

令和元年度卒業研究

**Web System to Help Non-Native English Speakers to Study
Specialty Subjects in English**

National Institute of Technology, Tokuyama College
Computer Science and Electronic Engineering

Haidah Waznah Binti Abdul Gafar

Abstract

English language mastery has become one of the most needed skills in this modern era. For certain countries, studying English as a foreign language proves to be difficult. When compared to other countries that uses English as their second language, one of the differences noted was that the latter are constantly put in an English environment. For example, they study certain subjects such as Science and Mathematics in English.

Specialty Studies in English (SSiE) is a web system developed to help non-native English Speakers to study specialty subjects in English without the need to translate texts into their native language. SSiE is built using NodeJS server and has two functions; an English-English dictionary (EED) and Simplifier.

EED takes single words and even whole paragraphs as input and outputs the definitions for each word. It utilizes Oxford Dictionary's API services to retrieve the definitions. On the other hand, the Simplifier simplifies text paragraphs into an easier-to-understand version. This is done through machine learning. The EED was successfully achieved, however, the simplifier system is still in development.

As English is used as communication language globally, most educational theses and resources are written in English. Hopefully, with the help of SSiE students can master English easier and with it, have access to various resources to deepen their knowledge or even change the world.

Contents

1	Introduction.....	3
2	System Construct.....	4
2.1	Development Environment & Concepts.....	4
2.1.1	NodeJS.....	4
2.1.2	Express.....	5
2.1.3	EJS.....	6
2.1.4	CSS.....	6
2.1.5	JavaScript.....	6
2.1.6	API.....	7
2.1.7	Text Simplification.....	7
2.1.8	Machine Learning.....	8
2.1.9	Python.....	8
2.2	English-English Dictionary (EED).....	9
2.2.1	npm Packages & Node Modules.....	9
2.2.2	Oxford Dictionaries API.....	12
2.2.3	System Operation.....	13
2.3	Simplifier.....	16
2.3.1	Moses SMT.....	16
2.3.2	Boxer (C&C Parser).....	17
2.3.3	Stanford Toolkit.....	18
2.3.4	Giza++.....	18
2.3.5	NLTK Toolkit.....	19
2.3.6	System Operation.....	19
3	Evaluation.....	22
3.1	English-English Dictionary.....	22
3.1.1	Response Time.....	22
3.1.2	Asynchronous JavaScript.....	23
3.2	Simplifier.....	24
4	Conclusion.....	25
5	Acknowledgements.....	26
6	Reference.....	27
7	Program List.....	29

1. Introduction

In the modern world of globalization, it has become more and more important to master English Language as English is used as a communication tool internationally. However, in certain countries such as Japan, its people have a hard time mastering the language. One of the factors that may cause this is that they were not put in an English environment and depended too much on translations to their native language.

An English environment is an environment where there is constant source of English around. For example, billboards written in English, or people around conversing in English. In an English environment, one will be constantly exposed to the English language; see English words, speak English, hear English sentences spoken, etc.

In some countries that uses English as their second language, students study certain subjects such as Science and Mathematics fully in English. In Malaysia, this was done under the Teaching and learning of Science and Mathematics in English (PPSMI) policy. Under this policy, all Science and Mathematics subjects including Biology, Chemistry and Physics are taught fully in English. All textbooks and exams questions were also written in English. With this, the people are constantly being put in an English environment. It is believed that by being surrounded by English often, Malaysians were able to master English easier compare to countries like Japan.

With this theory in mind, I am proposing a web system that could help non-native English speakers to study specialty subjects in English called ‘Specialty Studies in English’ or SSiE. SSiE has two functions. First is an English-English dictionary translator (EED). Users can input single word or entire text and the system in return will output definitions for each word. The definitions for each word were retrieved from the Oxford Dictionary API. The second feature is a Simplifier. Users can input entire text and the system will output a simpler and easier-to-understand version of the text. This is implemented through machine learning.

2. System Construct

As mentioned before, SSiE has two functions. EED and Simplifier. SSiE is built using NodeJS server and implements an abundance of services and tools. Fig. 1 shows SSiE's home page. From the home page, users can move to EED and Simplifier page by clicking the 'DICTIONARY' and 'SUMMARIZER' button respectively. EED has adopted various node modules and an API service. Simplifier on the other hand, adopted Moses Statistical Machine Translation (SMT) system, Giza++, Boxer, and Stanford toolkit. However, Simplifier was not able to be achieved in the end. Every system, tool and modules adopted along with their role will be explained below.

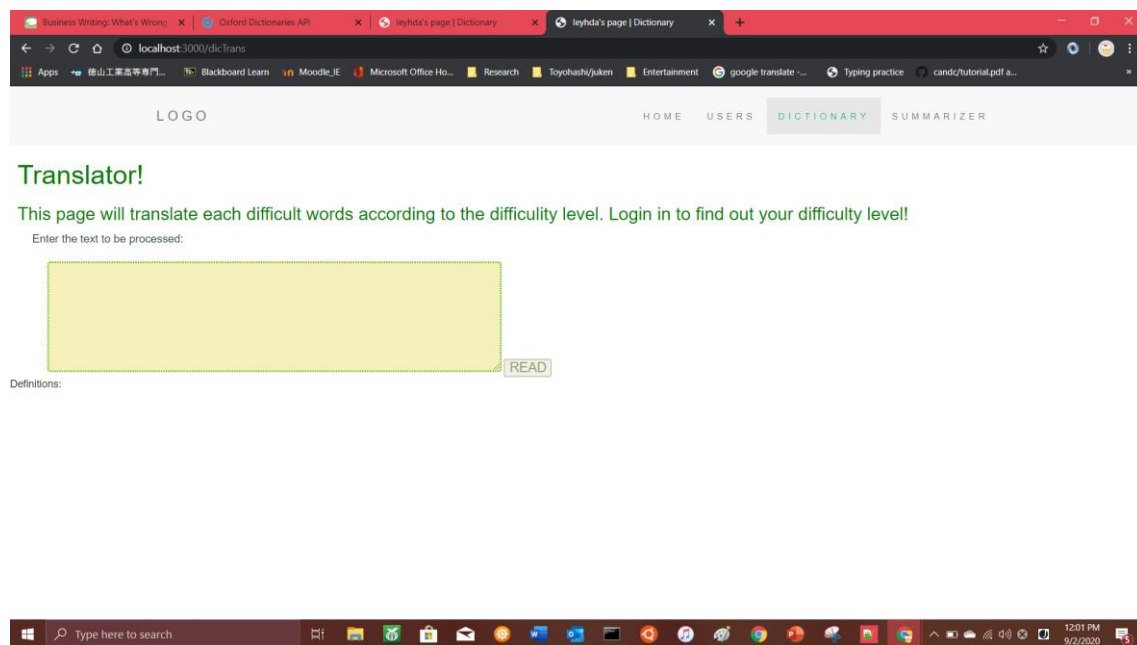


Fig. 1 SSiE home page

2.1 Development Environment & Concepts

2.1.1 NodeJS

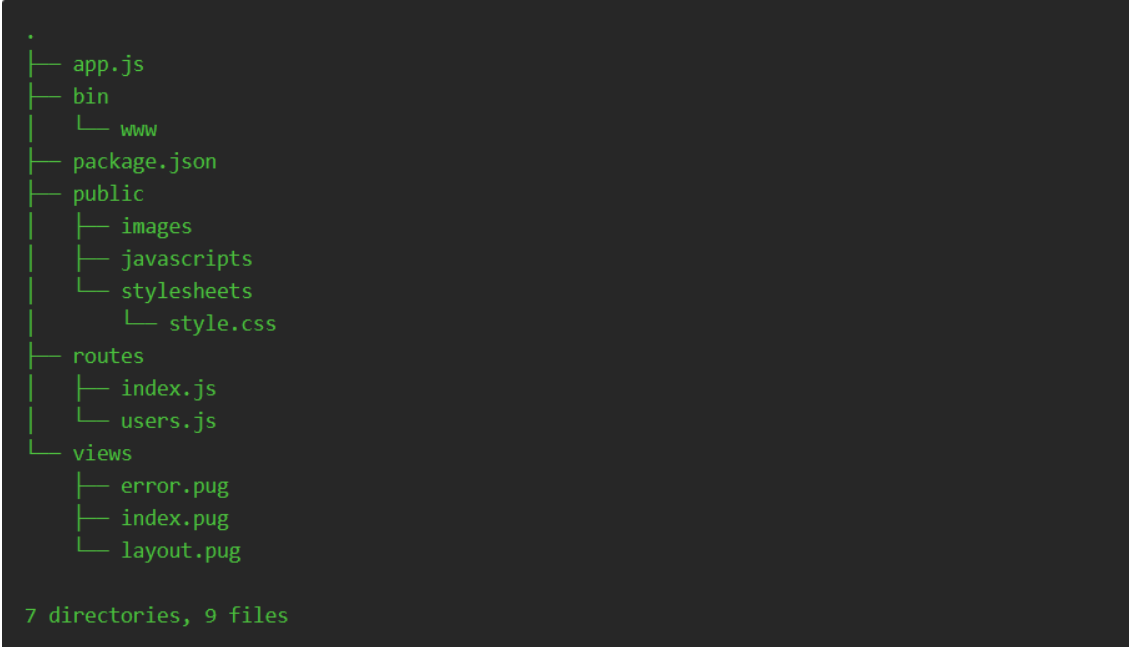
NodeJS is an open-source, cross-platform, server-side programming platform built on Google Chrome's V8 JavaScript runtime environment which executes JavaScript code outside of a browser[2]. It has a built-in module which allows data transfer to Hyper Text Transfer Protocol (HTTP). A module is any file or directory that can be loaded by the NodeJS using the `require()` function. With the HTTP module, NodeJS can be implemented as a web server.

NodeJS is asynchronous which means it does not wait for an API call's response before calling another one[1]. This makes it light-weighted and efficient to develop a real-time web application. NodeJS can be installed from their homepage and run using the command prompt.

