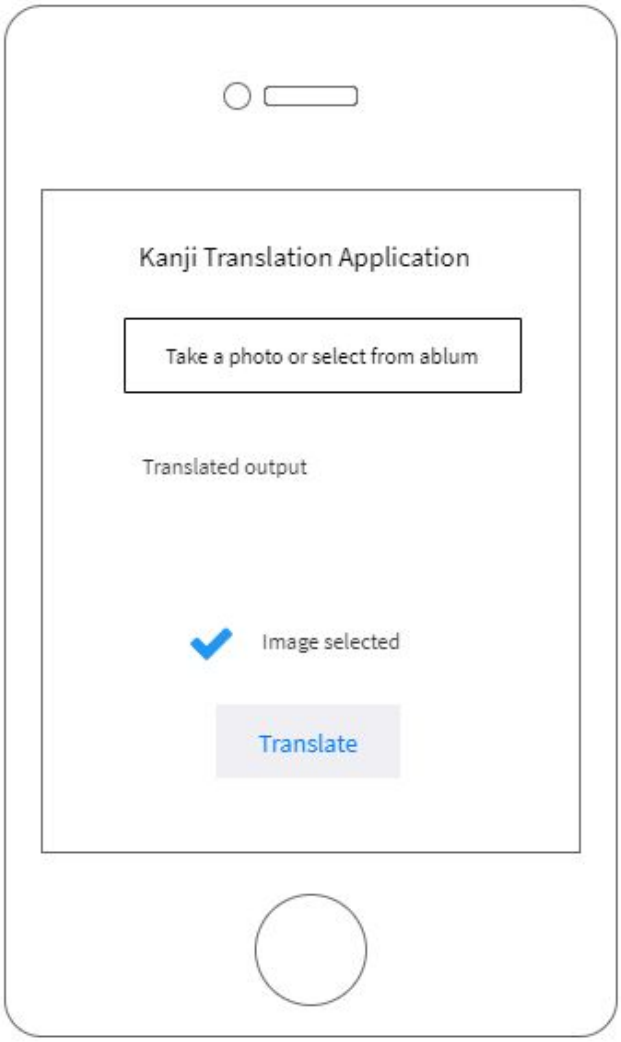
For the web portal, the main user interface contains a title, an upload button, an area for dragging/dropping images and a translation button.



The application takes an image as input. The user could choose to either drag and drop an image into the designated area or click the upload button to select an appropriate image to upload. Once an input is detected and user clicks the translate button the application will translate all recognized Kanji characters. The output will then be shown to the user and they will be presented the ability to view more info about specific translations or upload a new image to translate.

2.1.2. Mobile UI

The user interface for the potential IOS application will have similar functionality.



The mobile version will have all the functionality of the personal computer version as seen in the UI mockup. Additionally the mobile version will allow the user to use their mobile device's camera to take and upload a photo to translate.

2.2. Software Interface

For web portal, the application requires a database system to store a translation table with respect to each single character of Kanji form. Furthermore, an image processing library is required. (Undecided, depending on API)

3. Functional Requirements and Priority

3.1. Project Scope

3.1.1. This project depends on both open source Optical Character Recognition (OCR) and Kanji-to-Hiragana software. Our system will not implement either of these two pieces of software, but will rely heavily on open source projects of both. However, we are responsible for choosing OCR and Kanji-to-Hiragana that are both reliable and work well with our overall framework.

3.1.2. The following are within the scope of this project: Web UI, Feeding OCR with submitted queries from users, feeding the output of OCR to kanji-to-hiragana, and displaying the kanji-to-hiragana output onto the Web UI of the user that submitted the query.

3.1.3. The system will provide the best translation possible given the image, and if there is a problem with the image given useful feedback will be supplied instead.

3.2. Functional Requirements

- All requirements are submitted by Keith Stevens.
- Higher priority number corresponds to higher priority

Requirement	**Priority**
The application needs to check for correct file-type before an image is uploaded (should accept jpeg, or png)	3
The application should give have useful explanations for when something goes wrong: - Notify user of issues with pushing image to our system - Notify user of issues with translation (e.g. kanji not in translation database, cannot recognize kanji on image) - Notify user of issues with pushing translation results to the UI - Provide a catch-all notification in case something goes wrong that is not captured by any of the above notifications.	2
Allow users to upload photos of kanji through a Web UI	5
Provide user with translation of kanji on image through the same Web UI used for uploads	5

Use identical framework to translate english into phonetic pronunciation symbols	2
The application should be able to parse a sentence in Japanese and divide it into different words. This implies that the app should be able to offer contextual translations and translations of full sentences.	5
Would be nice to use Node.js for the Web UI + web backend	3
Required documentation shall include instructions on how to start the web server, markdown readme file, framework and architecture of UI, SRS	5
Allow for the user to see alternate translations for a given kanji if first pick does not match user need.	4
The application is required to handle nefarious requests, crash without information loss when crash is inevitable.	3

4. Non-Functional Requirements

4.1. Performance

4.1.1. Since response time is an essential attribute for web apps, It is required that the time between when users upload their images and when they see the pronunciations on the website should be less than 2 seconds.

4.1.2. The time between when user clicked the submit button and when server gets access to the image should be less than 1 second

4.1.3. The time between the beginning of the image processing and getting the machine readable pronunciation should be less than 0.5 seconds.

4.1.4. The time between when the server transforms the pronunciation into human readable format should be less than 0.5 second

4.1.5. The time between when the server starts sending out the pronunciation and when the pronunciation appears on the web interface should be less than 1 second.

4.2. Reliability

4.2.1. With 90% probability, users can upload images at any time within each hour without errors.

4.2.2. With 90% probability, the web app can correctly process the image and get pronunciation.

4.2.3. With 90% probability, the web app can show the pronunciation in correct format on the web interface

4.2.4. The application shall crash gracefully if it does so, that is it will exit itself, without needed to be killed by the browser or operating system.

4.3. Security

4.3.1. The image uploaded by each user should be invisible to other users.

4.3.2. The pronunciations and images uploaded should not be able to be accessible by other websites.

Project Timeline

1. Requirements engineering

Week 3, Jan 22nd - Jan 28th

- Work with the client to discover functional and nonfunctional requirements as well as project scope.
- Research open source OCR and translation software.
- Model the software and research other translation software with similar functionality.

2. Web Application Development

Weeks 4 - 6, Jan 29th - Feb 18th

- Design and implement the APIs for communicating between the chosen OCR and translation.
- Implement the front end where users upload images to be translated and translation output is displayed to the user.
- Implement front end user navigation.
- Implement the backend functions where images are processed and translated.
- Implement all above functionality for the web application version.
- Present current build.

3. Testing and improvement planning

Week 7, Feb 19th - Feb 25th

- Thoroughly test current implementation and identify needed improvements.
- Design improvements and plan further development.

4. Mobile development & extended functionality

Week 8-9, Feb 26th - Mar 11th

- Develop IOS application or mobile version of current web application.
- Implement planned improvements and extended functionality

5. Review & Enhancement

Week 10, Mar 12th - Mar 18th

- Enhance the performance of existing features
- Further optimize response time. If already <2s try to reach 200ms.
- Review user feedback and make changes/improvements as needed