multi-lingual chatting application

Tai Elliott Sinclair

Contents

[Analysis 2](#_Toc66452979)

[Research 2](#_Toc66452980)

[Outline 2](#_Toc66452981)

[Survey 2](#_Toc66452982)

[Survey Analysis 3](#_Toc66452983)

[Decisions 5](#_Toc66452984)

[Requirements 6](#_Toc66452985)

[POC 8](#_Toc66452986)

[Design 8](#_Toc66452987)

[Frontend 8](#_Toc66452988)

[App Form Design 8](#_Toc66452989)

[Login Form Design 8](#_Toc66452990)

[Create Account Form Design 9](#_Toc66452991)

[Backend 10](#_Toc66452992)

[Login Diagrams 10](#_Toc66452993)

[Create Account Diagrams 11](#_Toc66452994)

[Main Form Diagrams 11](#_Toc66452995)

[Testing Strategy 15](#_Toc66452996)

[Technical Solution 15](#_Toc66452997)

[Testing 15](#_Toc66452998)

# Analysis

## Research

### Outline

I have been commissioned to create a chatting application with cross-language compatibilities. Loki has requested this on behalf of his company who work as a start-up creating useful software for schools. The language chosen to test was Japanese due to its diverse alphabet. Language such as Chinese which use characters called kanji were decided against because of the complexity of the character system. Japanese contains alphabets which are non-roman character based yet can be ‘somewhat’ simply transliterate into roman characters (these alphabets being *katakana* and *hiragana*). Because of this, Japanese was chosen as the language for which to create this application.

The goal of this application is to support students who are learning Japanese by creating an environment in which they can communicate with others in Japanese. This will involve typing in roman characters and having them transliterated in real time, as well as the ability to draw in *hiragana*, and have the drawing recognised and displayed as plaintext. This will utilise machine learning.

### Survey

In order to make decisions on how everything should be set out, I gave out a small 3 question survey to 30 students of Japanese.

Note: since Japanese is rarely included in school curriculums, when ‘school’ is referenced it simply refers to places of education, as such anything from universities to webinars count and the students of them are included in the survey.

Survey:

What is your age group?

0-16 17-26 27-36 37-46 47-56 57+

* This question is arguable the most important to consider for all frontend development. Decisions about UI, simplicity and style are mostly driven my customer age. For example, the young (on average) have a smaller attention span and are unlikely to spend a lot of time trying to figure out a complex UI.

What is your experience level with Japanese?

Beginner Intermediate Early-Expert Expert Fluent

* This was a question insisted by Loki’s company who intend to use this information for marketing purposes. However, it also has use for the developer as beginners are likely to require assistance with writing characters.

Do you prefer simplistic or complex designs?

Simplistic Complex Couldn’t care less

* This is an important question as it allows me to decide how simple to make the UI. Of course the more complicated the UI the longer it’ll take to build

### Survey Analysis

The results of this small survey are given here:

The first question was asking the age of those learning the language.

The chart shows that a vast majority of Japanese learners are between 17 and 26. Since most people around that age have spent their lives on devices, I won’t need to make the UI super intuitive. This allows me to choose a higher-level language which are often faster to code with.

The second question was regarding the proficiency level of the users.

This was a surprising result, as the favoured category would be beginner. That being said, it seems understandable that those who are just starting out are unlikely to want to communicate with other learners just yet. This result means, that the majority of learners consider themselves intermediate. Of course, this is answered by the person themselves, so it is easily plausible that someone over – or under – estimates their ability. The lack of specificity when giving categories could lead to this. For future surveys it will be important to give specific ideals for each category, so people are surer of their standpoint. However, since the survey has already been administered these are the results we will be using for the decisions.

Most of the users being intermediates allows for an easier to use (but less helpful) draw input. If they were all beginners, I would have had to create a way of explaining the drawings, as to make them easier for a beginner to draw. That is, I would have created correction software. Which would be very useful for beginners, however, for higher levels, ease is more important than helpfulness. Which is the same reason why so many people find autocorrect to be very troublesome.

The final question was asking the users opinion on whether the design should be simple or complex.

This was very close between simple and complex. So as to account for both, I intent to add in a login mechanism and the ability to sift through the users and send messages to specific ones. This takes away from the most basic which would simply be a chat room where everyone can message everyone, however it does not add unnecessary complexity which would become annoying to work with.