2.1.2 Express

Express is a web application framework for NodeJS. It is very minimal and flexible and provides a reliable set of features to aid in building a web or mobile application[3]. It provides a thin layer of basic web application features without clouding the existing NodeJS features.

Express can be installed with the node package manager (npm) command in the command prompt. npm can be run once NodeJS is installed in the computer. With the Express application generator, an application generator can be quickly created. The generated app has the following directory structure (assuming the view engine is pug):



```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug

7 directories, 9 files
```

Fig. 2 Express-generated application skeleton

Here, 'app.js' is the main code – the server, and 'package.json' lists the dependencies of the app. There are 3 directories in a basic Express-generated application skeleton; public, routes, and views. In the 'public' directory is where all the static files such as images, JavaScript and CSS files related to the web app is stored. On the other

hand, NodeJS files to be served as the client side is stored in the 'routes' directory and the 'views' directory is where the templates (or view) files are located.

2.1.3 EJS

EJS (Embedded JavaScript) is one of the view (or templating) engines adopted by Express. View engines allow an Express app to serve static files[4]. During runtime, the variables in a template file are replaced with actual values and is transformed into an HTML file to be sent to the client. This approach makes it easier to design an HTML page. There are a few popular view engines such as Pug, Mustache and EJS. The Express application generator has Jade as its default view engine, but it also supports several others.

EJS is simple as it helps generate HTML markup using plain JavaScript[5]. This leads it to be easier to debug, as it is plain JavaScript exceptions, there is no intermediate language which may cause language mix-ups.

2.1.4 CSS

Cascading Style Sheet (CSS) is a simple language designed to help style the presentation of a markup language such as HTML[6]. With CSS developers can style the background colors, fonts and animate web pages. In SSiE, CSS is also implemented to create forms for the purpose of adding users to database.

2.1.5 JavaScript

JavaScript (frequently abbreviated to JS) is a lightweight, interpreted, object-oriented language with first-class functions, and although is best known as the scripting language for Web pages, it is also adopted in many non-browser environments such as NodeJS as well[7]. JavaScript often runs on the client side, handling the web page's behavior during occurrences of events. Along with HTML and CSS, JavaScript is one of the most basic and most important language needed to develop a web application.

JavaScript is the main language used for SSiE's EED function. The server side, client side and view in EED are all written in JavaScript to reduce syntax mistakes due to language mix-ups.

2.1.6 API

Application Programming Interface (API) is a set of routines, protocols and tools for building a software application. Basically, API is a software intermediary that allows two software to interact with each other[8][9]. Some of the more popular APIs include Google Maps API, Twitter API and YouTube API.

When for example, a mobile application is run, it connects to the Internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions and sends it back to the mobile app. The app then interprets said data and presents the information in a readable way – all of this happens via API[8].

For example, assuming a person wants to book flight tickets. Instead of buying directly from the airline company's website, they book them through a travel agency. In this case, the travel agency accesses airline companies' data (available seats, flight number, cabin class, etc.) through their API and responses with information appropriate to the person's request.

2.1.7 Text Simplification

Text Simplification (often abbreviated to TS) is any process which reduces text complexity at different levels (lexical, syntactic, etc.) while maintaining the information contained in it. It is a research area of Natural Language Processing (NLP) whose goal is to maximize text comprehension through simplification of its linguistic structure[10]. A simplified version of a text could benefit low literacy readers, English learners, children, etc. Furthermore, simplifying texts could also improve performance on other NLP tasks, such as information extraction, semantic role labeling and machine translation.

An example of text simplification can be seen in the two examples below[11]:

- | | |
|-------------|--|
| [Complex 1] | <i>"Owls are the order Strigiformes, comprising 200 bird of prey species."</i> |
| [Simple 1] | <i>"An owl is a bird. There are about 200 kinds of owls."</i> |
| [Complex 2] | <i>"Owls hunt mostly small mammals, insects, and other birds though some species specialize in hunting fish."</i> |
| [Simple 2] | <i>"Owls' prey may be birds, large insects (such as crickets), small reptiles (such as lizards) or small mammals (such as mice, rats, and rabbits)."</i> |

From the example above, it can be said that text simplification is performed in various methods. Some notable methods from the example above are; explaining unusual concepts (*“large insects (such as crickets), small reptiles (such as lizards) or small mammals (such as mice, rats, and rabbits).”*), replacing unfamiliar words with more common terms (*“comprising”* → *“There are about”*), changing syntactic structures to a simpler pattern (splitting Complex 1 into two sentences in Simple 1) and removal of unimportant information (*“though some species specialize in hunting fish.”* in Complex 2 is removed).

2.1.8 Machine Learning

Machine Learning (often abbreviated to ML) is a part of Artificial Intelligence (AI) that equips system with the ability to learn and improve from experience and/or data without being explicitly programmed. In ML, machines observe data (examples, direct instructions, reoccurrence of events, experience, etc.) to look for patterns and make better decisions/predictions in the future without the need for human intervention[12]. Some of the more well-known examples of machine learning are self-driving cars and personalized online ads or shopping item recommendations[13]. In SSiE, machine learning is used to help simplify texts into a more comprehensible version.

There are four popular methods of ML. They are: Supervised Learning, Unsupervised Learning, Semi-supervised Learning and Reinforcement Learning. Supervised and Unsupervised Learning are the most widely adopted methods[13]. SSiE implements the Supervised Learning algorithms where machines apply what has been learned using labeled examples in the past to a new dataset to predict future events. The learning algorithm analyzes a known training dataset and produces an inferred function to make predictions about the output values or in SSiE’s Simplifier’s case, a simplified text version.

2.1.9 Python

Python is an interpreted, high-level, general-purpose programming language that works quickly and integrate systems effectively. Python is one of the most common programming languages adopted for machine learning. This is due to its simplicity and consistency, access to great libraries and framework required for AI and ML, flexibility, platform independence and wide community[14]. This is essential as AI projects contrasts

with common software projects for they require different technology stacks and skills as well as the demand for deep research.

2.2 English-English Dictionary

One of the features of SSiE is it has an English-English Dictionary (EED) function. Users can input single words or copy-and paste entire text, and the system in return will output English definitions (including their subsenses) for each word. The definitions are retrieved from Oxford English Dictionary data via their API services. The words (and their definitions) displayed are sorted in the order they were input and is compared to each other to ensure no repetitive words are displayed.

EED is developed solely with JavaScript for both server and client side. It utilizes several npm packages (node modules) to ensure the system operates smoothly. The modules, system operation, etc. will be further explained below.

2.2.1 npm Packages & Node Modules

The npm registry contains packages, many of which are also Node modules, or contain Node modules. A package is a file or directory that is described by a package.json file. A package must contain a package.json file in order to be published to the npm registry. On the other hand, a module is any file or directory in the node_modules directory can be loaded by the NodeJS require() function. To be loaded by the NodeJS require() function, a module must be either; a folder with a package.json file containing a main field, a folder with an index.js file in it, or a JavaScript file. Since modules are not required to have package.json file, not all modules are packages. Only modules that have a package.json file are also packages[15].

EED adopts several packages and modules such as body-parser, ssl-root-cas, request, https, and wink-lemmatizer.

Firstly, the body-parser parses incoming request bodies in a middleware, allowing them to be accessible under the req.body property. The body-parser object exposes various factories to create middlewares. All middlewares will populate the req.body property with the parsed body when the Content-Type request header matches the type option. For example, in Fig. 3, the body-parser module is adopted to retrieve the object 'wordId' from the body of the request made by the client-side.

```

router.post('/entries',function(searchReq, searchRes){
  console.log("wordId: "+searchReq.body.wordId);
  var searchInputVal= searchReq.body.wordId;
  var postRequest = {
    host: "od-api.oxforddictionaries.com",
    port: "443",
    path: "/api/v2/entries/en-gb/" + searchInputVal + '?fields=' + fields + '&strictMatch=' + strictMatch + '&definitions',
    method: "GET",
    headers: {
      'app_id': app_id,
      'app_key': app_key,
    }
  };
});

```

Fig. 3 Example of body-parser module adopted in EED

Next, the ssl-root-cas module solves node's SSL woes when including a custom certificate particularly if a non-standard Root CA is added. Common errors that requires this module to be added are CERT_UNTRUSTED, UNABLE_TO_VERIFY_LEAF_SIGNATURE, and unable to verify the first certificate.

```

var rootCas = require ('ssl-root-cas/latest').inject();
https.globalAgent.options.cs = rootCas;

```

Fig. 4 ssl-root-cas module adopted in EED

The request module designed to be the simplest way to make HTTP calls. It supports HTTPS and follows redirects by default. Furthermore, request emits a 'response' event when a response is received.

```

var request = https.request(postRequest, function(response) {
  var searchData = "";
  response.on( "data", function(data) { searchData += data; } );
  response.on( "end", function(data) {
    var parsed = JSON.parse(searchData);
    if (parsed.error!=null){
      var reqNo = searchReq.body.requestNo;
      var TotalReq = searchReq.body.TotalReq;
      var objErr = { reqNo:reqNo, TotalReq:TotalReq, id:searchInputVal, error:parsed.error };
      searchRes.end(JSON.stringify(objErr));
      console.log(objErr);
    }
    else{
      var sense = parsed.results[0].lexicalEntries[0].entries[0].senses;
      var id = parsed.word;
      var reqNo = searchReq.body.requestNo;
      var TotalReq = searchReq.body.TotalReq;
      var objectt = { reqNo:reqNo, TotalReq:TotalReq, id:id , definition:sense }
      var qer = JSON.stringify(objectt);
      console.log(objectt);
      searchRes.end(qer);
    }
  });
});

request.on('error',function(error){
  console.log('problem with request: ' + error.message);
});

request.write(JSON.stringify(searchInputVal));

```

Fig. 5 request module adopted in EED

HTTPS is the HTTP protocol over TLS/SSL (Transport Layer Security/Secure Sockets Layer). In NodeJS this is implemented as a separate module. Furthermore, `https.request` makes a request to a secure web server.

```
var request = https.request(postRequest, function(response){

    var searchData = "";
    response.on( "data", function(data) { searchData += data;} );
    response.on( "end", function(data) {
        var parsed = JSON.parse(searchData);
        if (parsed.error!=null){
            var reqNo = searchReq.body.requestNo;
            var TotalReq = searchReq.body.TotalReq;
            var objErr = { reqNo:reqNo, TotalReq:TotalReq, id:searchInputVal, error:parsed.error };
            searchRes.end(JSON.stringify(objErr));
            console.log(objErr);
        }
        else{
            var sense = parsed.results[0].lexicalEntries[0].entries[0].senses;
            var id = parsed.word;
            var reqNo = searchReq.body.requestNo;
            var TotalReq = searchReq.body.TotalReq;
            var objectt = { reqNo:reqNo, TotalReq:TotalReq, id:id, definition:sense }
            var qer = JSON.stringify(objectt);
            console.log(objectt);
            searchRes.end(qer);
        }
    });
});
```

Fig. 6 https module adopted in EED

Finally, wink-lemmatizer module obtains the base form or lemmas of nouns, verbs or adjectives[16].

```
// Load wink-Lemmatizer
var lemmatize = require( 'wink-lemmatizer' );

// Lemmatize adjectives
lemmatize.adjective( 'farthest' );
// -> 'far'
lemmatize.adjective( 'coolest' );
// -> 'cool'
lemmatize.adjective( 'easier' );
// -> 'easy'

// Lemmatize nouns
lemmatize.noun( 'knives' );
// -> 'knife'
lemmatize.noun( 'potatoes' );
// -> 'potato'
lemmatize.noun( 'men' );
// -> 'man'

// Lemmatize verbs
lemmatize.verb( 'eaten' );
// -> 'eat'
lemmatize.verb( 'pushes' );
// -> 'push'
lemmatize.verb( 'suggesting' );
// -> 'suggest'
```

Fig. 7 Example implementation of wink-lemmatizer

```

router.post('/lemmas',function (lemmaReq, lemmaRes){
  console.log(lemmaReq.body);
  var arrList = lemmaReq.body;
  var NewarrList = [];
  for (var i=0; i<arrList.length; i++){
    var word = arrList[i].wordId.toLowerCase();
    var wordV = lemmatize.verb(word);
    var wordN = lemmatize.noun(word);
    var wordA = lemmatize.adjective(word);

    if (word!=wordV){
      NewarrList[i] = {wordID:wordV};
      //continue;
    }else if(word!=wordN){
      NewarrList[i] = {wordID:wordN};
      //continue;
    }else if(word!=wordA){
      NewarrList[i] = {wordID:wordA};
      //continue;
    }else{
      NewarrList[i] = {wordID:word};
      //continue;
    }
  }
  //for
  console.log(NewarrList);
  lemmaRes.end(JSON.stringify(NewarrList));
});

```

Fig 8. wink-lemmatizer module adopted in EED

2.2.2 Oxford Dictionaries API

The Oxford Dictionaries API is the self-service toolkit for a world-renowned dictionary data. It is built to equip developers with Oxford's diverse and customizable datasets. Furthermore, the Oxford Dictionaries API provides reliable, up-to-date monolingual and bilingual data in an accessible, scalable environment.

In addition, Oxford University Press© provides a free prototype API services for developers and researchers with a limited request of 1000 calls per month[17]. Once consumers sign up, they will be given an exclusive API ID and key that are required to be written in the header section of the request to gain access to their data. Although there are several other online dictionary API services available, Oxford Dictionaries API was one of the only API services that provides free prototype plan for researchers.

Additionally, consumers are able to specify which field they require access to which helps to avoid response data to be cloudy. For example, EED only consumes the 'definition' part of the response thus irrelevant data/information such as pronunciation and examples are omitted from response. The field is specified in the url when request were being made.



Fig. 9 SSiE's registered account on Oxford Dictionaries database

```
router.post('/entries',function(searchReq, searchRes){
  console.log("wordId: "+searchReq.body.wordId);
  var searchInputVal= searchReq.body.wordId;
  var postRequest = {
    host: "od-api.oxforddictionaries.com",
    port: "443",
    path: "/api/v2/entries/en-gb/" + searchInputVal + '?fields=' + fields + '&strictMatch=' +
    strictMatch + '&definitions',
    method: "GET",
    headers: {
      'app_id': app_id,
      'app_key': app_key,
    }
  };
  console.log('searchInputVal: '+searchInputVal);
  console.log(postRequest);

  var request = https.request(postRequest, function(response){
```

Fig. 10 Specifying field for API consuming

2.2.3 System Operation

Fig. 11 and Fig. 12 describes the system operation of EED. As shown in Fig. 12, EED first puts the user's input into an array, tags each word with their request number and sends the array to the server (at path '/lemmatizer') to lemmatize the words into their base form. For example, the word 'jumping' will be lemmatized to 'jump'. Each word in the array will be run through each lexical category (noun, verb, adjective) in the lemmatizer to retrieve their lemmas regardless of their lexical category.

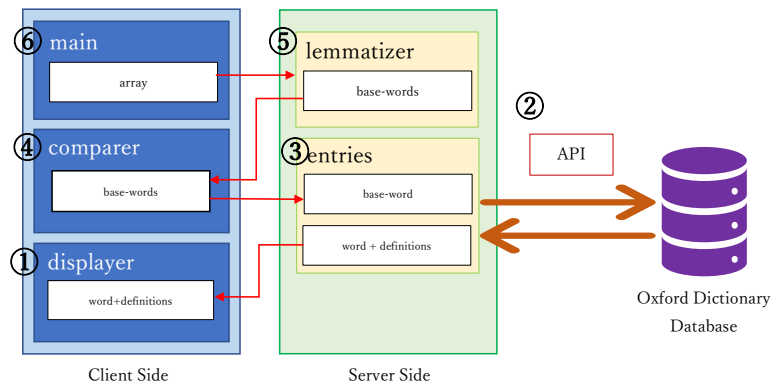


Fig. 11 System Operation Diagram

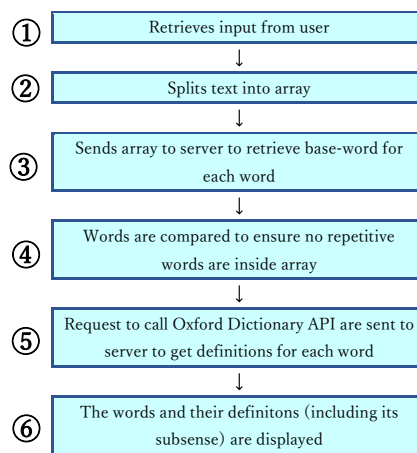


Fig. 12 System Operation Flow

Next, the new array of lemmatized words is sent back to the client side where they are compared with each other to remove repetitive words. The order this is performed is essential as, if the words are compared first before being sent to lemmatizer, then words such as ‘swimming’ and ‘swimmer’ would be considered different by the system resulting in calling the API twice for the word ‘swim’ and its definition to be displayed twice. This needs to be averted to avoid wasteful processing time.

After, each word is tagged a new request number and the newly compared lemmas are sent one by one to the server (at the ‘/entries’ path). Here, Oxford Dictionaries API request are made to retrieve definitions for each word from their database and the words and their respective definitions are sent back to the client side as JSON object.

Back at the client side, as JavaScript is asynchronous, the response will return in an unorderly manner. Thus, the words are put into an array and sorted according to their request number. This would ensure the definition will be displayed in the same order they were input.

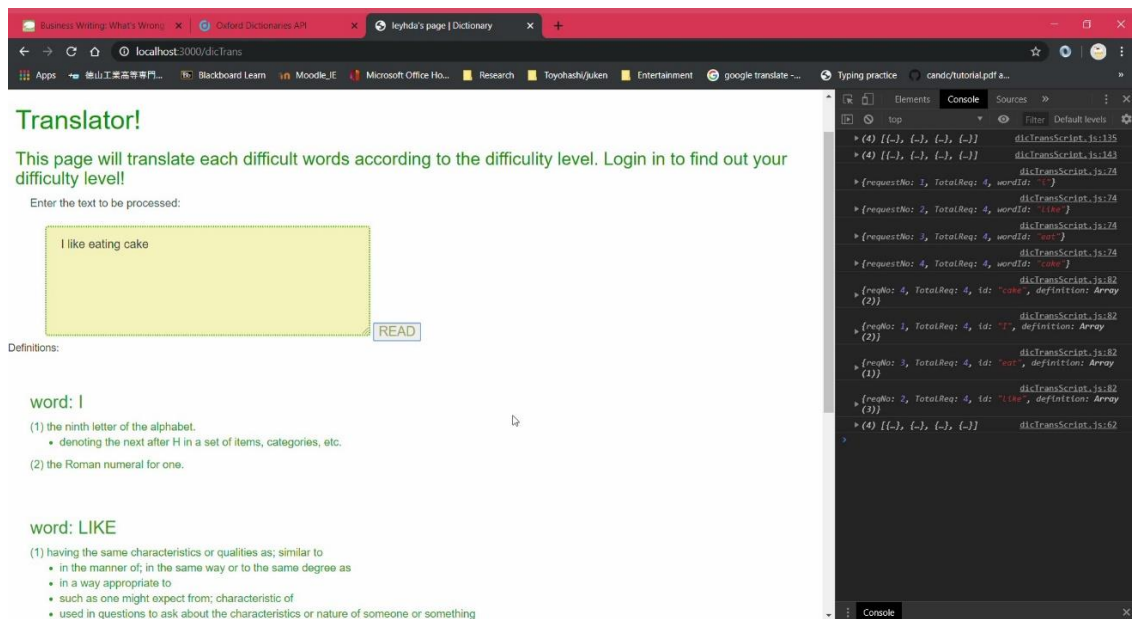


Fig. 12 EED translating example

2.3 Simplifier

The Simplifier takes in user's input of a whole text paragraph and outputs a simpler, easier-to understand version. As mentioned before, this is achieved through machine learning. In other words, Simplifier is essentially a text simplification system.