### Decisions

The first, and biggest, decision to make when it comes to a project like this is ‘which language to code in?’

I settled on python, this seems like a risk at first, because python is an interpreted backend language. However, these seemingly crucial issues can be avoided. Python, while interpreted, can still be made executable by using packages such as pyinstaller, and there is a library called pygame which allows 2D forms and games to be developed using python. This does avoid the issues, however one could ask whether the dodging would be worth it, since other compiled languages have build in frontend functionality.

The reason I chose python for this project was partially because I have 6 years of experience working with the language and a good understanding of its packages, and therefore confidence in my ability to create a working application. It was also due to the versatility of the backend. While other languages would allow a strong frontend, when it comes to messaging apps (especially those with AI implementation) having a well-done, neat backend is far more important. The final reason, and biggest, was that it was the language open on my computer when I started.

A second decision I had to make was whether to create the AIOCR from scratch or use a python package.

I ended up settling on a package mostly because of speed. Not simply the time it would take for me to code it (which would be longer, and would end in a longer deadline), but the time it would take to convert the image to text. That would be a big reason for someone not to use that feature. People don’t like waiting, even if it’s only milliseconds.

Why would a package be faster?

Using packages like opencv and easyocr will allow the program to quickly and efficiently unpack all of the images, turn them computer readable. It would them throw them through the OCR which would convert them to text.

While my handwritten code would accomplish the same thing, it would be doing it in a much less efficient manner and would lead to a few milliseconds of difference between clicking the enter button and seeing the text appear. Given the need to make things quick and argument could be made that a lower-level language like C would have been much more effective at this task, to which I say… yeah, you’re probably right… but I’ve done it now.

I have decided to use easyocr instead of a more popular package such as tesseract. This is largely due to the fact that tesseract is a much larger package and will require more space on the end users system, and it would be inefficient and annoying for them to have to remove other applications from their system because we use an unnecessarily beefy package. The other reason is that I tried using tesseract and could get it to work.

The final large decision I had to make was whether to have two separate forms for input. One being a draw input and one being a text input or to have a single form with both options.

I decided on a single form, because drawing the characters is time consuming, so it is unlikely that a user will draw the entire message. As such, it would be inconvenient for them to continuously switch between the two options and will likely end in the draw function not being used.

## Requirements

1. Allow the user to enter English text which is to be transliterated into Japanese *hiragana.*
   1. A text box must allow the user to type with an English keyboard.
   2. The text entered must be converted to Japanese.
   3. The text must move to a new line when it covers the text box’s dimensions.
2. Allow the user to draw the character.

2.1) There must be a box with draw functionality.

2.2) Once enter button has been pressed an image must be taken.

2.3) The image must be processed.

2.4) This must be sent through an OCR.

2.5) Character must be returned.

2.6) Character must be displayed on the input box.

1. Allow the user to login.

3.1) A login-form should be displayed.

3.2) The user should be prompted to enter a username and password.

3.4) The backend should then query the database for a matching username.

3.5) The entered password should then be hashed and compared to that username’s affiliated password.

3.6) If matched the user should be logged in to their page.

3.7) If they don’t match an error message should be displayed.

1. Allow the user to create an account.

4.1) A create account button should be displayed on login.

4.2) On button click a new form should be displayed.

4.3) The form should prompt the user to enter a username, password, first name, and last name.

4.4) The backend should then store this information in a database.

4.5) The user should be told that the account was created.

4.6) The user should be automatically logged in.

1. Allow the user to find other users.

5.1) There should be a search box.

5.2) The user should be able to search a username.

5.3) All matching names should be displayed and clickable.

5.4) Once clicked the conversation should load.