Simplifier performs text simplification on a semantical level. There are four rewriting operations performed accordingly by Simplifier during text simplification process. They are splitting, deletion, substitution, and reordering.

Some of the requirements for this feature are Moses Statistical Machine Translation (SMT) system, Boxer (a.k.a. C&C Parser), Stanford Toolkit, Giza++, and NLTK Toolkit. Simplifier was essentially written Python programming language.

2.3.1 Moses SMT

Moses is an implementation of the statistical (or data-driven) approach to machine translation (MT)[18]. In statistical machine translation (SMT), translation systems are trained on large quantities of parallel data and monolingual data. From the parallel data, the system learns how to translate small segments while from the monolingual data, the system learns how the target language should look like.

The two main components in Moses are the training pipeline and the decoder. The training pipeline is a collection of tools which take the raw data and turn it into a machine translation model. On the other hand, the decoder is an application which, given a trained machine translation model and a source sentence, will translate the source sentence into the target language.

There are various stages involved in producing a translation system from training data which are implemented as a pipeline. The data needs to be prepared before it is used in training, tokenizing the text and converting tokens to a standard case. Sentence pairs which look to be misaligned are removed. The parallel sentences are then word-aligned. An important part of the translation system is the language model, a statistical model built using monolingual data in the target language and used by the decoder to try to ensure the fluency of the output. Moses relies on external tools for language model building. The final step in the creation of the machine translation system is tuning, where the different statistical models are weighted against each other to produce the best possible translations.

The Moses decoder's job is to find the highest scoring sentence in the target language corresponding to a given source sentence. It is also possible for the decoder to

output a ranked list of translation candidates and also supply various types of information about how it came to its decision.

2.3.2 Boxer (C&C Parser)

Boxer takes a Combinatory Categorical Grammar (CCG) derivation output by the Clark & Curran (C&C) parser and generates a semantic representation. A parser is a program to work out the grammatical structure of sentences, for example, which groups of words go together (as ‘phrases’) and which words are subject or object of a verb.

Boxer implements a first-order fragment of Discourse Representation Theory (DRT) and can generate the box-like structures of DRT known as Discourse Representation Structures (DRSs). DRT is a formal semantic theory backed up with a model theory, and it demonstrates a large coverage of linguistic phenomena. Boxer follows the formal theory closely, introducing discourse referents for noun phrases and events in the domain of a DRS, and their properties in the conditions of a DRS.

Boxer also implements Van der Sandt’s theory of presupposition projection treating proper names and definite descriptions as anaphoric expressions, by binding them to appropriate previously introduced discourse referents, or accommodating on a suitable level of discourse representation[19].

Due to its ability to generate a DRSs, Boxer is implemented by Simplifier to build a parse tree.

```
x0 x1 x2 x3
named(x0,barnum,per)
named(x0,mr,titl)
thing(x1)
worst-case(x2)
scenario(x2)
call(x3)
but(x3)
event(x3)
agent(x3,x0)
patient(x3,x1)
theme(x3,x2)
```

Fig. 13 An example of Boxer output

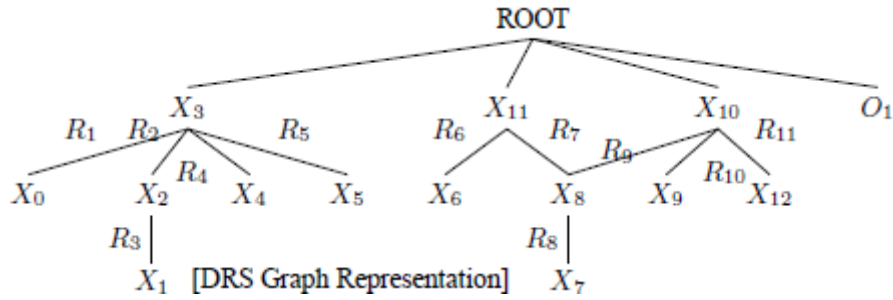


Fig. 14 An example of a parse tree by boxer

2.3.3 Stanford Toolkit

Stanford toolkit is a Part-Of-Speech Tagger (POS Tagger) which is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc., although generally computational applications implements more fine-grained POS tags like 'noun-plural'[20]. In Simplifier, Stanford toolkit is implemented as a tokenizer. Tokenization is the process of demarcating and possibly classifying sections of a string of input characters. The resulting tokens are then passed on to some other form of processing. The process can be considered a sub-task of parsing input.

```
<sentence>
  <word>The</word>
  <word>quick</word>
  <word>brown</word>
  <word>fox</word>
  <word>jumps</word>
  <word>over</word>
  <word>the</word>
  <word>lazy</word>
  <word>dog</word>
</sentence>
```

Fig. 15 Example of tokenizing the sentence “The quick brown fox jumps over the lazy dog”.

2.3.4 Giza++

Giza++ is a toolkit to train word alignment models[21]. Word alignment is a mapping of words between two sentences that have the same meaning in two different languages. For example, as shown in Fig. 16, through word-alignment system, the computer can understand that ‘saw’ has the same meaning as ‘見た’, ‘white bird’ as ‘白い鳥’, and so on.

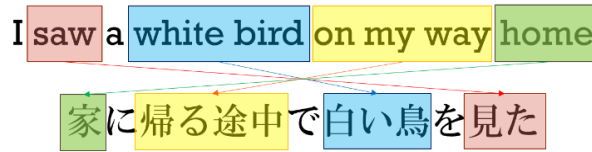


Fig. 16 Example of word alignment

2.3.5 NLTK Toolkit

Natural Language Toolkit (NLTK) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.

2.3.6 System Operation

Simplifier's TS system was meant to be trained on PWKP dataset. As shown in Fig. 17, simplifier first tokenizes sentences with Stanford Toolkit. The tokens are then constructed into a parse tree by Boxer. The parse tree shows the relationship between each word/event in the sentence. It is the most important part of this TS system.

Then, Simplifier calculates the probability of splitting to occur. With that, sentence would be split into two or more to make it more comprehensible. Afterwards, Simplifier deletes unimportant phrases (such as adjectives, adverbs, etc.). Finally, it calculates the probability of performing substitution (exchange difficult words with more common ones) and reordering and implements them accordingly. This is performed by utilizing knowledge about alignment and translation probabilities after being trained on a dataset.

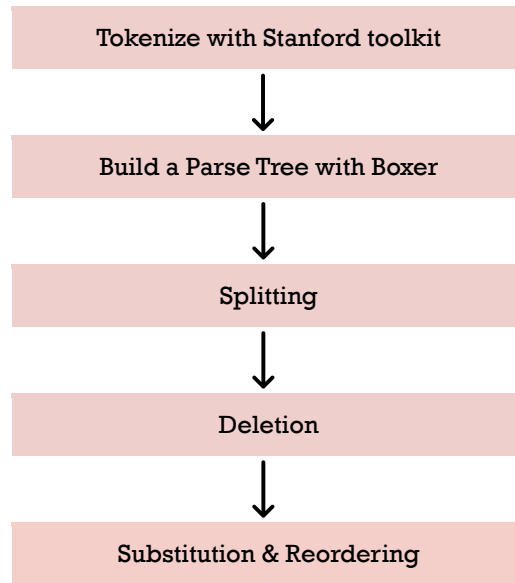


Fig. 17 Simplifier System Operation Flow

```

teyhdaaa@DESKTOP-PD6R1MO:~$ ls
Main-Failed-Sentence      extract_wikipedia_corpora_boxer_test.py
PWKP_108016               extract_wikipedia_corpora_boxer_training.py
PWKP_108016.tar.gz        sample
PWKP_108016_shortversion  install
PWKP_108016_shortversion.tokenized-boxer.xml  learningbyreading
python3.7.3               nlpdecoder
Sentence-Simplification-AS-4.0  nlp_data
Simple-Failed-Sentence     sample-models
boost_1_69_0              sample-models.tgz
boost_1_69_0.tar.bz2      simple_sent_shortversion
boxer-output.xml          stanford-postagger-2017-10-16
candc                      stanford-toolkit
candc-1.00.tgz            simpl-devel
complex_sent_shortversion  ubuntustuff
teyhdaaa@DESKTOP-PD6R1MO:~$
  
```

Fig. 18 Tools for Simplifier installed on Ubuntu

```
ieyhdaaa@DESKTOP-PD6RIMO: ~
GNU nano 2.9.3 PWKP_108016_shortversion

The traditional etymology is from the Latin aperire, "to open," in allusion to its being the season when trees and flow$
The name April comes from that Latin word aperire which means "to open".
This probably refers to growing plants in spring.

April is the fourth month of the year in the Gregorian Calendar, and one of four months with a length of 30 days.
April is the fourth month of the year with 30 days.

The birthstone of April is the diamond, and the birth flower is typically listed as either the Daisy or the Sweet Pea.
April's flower is the Sweet Pea and its birthstone is the Diamond.

August is the eighth month of the year in the Gregorian Calendar and one of seven Gregorian months with the length of 3$
August is the eighth month of the year.
It has 31 days.

This month was originally named Sextilis in Latin, because it was the sixth month in the ancient Roman calendar, which $
This month was first called Sextilis in Latin, because it was the sixth month in the old Roman calendar.
The Roman calendar began in March about 735 BC with Romulus.

It became the eighth month either when January and February were added to the beginning of the year by King Numa Pompili$
It was the eighth month when January or February were added to the start of the year by King Numa Pompilius about 700 B$
Or, when those two months were moved from the end to the beginning of the year by the decemvirs about 450 BC (Roman wri$

August's flower is the gladiolus or poppy, and its birthstone is the peridot.
August's flower is the Gladioli with the birthstone being Peridot.

The first and broadest sense of art is the one that has remained closest to the older Latin meaning, which roughly tran$
The first and broadest sense of "art" means "arrangement" or "to arrange."

Sculptures, cave paintings, rock paintings, and petroglyphs from the Upper Paleolithic dating to roughly 40000 years ag$
There are sculptures, cave paintings, rock paintings and petroglyphs dating from the Upper Paleolithic era, about 40000$

In the east, Islamic art's rejection of iconography led to emphasis on geometric patterns, calligraphy, and architectur$
Islamic art includes geometric patterns, Islamic calligraphy, and architecture.

India and Tibet saw emphasis on painted sculptures and dance with religious painting borrowing many conventions from sc$
In India and Tibet, painted sculptures, dance, and religious painting were done.

China saw many art forms flourish, jade carving, bronzework, pottery (including the stunning terracotta army of Emperor$
In China, arts included jade carving, bronzework, pottery, poetry, calligraphy, music, painting, drama, and fiction.

Chinese styles vary greatly from era to era and are traditionally named after the ruling dynasty.
There are many Chinese artistic styles, which are usually named after the ruling dynasty.

10 December 2007, Spain is divided into 17 autonomous communities.
Spain is divided in 17 parts called autonomous communities.

G Get Help  O Write Out  W Where Is  K Cut Text  J Justify  C Cur Pos  M-U Undo  M-A Mark Text
X Exit      R Read File  Y Replace  U Uncut Text  T To Spell  G Go To Line  M-E Redo  M-C Copy Text
```

Fig. 19 PWKP dataset sample

3. Evaluation

3.1 English-English Dictionary

EED was successfully achieved. Any single word or whole text paragraphs can be input, and the system will be able to return definitions for each word. The words and their definitions are also displayed in order they were input.

3.1.1 Response Time

Three words with different definitions length were sent to the server 10 times in a row to observe the system's response time. The words are 'do' (very long definition), 'cake' (medium-length definition), and 'pulchritudinous' (very short definition). Based on the graph in Fig. 20 and Table 1 & 2, it can be seen that words with longer definitions causes more delay compare to words with shorter definitions.

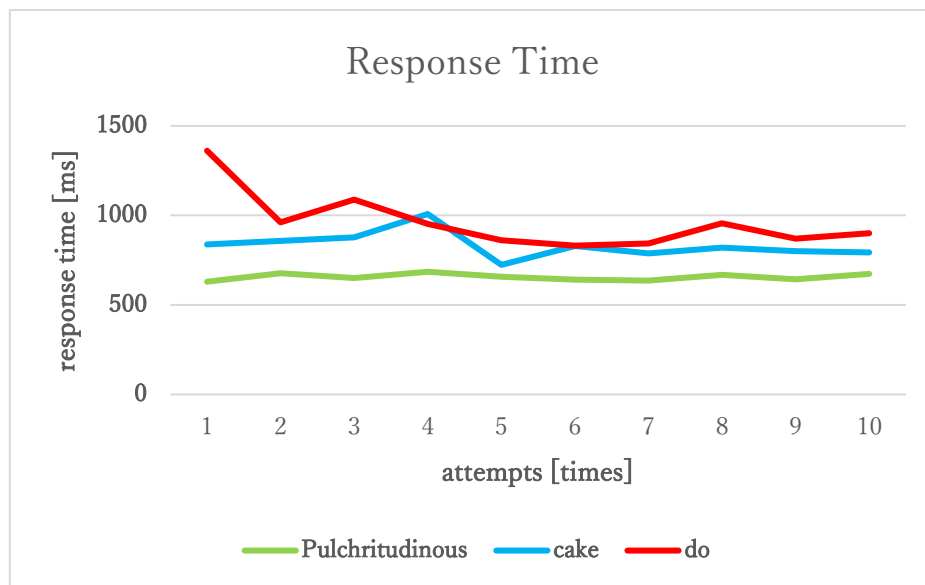


Fig. 20 Response time of EED

Table 1 Time taken for each response to return to client side

<i>words</i> <i>Attempt</i>	Pulchritudinous	cake	do
1	629.855	837.251	1361.255
2	676.697	857.589	960.796
3	651.035	878.171	1088.293
4	685.159	1007.76	952.938
5	658.01	723.758	861.132
6	641.118	828.781	831.138
7	635.952	787.818	842.813
8	667.409	820.324	955.363
9	642.285	800.024	869.971
10	673.302	792.961	900.134
<i>Average</i>	656.0822	833.4437	962.3833

Table 2 Average Response Time

Words	Average Response Time [ms]
Pulchritudinous	656.0822
Cake	833.4437
Do	962.3833

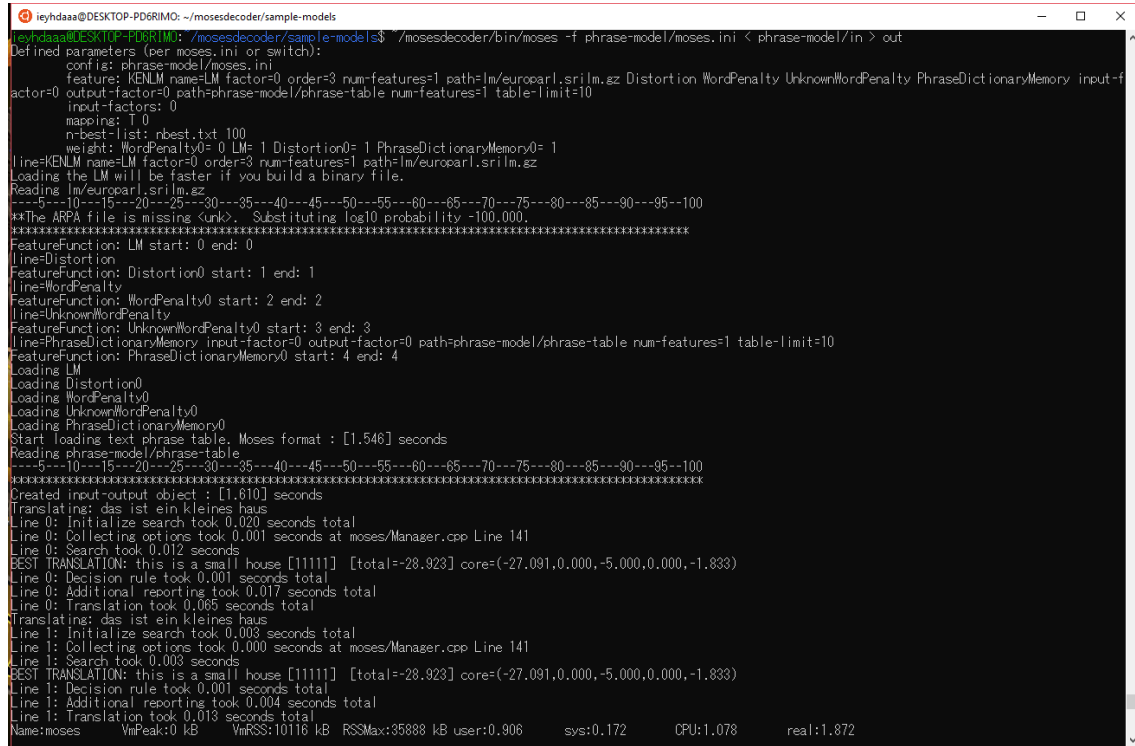
3.1.2 Asynchronous JavaScript

JavaScript is asynchronous, thus, response returns to the client side in unordered manner. In order to wait for all the responses to return and then sort them, a small delay regrettably occurs.

Before, SSiE displays the words and definitions in order they were returned from the server. Considering as this may cause comprehension difficulty, it was improved. However, in exchange for that, the delay to display definitions becomes longer.