# POC

# Design

## Frontend

### App Form Design

[Username]e

Enter

Search…

Message…

[Contact Name]

### Login Form Design

Create Account

Login

Username…

Password…

Enter

### Create Account Form Design

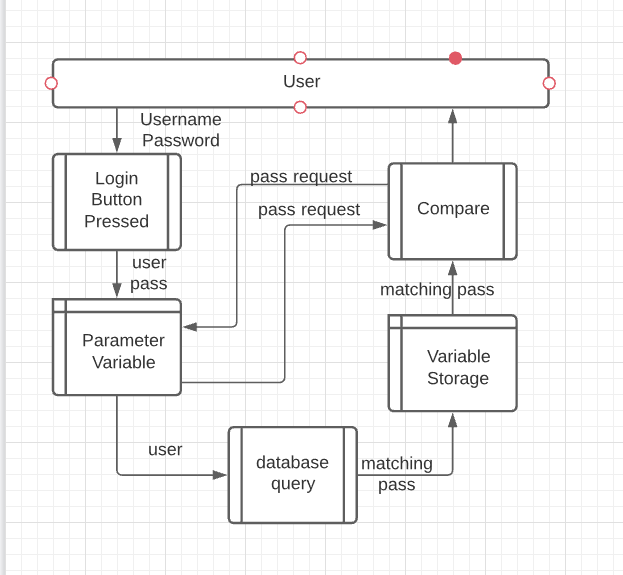
I would do the last bit of this form, but unfortunately, word is a twat.

### Class Diagrams

## Backend

### Login Diagrams

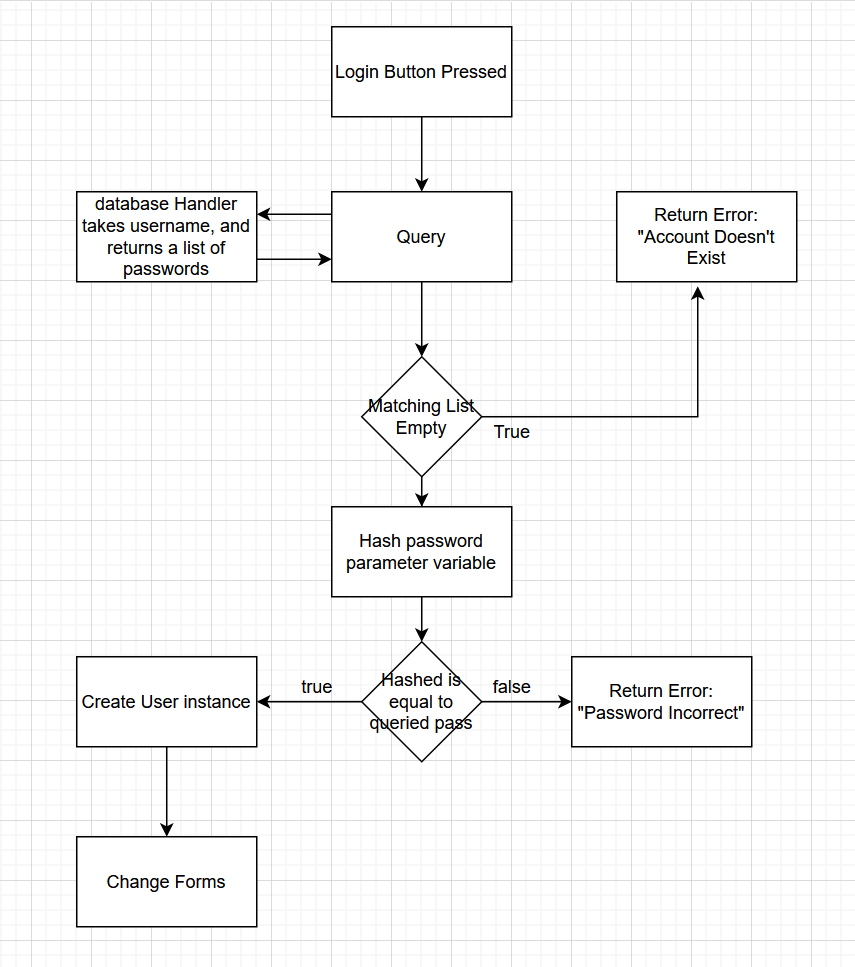
#### Login Button DFD



While this is a fairly convoluted diagram, considering how simple the premise is, it is easily explained.

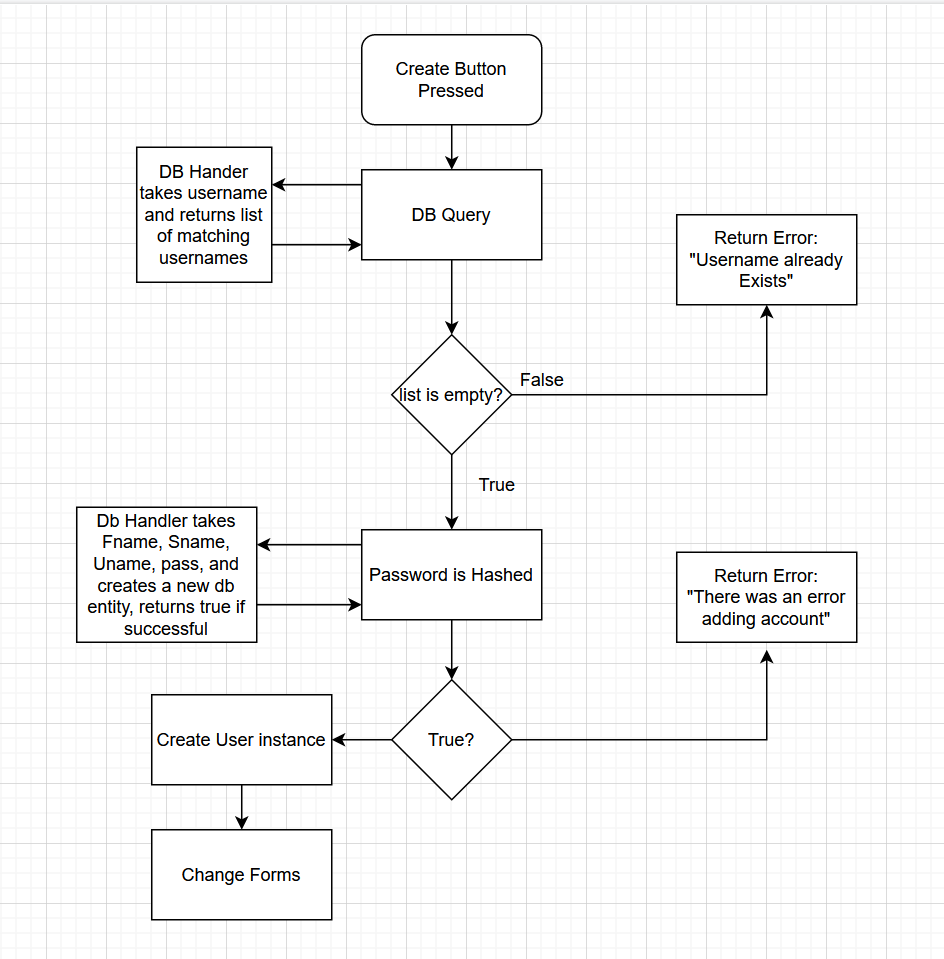
The user types in a username and password. Then clicks the Login Button, this data is then sent as parameter variables to a handler function. This then sends the username to a database query function which returns a matching password which is held as a variable. This is sent to a compare function which requests the pass data from the parameter variable storage, this is returned. Then the result (true or false) is sent to the function which handles the user.