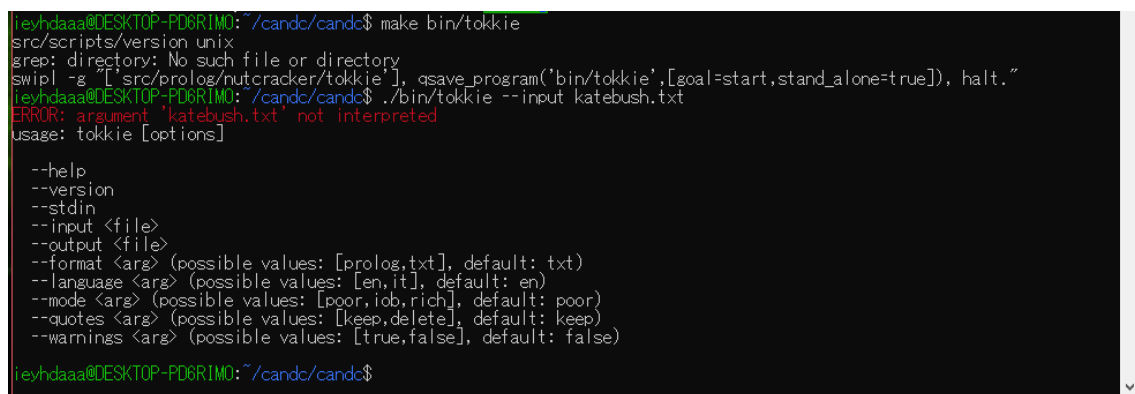
3.2 Simplifier

The Moses SMT system was successfully installed and implemented. However, problem arise when attempting to execute Boxer. As Boxer creates the parser tree, and the parser tree is the most important tool to execute text simplification, the system could not be finished.



```
ieyhdaaa@DESKTOP-PD6R1MO: ~/mosesdecoder/sample-models
ieyhdaaa@DESKTOP-PD6R1MO: ~/mosesdecoder/sample-models$ ~/mosesdecoder/bin/moses -f phrase-model/moses.ini < phrase-model/in > out
Defined parameters (per moses.ini or switch):
  config: phrase-model/moses.ini
  feature: KENLM name=LM factor=0 order=3 num-features=1 path=lm/europarl.srlm.gz Distortion WordPenalty UnknownWordPenalty PhraseDictionaryMemory input=f
  actor=0 output-factor=0 path=phrase-model/phrase-table num-features=1 table-limit=10
  input-factors: 0
  mapping: T 0
  n-best-list: nbest.txt 100
  weight: WordPenalty0= 0 LM= 1 Distortion0= 1 PhraseDictionaryMemory0= 1
line=KENLM name=LM factor=0 order=3 num-features=1 path=lm/europarl.srlm.gz
Loading the LM will be faster if you build a binary file.
Reading lm/europarl.srlm.gz
---5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95---100
**The ARPA file is missing <unk>. Substituting log10 probability -100,000.
*****
FeatureFunction: LM start: 0 end: 0
line=Distortion
FeatureFunction: Distortion0 start: 1 end: 1
line=WordPenalty
FeatureFunction: WordPenalty0 start: 2 end: 2
line=UnknownWordPenalty
FeatureFunction: UnknownWordPenalty0 start: 3 end: 3
line=PhraseDictionaryMemory input-factor=0 output-factor=0 path=phrase-model/phrase-table num-features=1 table-limit=10
FeatureFunction: PhraseDictionaryMemory0 start: 4 end: 4
Loading LM
Loading Distortion0
Loading WordPenalty0
Loading UnknownWordPenalty0
Loading PhraseDictionaryMemory0
Start loading text phrase table. Moses format : [1.546] seconds
Reading phrase-model/phrase-table
---5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95---100
*****
Created input-output object : [1.610] seconds
Translating: das ist ein kleines haus
Line 0: Initialize search took 0.020 seconds total
Line 0: Collecting options took 0.001 seconds at moses/Manager.cpp Line 141
Line 0: Search took 0.012 seconds
BEST TRANSLATION: this is a small house [11111] [total=-28.923] core=(-27.091,0.000,-5.000,0.000,-1.833)
Line 0: Decision rule took 0.001 seconds total
Line 0: Additional reporting took 0.017 seconds total
Line 0: Translation took 0.065 seconds total
Translating: das ist ein kleines haus
Line 1: Initialize search took 0.003 seconds total
Line 1: Collecting options took 0.000 seconds at moses/Manager.cpp Line 141
Line 1: Search took 0.003 seconds
BEST TRANSLATION: this is a small house [11111] [total=-28.923] core=(-27.091,0.000,-5.000,0.000,-1.833)
Line 1: Decision rule took 0.001 seconds total
Line 1: Additional reporting took 0.004 seconds total
Line 1: Translation took 0.013 seconds total
Name:moses VmPeak:0 kB VmRSS:10116 kB RSSMax:35888 kB user:0.906 sys:0.172 CPU:1.078 real:1.872
```

Fig. 21 Moses successful implementation



```
ieyhdaaa@DESKTOP-PD6R1MO: ~/candc/candc$ make bin/tokkie
src/scripts/version unix
grep: directory: No such file or directory
swipl -g "[src/prolog/nutcracker/tokkie]', qsave_program('bin/tokkie',[goal=start,stand_alone=true]), halt."
ieyhdaaa@DESKTOP-PD6R1MO: ~/candc/candc$ ./bin/tokkie --input katebush.txt
ERROR: argument 'katebush.txt' not interpreted
usage: tokkie [options]

--help
--version
--stdin
--input <file>
--output <file>
--format <arg> (possible values: [prolog,txt], default: txt)
--language <arg> (possible values: [en,it], default: en)
--mode <arg> (possible values: [poor,iob,rich], default: poor)
--quotes <arg> (possible values: [keep,delete], default: keep)
--warnings <arg> (possible values: [true,false], default: false)

ieyhdaaa@DESKTOP-PD6R1MO: ~/candc/candc$
```

Fig. 22 Failed implementation of Boxer

4. Conclusion

English is widely used as a communication tool internationally. Certain countries that uses English as their second language can be said to be able to master English easier as they are regularly put in an English environment such as studying certain subjects in English. I believe by being put in an English environment regularly, countries that have trouble mastering English will be able to acquire it easier.

Specialty Studies in English (SSiE) is a web system to help non-native English speakers to study specialty subjects in English without the need to translate into their native language. SSiE has two functions, English-English Dictionary translator (EED) and Simplifier. For EED, users can input single word or entire text paragraph and the system will output English definitions for each word. Simplifier, on the other hand, outputs a simpler, easier-to-understand version of the text input by users. Simplifier's system is still under development.

In the future, for EED, I plan to develop the system to be able to output definitions catered to users' English mastery level. Another senior of my research lab has already created this system thus, I plan on integrating her system to SSiE.

Furthermore, I plan to improve the system response time in two ways. First, by moving the comparer function to server side to reduce data transmitting time as SSiE's current model sends data to server twice (once to '/lemmatizer' and another to '/entries'). Secondly, I will improve the displayer function and utilize JavaScript's asynchronous feature to shorten delay of time.

As for Simplifier, I plan to either try implementing boxer another way, or try to develop a TS model which does not use boxer at all.

5. Acknowledgements

I would like to thank the Associate Professor, Prof. Hideaki Yanagisawa for all of his guidance that had helped me throughout this research. Furthermore, I would also like to thank my fellow classmates that had helped me with the programming and debugging of this system, especially Miss Tsukushi Hayai and Miss Mirano Sumida. Also, I am grateful to my parents and friends for supporting me throughout this whole process. Thank you.

6. References

- [1] Rimal, A. (2019). Developing a web Application on NodeJS and MongoDB using ES6 and Beyond (Bachelor's Thesis, Metropolia university of Applied Sciences, Helsinki, Finland) pp. 7 - 9. Retrieved from: https://www.theseus.fi/bitstream/handle/10024/159951/Rimal_Aashis.pdf?sequence
- [2] NodeJS. About NodeJS. Available from: <https://nodejs.org/en/about/> [Accessed 18 February 2020]
- [3] Express. Express: Node.js Web Application Framework. Available from: <https://expressjs.com/> [Accessed 18 February 2020]
- [4] Express. Using Template Engines with Express. Available from: <https://expressjs.com/en/guide/using-template-engines.html> [Accessed 18 February 2020]
- [5] EJS. EJS: What is EJS. Available from: <https://ejs.co/> [Accessed 18 February 2020]
- [6] Bos B. W3C Cascading Style Sheet Homepage: What is CSS? Available from: <https://www.w3.org/Style/CSS/Overview.en.html> [Last Updated: 14 February 2020, Accessed: 18 February 2020]
- [7] JavaScript. MDN Web Docs: About JavaScript. Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript [Last Updated: 4 December 2019, Accessed 18 February 2020]
- [8] MuleSoft. What is an API? (Application Programming Interface) Available from: <https://www.mulesoft.com/resources/api/what-is-an-api> [Accessed 18 February 2020]
- [9] Vangie Beal. API (Application Program Interface) Available from: <https://www.webopedia.com/TERM/A/API.html> [Accessed 18 February 2020]
- [10] Irina Temnikova. (April 2012) **Text Complexity and Text Simplification in the Crisis Management domain** (unpublished doctoral dissertation, University of Wolverhampton, Wolverhampton, United Kingdom) pp.66.
- [11] sebastianruder. NLP-Progress: Simplification. Available from: <http://nlpprogress.com/english/simplification.html> [Accessed 18 February 2020]
- [12] Expert System. (March 2017) What is Machine Learning? A Definition. Available from: <https://expertsystem.com/machine-learning-definition/> [Accessed 18 February 2020]
- [13] SAS. Machine Learning What Is It and Why It Matters. Available from: https://www.sas.com/en_us/insights/analytics/machine-learning.html [Accessed 18 February 2020]

- [14]Angela Beklemysheva. SteelKiwi: “Why Use python for AI and Machine Learning?”. Available from: <https://steelkiwi.com/blog/python-for-ai-and-machine-learning/> [Accessed 18 February 2020]
- [15]npm. npm: About Packages and Modules. Available from: <https://docs.npmjs.com/about-packages-and-modules> [Accessed 18 February 2020]
- [16]npm. npm: wink-lemmatizer. Available from: <https://www.npmjs.com/package/wink-lemmatizer> [Accessed 19 February 2020]
- [17]Oxford University Press©. Oxford Dictionaries API: Pricing. Available from: <https://developer.oxforddictionaries.com/> [Accessed 19 February 2020]
- [18]Moses statistical machine translation system. Moses: Overview. Available from: <http://www.statmt.org/moses/?n=Moses.Overview> [Last Updated: 13 August 2013, Accessed: 19 February 2020]
- [19]Curran, James & Clark, Stephen & Bos, Johan. (2007). Linguistically Motivated Large-Scale NLP with C&C and Boxer. Proceedings of the ACL-07 Demo and Poster Sessions, Prague, Czech Republic. 10.3115/1557769.1557781.
- [20]Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. (2003). Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003, pp. 252-259.
- [21]Masato Hagiwara. Using GIZA++ to Obtain Word Alignment Between Bilingual Sentences. Available from: <https://masatohagiwara.net/using-giza-to-obtain-word-alignment-between-bilingual-sentences.html> [Accessed 19 February 2020]
- [22]Narayan, S., & Gardent, C. (2014, June). Hybrid simplification using deep semantics and machine translation. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, Maryland, USA, June 23-25 2014, pp. 435-445

7. Program List

→ app.js (server)

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var defSearchRouter = require('./routes/defSearch');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/defSearch', defSearchRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
```

```

    res.locals.message = err.message;
    res.locals.error = req.app.get('env') === 'development' ? err : {};

    // render the error page
    res.status(err.status || 500);
    res.render('error');
  });

module.exports = app;

→ defSearch.js (server (in 'routes' directory))
var express = require('express');
var router = express.Router();