#### Login Button Pressed Event Handler FC



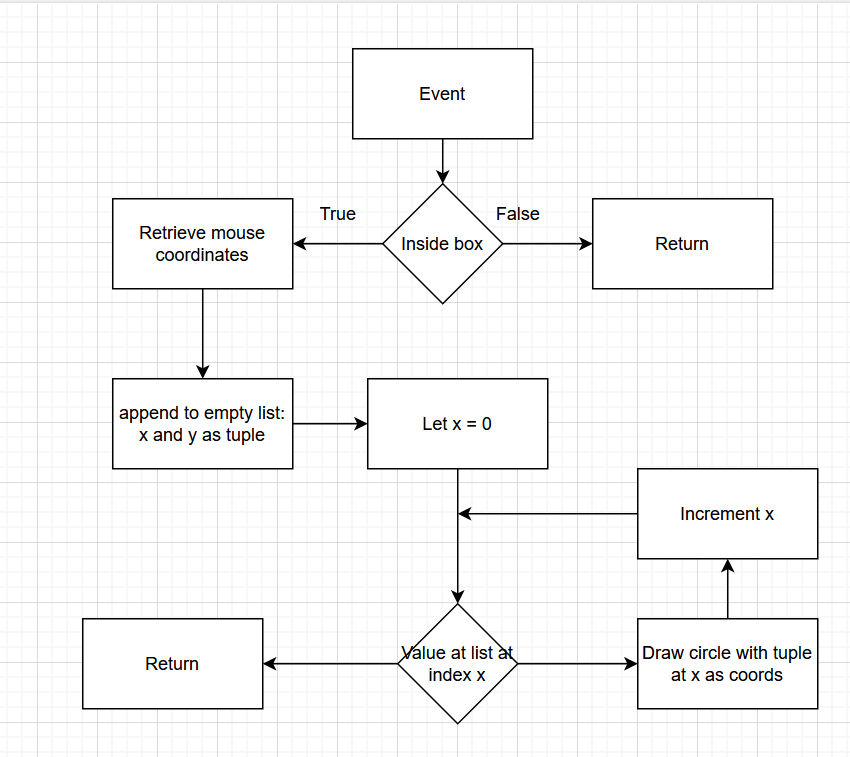
### Create Account Diagrams

#### Create Button Pressed Event Handler FC

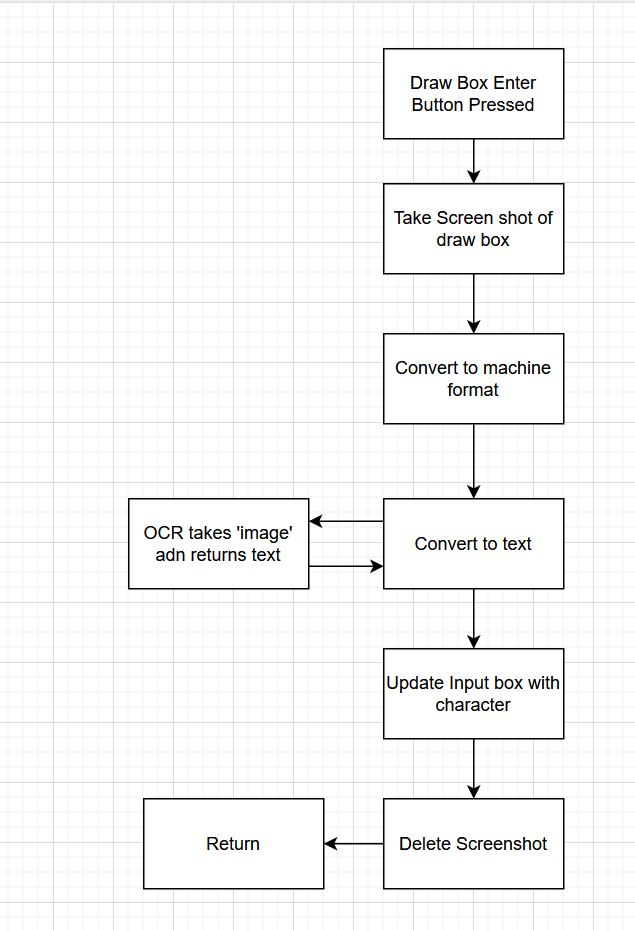


### Main Form Diagrams

#### Draw Box FC



#### Draw Box Enter Button Pressed



+-

# Testing Strategy

## Criteria

From the requirements, the main Testing criteria are:

* Able to type.
* Text goes to hiragana.
* Box can be draw inside.
* User can login.
* Account can be created.
* User can search.
* Messages are displayed.

Sub criteria include:

* Button Functionality.
* Error Handling.
* Correct name displayed.

## Testing Tables

I will test the solution by three criteria: “Typical”, “Erroneous”, and “Extreme”. Using three examples of each.

### Able to Type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No. | Description | Data Type | Expected | Pass/Fail | Cross Ref. |
| 1 | This test is checking that the user can enter text without issue | Typical | Once a keyboard button is pressed the letter appears in the box |  |  |
| 2 | This test is checking that the text is converted to hiragana once typed | Typical | Once 2 or more suitable characters have been displayed |  |  |
| 3 | This test is checking that the text never exceeds the bar length | Typical |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |
| 6 |  |  |  |  |  |
| 7 |  |  |  |  |  |
| 8 |  |  |  |  |  |
| 9 |  |  |  |  |  |

# Technical Solution

# Testing