//for use of oxford dictionary api
var bodyParser = require('body-parser');
var https = require("https");
var request = require('request');
var lemmatize = require('wink-lemmatizer');

var rootCas = require ('ssl-root-cas/latest').inject();
https.globalAgent.options.cs = rootCas;

var app_id = "c471c6c7"; // insert your APP Id
var app_key = "7dba9e30abd5656dc4be42fec3094b47"; // insert your APP Key
var fields = "definitions";
var strictMatch = "false";

router.use(bodyParser.urlencoded({ extended: true }));
router.use(bodyParser.json());

router.post('/entries',function(searchReq, searchRes){
    console.log("wordId: "+searchReq.body.wordId);
    var searchInputVal= searchReq.body.wordId;
    var postRequest = {
        host: "od-api.oxforddictionaries.com",

```

```

        port: "443",
        path: "/api/v2/entries/en-gb/" + searchInputVal + "?fields=" + fields +
        '&strictMatch=' + strictMatch + '&definitions',
        method: "GET",
        headers: {
            'app_id': app_id,
            'app_key': app_key,
        }
    };
    console.log('searchInputVal: '+searchInputVal);
    console.log(postRequest);

    var request = https.request(postRequest, function(response){

        var searchData = "";
        response.on( "data", function(data) { searchData += data;} );
        response.on( "end", function(data) {
            var parsed = JSON.parse(searchData);
            if (parsed.error!=null){
                var reqNo = searchReq.body.requestNo;
                var TotalReq = searchReq.body.TotalReq;
                var objErr = { reqNo:reqNo, TotalReq:TotalReq,
id:searchInputVal, error:parsed.error };
                searchRes.end(JSON.stringify(objErr));
                console.log(objErr);
            }
            else{
                var
                sense
                =
                parsed.results[0].lexicalEntries[0].entries[0].senses;
                var id =  parsed.word;
                var reqNo = searchReq.body.requestNo;
                var TotalReq = searchReq.body.TotalReq;
                var objectt = { reqNo:reqNo, TotalReq:TotalReq,
id:id , definition:sense }

                var qer = JSON.stringify(objectt);
                console.log(objectt);
            }
        });
    });

```



```

        searchRes.end(qer);
    }
});

});

request.on('error',function(error){
    console.log('problem with request: ' + error.message);
});

request.write(JSON.stringify(searchInputVal));

});

router.post('/lemmas',function (lemmaReq, lemmaRes){
    console.log(lemmaReq.body);
    var arrList = lemmaReq.body;
    var NewarrList =[];
    for (var i=0; i<arrList.length; i++){
        var word = arrList[i].wordId.toLowerCase();
        var wordV = lemmatize.verb(word);
        var wordN = lemmatize.noun(word);
        var wordA = lemmatize.adjective(word);

        if (word!==wordV){
            NewarrList[i] = {wordID:wordV};
        }else if(word!==wordN){
            NewarrList[i] = {wordID:wordN};
        }else if(word!==wordA){
            NewarrList[i] = {wordID:wordA};
        }else{
            NewarrList[i] = {wordID:word};
        }
    }
}

console.log(NewarrList);
lemmaRes.end(JSON.stringify(NewarrList));

```

```
});
```

```
module.exports = router;
```

→ index.js (server (in 'routes' directory))

```
var express = require('express');
```

```
var router = express.Router();
```

```
/* GET home page. */
```

```
router.get('/', function(req, res, next) {
```

```
  res.render('index', { page: 'Home', menuId: 'home' });
```

```
});
```

```
router.get('/dicTrans', function(req, res, next) {
```

```
  res.render('dicTrans', {page:'Dictionary', menuId:'dictionary'});
```

```
});
```

```
router.get('/summarizer', function(req, res, next) {
```

```
  res.render('summarizer', {page:'Summarizer', menuId:'summarizer'});
```

```
});
```

```
module.exports = router;
```

→ users.js (server (in 'routes' directory))

```
var express = require('express');
```

```
var router = express.Router();
```

```
/* GET users listing. */
```

```
router.get('/', function(req, res, next) {
```

```
  res.send('respond with a resource');
```

```
});
```

```
module.exports = router;
```

```

→ index.ejs (view)
<!DOCTYPE html>
<html lang="en">
<head>
  <% include partials/head %>
  <link rel="stylesheet" type="text/css" href="/stylesheets/indexStyle.css">
  <script src="/javascripts/indexScript.js"></script>
</head>
<body>
  <% include partials/menu %>

  <div class="welcome">Welcome!! <br> Let's learn together</div>
  <p class="intro"> Here, we encourage our users to learn and understand their course
  subjects in English rather than translating texts into their first-language to understand
  materials!</p>
  <h2 class="login">Login Form</h2>

  <!--login-->
  <button          onclick="document.getElementById('id01').style.display='block'"
  style="width:auto; margin-left:15px;">Login</button>
  <div id="id01" class="modal">

    <form class="modal-content animate" action="/action_page.php">
      <div class="imgcontainer">
        <span          onclick="document.getElementById('id01').style.display='none'"
class="close" title="Close Modal">&times;</span>
        
      </div>

      <div class="container">
        <label for="uname"><b>Username</b></label>
        <input type="text" placeholder="Enter Username" name="uname" required>

        <label for="psw"><b>Password</b></label>
        <input type="password" placeholder="Enter Password" name="psw" required>

```

```

        <button type="submit">Login</button>
        <label>
            <input type="checkbox" checked="checked" name="remember"> Remember
me
        </label>
    </div>

```

```

    <div class="container" style="background-color:#f1f1f1">
        <button
                                                    type="button"
onclick="document.getElementById('id01').style.display='none'"
class="cancelbtn">Cancel</button>
        <span class="psw">Forgot <a href="#">password?</a></span>
    </div>
</form>
</div>

```

```

</body>
<% include partials/script %>
</html>

```

→ dicTrans.ejs (view)

```

<!DOCTYPE html>
<html lang="en">
<head>
    <% include partials/head %>
    <link rel="stylesheet" type="text/css" href="/stylesheets/dicTransStyle.css">
    <script src="/javascripts/dicTransScript.js"></script>
</head>
<body>
    <% include partials/menu %>

```

```

<h1 class="title">Translator!</h1>
<h3 class="explain">This page will translate each difficult words according to the
difficulty level. Login in to find out your difficulty level!</h3>

```

```

<p class="start">Enter the text to be processed: </p>

```

```
<textarea rows="100" cols="50" class="text_input" id="textinput"></textarea>
<button          class="ctrl-standard          typ-subhed          fx-bubbleUp"
onclick="getString()">READ</button>
```

```
<p>Definitions:</p>
<div id="translate"></div>
```

```
</body>
<% include partials/script %>
</html>
```

→ summarizer.ejs (view)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <% include partials/head %>
  <link rel="stylesheet" type="text/css" href="/stylesheets/summarizerStyle.css">
  <script src="/javascripts/dicTransScript.js"></script>
</head>
<body>
  <% include partials/menu %>
```

```
<h1 class="title">Summarizer!</h1>
<h3 class="explain">Summarize your text by reading and finding the main points so you
can skip the unnecessary information and grasp the gist of your text to understand it easier
</h3>
<p class="start">Enter the text to be processed: </p>
<textarea rows="6" cols="100" class="text_input" id="textinput"></textarea>
<input type="button" onclick="getString()" value="Read">
<p id="poutput"></p>
```

```
</body>
<% include partials/script %>
</html>
```

→ error.ejs (view)

```
<h1><%= message %></h1>
<h2><%= error.status %></h2>
<pre><%= error.stack %></pre>
```

→ head.ejs (view)

```
<title>Ieyhda's page | <%= page %></title>
<meta charset="utf-8">
<meta name="active-menu" content="<%= menuId %>">
<meta name="viewport" content="width=device-width, initial-scale=1">
<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.4.0.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<link rel="stylesheet" href="/stylesheets/style.css">
```

→ menu.ejs (view)

```
<nav class="navbar navbar-default">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target="#myNavbar">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">LOGO</a>
    </div>
    <div class="collapse navbar-collapse" id="myNavbar">
      <ul class="nav navbar-nav navbar-right">
        <li id="home"><a href="/">HOME</a></li>
        <li id="login"><a href="/users">USERS</a></li>
        <li id="dictionary"><a href="/dicTrans">DICTIONARY</a></li>
        <li id="summarizer"><a href="/summarizer">SUMMARIZER</a></li>
      </ul>
    </div>
  </div>
```

```
</div>
</nav>
```

→ script.ejs (view)

```
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<script src="./javascripts/menu.js"></script>
```

→ indexScript.js (client (in 'javascripts' directory))

//add "!!!" after welcome divider

```
$(document).ready(function() {
    $(".welcome").append("!!!");
});
```

// Get the modal

```
var modal = document.getElementById('id01');
```

// When the user clicks anywhere outside of the modal, close it

```
window.onclick = function(event) {
    if (event.target == modal) {
        modal.style.display = "none";
    }
}
```

→ dicTransScript.js (client (in 'javascripts' directory))

```
function displayer(stringg){ //display data
    //getting wordID
    var wordx = stringg.id;
    var wordUX = wordx.toUpperCase();
    var translateP = document.getElementById("translate");
    var headW = document.createElement("H3");
    var newText = "word: " + wordUX;
    var headWord = document.createTextNode(newText);
    headW.appendChild(headWord);
    translateP.appendChild(headW);
```

```

        if (stringg.error!=null){
            var translateP = document.getElementById("translate");
            translateP.innerHTML = translate.innerHTML + "ERROR : " +
stringgg.error + '<br>Please try entering root word<br>';
        }
        else {
            var wordNo = stringg.reqNo; //setting word number

            //getting definitions
            var sayy = stringg.definition;
            var num = 1;
            for (x in sayy){
                var deff = sayy[x].definitions;
                var subb = sayy[x].subsenses;
                translateP.innerHTML = translate.innerHTML + "(" + num +
") " + deff + "<br>";
                if (subb!=null){
                    for(m in subb){
                        var newUL =
document.createElement("UL");
                        var mynewUL = "ULnewID" + wordNo +
num;
                        newUL.setAttribute('id', mynewUL);
                        translateP.appendChild(newUL);

                        var newLI = document.createElement("LI");
                        var SubText =
document.createTextNode(subb[m].definitions);
                        newLI.appendChild(SubText);

                        document.getElementById(mynewUL).appendChild(newLI);
                    }
                }
                num += 1;
            }
            translateP.innerHTML = translate.innerHTML + "<br><br>";

```



```

    }
}

var respArr = [];

function RespManager(APIresp, arrLength){ //putting in new array
    respArr.push(APIresp);

    if (respArr.length==arrLength){ //display in order
        var cnt = 1;
        for (j=0; j<arrLength; j++){
            for(i=0; i<arrLength; i++){
                if (respArr[i].reqNo==cnt){
                    displayer(respArr[i]);
                    break;
                }
            }
            cnt += 1;
        }
    }
}

function getDef(newObj){ //getting definitions from server
    var requestNo = 1;

    for (var j=0; j<newObj.length; j++){
        var data = {requestNo: requestNo, TotalReq: newObj.length, wordId:
newObj[j].wordID}; //format data to pass to server as JSON
        requestNo += 1;

        console.log(data);
        var dataa = JSON.stringify(data);

        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function(){
            if(this.readyState == 4 && this.status == 200){

```

```

        var objectt = this.responseText;
        var stringg = JSON.parse(objectt);
        console.log(stringg);
        var TotalNo = stringg.TotalReq;
        RespManager(stringg,TotalNo);
        //displayer(stringg);
    };
};

xhttp.open("POST",'/defSearch/entries',true);
xhttp.setRequestHeader("Content-type", "application/json");
xhttp.send(dataa);
}

}

function comparer(LemmArray){ //to compare for the same words
    var flag = 0;
    var NewLemmaArray = [];
    var cnt = 0;
    for (var i = 0; i <= LemmArray.length-1; i++) {
        //checking for same words
        for (var j = 0; j <= i-1; j++){
            if(LemmArray[i].wordID==LemmArray[j].wordID){
                flag = 1;
                break;
            }
            else {
                continue;
            }
        }
    }

    //if not the same words
    if(flag == 0){
        if(LemmArray[i]!=null | LemmArray[i]!=""){
            NewLemmaArray[cnt]=LemmArray[i];

```

```

        cnt += 1;
    }
}

flag = 0;
} //for i
return NewLemmaArray;
}

function getString(){
    var deet = document.getElementById("textinput").value;
    var words = deet.split(/[ .,:;!~,'"&|()<>{}¥[¥]¥r¥n/¥¥]+/);
    var data=[]; //initializing an empty array

    for (var j=0; j<words.length; j++){
        if (words[j]!=null | words[j]!=""){
            data[j] = {wordId: words[j]};
        }
    }
    //format data to pass to server as JSON

    console.log(data);
    var dataa = JSON.stringify(data);

    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function(){
        if(this.readyState == 4 && this.status == 200){
            var lemmaResp = this.responseText;
            var lemmaObj = JSON.parse(lemmaResp);
            console.log(lemmaObj);
            var NewArray = comparer(lemmaObj);
            getDef(NewArray);
        }
    };

    xhttp.open("POST","/defSearch/lemmas",true);
    xhttp.setRequestHeader("Content-type", "application/json");

```

```

        xhttp.send(dataa);
    }

```

→ summarizerScript.js (client (in 'javascripts' directory))

```

function getString(){
    var input = document.getElementById("textinput").value;
    Algorithmia.client("simt3OyuMtiW4UMZenMG9UIf1XB1")
    .algo("nlp/Summarizer/0.1.8?timeout=300") // timeout is optional
    .pipe(input)
    .then(function(output) {
        outputP = document.getElementById("poutput");
        outputP.innerHTML = outputP.innerHTML + output;
    });
}

```

→ menu.js (client (in 'javascripts' directory))

```

$(document).ready(function(){
    var element = $('meta[name="active-menu"]').attr('content');
    $('#'+ element).addClass('active');
});

```

→ indexStyle.css (client (in 'stylesheets' directory))

```

.welcome {
    color:#5C5CD6;
    font-size:36px;
    font-family:cursive;
    text-align:center;
    padding:20px;
}

```

```

.intro{
    color:#3333CC;
    font-size:15px;
    font-family:Arial;
    text-align:center;
}

```

```

.login{
    color:green;
    font-size:20px;
    font-family:Arial;
    text-align:left;
    margin-left:15px;
}
/*-----menu-----*/
body {margin:0;}

.icon-bar {
    width: 100%;
    background-color: #555;
    overflow: auto;
}

.icon-bar a {
    float: left;
    width: 25%;
    text-align: center;
    padding: 12px 0;
    transition: all 0.3s ease;
    color: white;
    font-size: 36px;
}

.icon-bar a:hover {
    background-color: #000;
}

.active {
    background-color: #D65C85;
}
/*-----end of menu-----*/

```

```

/*-----login-----*/
body {font-family: Arial, Helvetica, sans-serif;}

/* Full-width input fields */
input[type=text], input[type=password] {
    width: 100%;
    padding: 12px 20px;
    margin: 8px 0;
    display: inline-block;
    border: 1px solid #ccc;
    box-sizing: border-box;
}

/* Set a style for all buttons */
button {
    background-color: #4CAF50;
    color: white;
    padding: 14px 20px;
    margin: 8px 0;
    border: none;
    cursor: pointer;
    width: 100%;
}

button:hover {
    opacity: 0.8;
}

/* Extra styles for the cancel button */
.cancelbtn {
    width: auto;
    padding: 10px 18px;
    background-color: #f44336;
}

/* Center the image and position the close button */

```

```

.imgcontainer {
    text-align: center;
    margin: 24px 0 12px 0;
    position: relative;
}

img.avatar {
    width: 40%;
    border-radius: 50%;
}

.container {
    padding: 16px;
}

span.psw {
    float: right;
    padding-top: 16px;
}

/* The Modal (background) */
.modal {
    display: none; /* Hidden by default */
    position: fixed; /* Stay in place */
    z-index: 1; /* Sit on top */
    left: 0;
    top: 0;
    width: 100%; /* Full width */
    height: 100%; /* Full height */
    overflow: auto; /* Enable scroll if needed */
    background-color: rgb(0,0,0); /* Fallback color */
    background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
    padding-top: 60px;
}

/* Modal Content/Box */

```

```

.modal-content {
    background-color: #fefefe;
    margin: 5% auto 15% auto; /* 5% from the top, 15% from the bottom and centered */
    border: 1px solid #888;
    width: 30%; /* Could be more or less, depending on screen size */
}

/* The Close Button (x) */
.close {
    position: absolute;
    right: 25px;
    top: 0;
    color: #000;
    font-size: 35px;
    font-weight: bold;
}

.close:hover,
.close:focus {
    color: red;
    cursor: pointer;
}

/* Add Zoom Animation */
.animate {
    -webkit-animation: animatezoom 0.6s;
    animation: animatezoom 0.6s
}

@-webkit-keyframes animatezoom {
    from {-webkit-transform: scale(0)}
    to {-webkit-transform: scale(1)}
}

@keyframes animatezoom {
    from {transform: scale(0)}

```



```

    to {transform: scale(1)}
}

/* Change styles for span and cancel button on extra small screens */
@media screen and (max-width: 300px) {
    span.psw {
        display: block;
        float: none;
    }
    .cancelbtn {
        width: 100%;
    }
}
/*-----end of login-----*/

```

→ dicTransStyle.css (client (in 'stylesheets' directory))

```

.start {
    color:#2F4F4F;
    font-size:15px;
    font-family:Arial;
    margin-left:30px;
    text-align:left;
}

.title {
    margin-left: 10px;
    color: Green;
    font-family:Arial;
}

.explain {
    margin-left: 10px;
    color: Green;
    font-family:Arial;
}
/*-----read button-----*/

```

```

body::before {
    display: none;
}
.typ-subhed {
    font-family: 'Oswald', sans-serif;
    font-size: 18px;
    line-height: 20px;
    letter-spacing: 0;
}
.ctrl-standard.fx-bubbleUp {
    color: #9fa255;
    border-color: #9fa255;
}
.ctrl-standard.fx-bubbleUp::after {
    top: auto;
    border-radius: $right-val1 $left-val1 0 0/$right-val2 $left-al2 0 0;
    background: #9fa255;
}
[class*="ctrl-"] {
    padding: 5px 10px;
    border-radius: 2px;
    border: 1px solid #231f20;
}
[class*="ctrl-"][class*="fx-"]:hover,
[class*="ctrl-"][class*="fx-"].active {
    color: #231f20;
}
[class*="fx-"],
[class*="fx-"]:hover {
    transition: color 0.5s ease-in-out;
}
[class*="fx-"]:not(.fx-dyna) {
    position: relative;
    z-index: 1;
    transition-delay: 0.2s !important;
    overflow: hidden;
}

```

```

    }
    [class*="fx-"]:not(.fx-dyna)::after,
[class*="fx-"]:not(.fx-dyna)::before {
    content: "";
    display: block;
    position: absolute;
    z-index: -1;
}
[class*="fx-bubble"]::after {
    transition: height 0.5s ease-in-out;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
    height: 0;
    width: 100%;
}
[class*="fx-bubble"]:hover::after,
[class*="fx-bubble"].active::after {
    transition: height 0.5s ease-in-out;
    height: 300%;
}
[class*="fx-bubble"]:disabled:hover::after {
    height: 0;
}
/*-----end read button-----*/
#textinput {
    width: 40%;
    height: 150px;
    padding: 12px 20px;
    box-sizing: border-box;
    border: 2px dotted #6DB72C;
    border-radius: 4px;
    background-color: #f6f0bb;
    font-size: 16px;
    margin-top: 10px;
}

```

```

        margin-left: 50px;
    }

    #translate {
        margin-top: 50px;
        margin-left: 30px;
        color: ForestGreen;
        font-size: 15px;
        font-family: Arial;
        text-align: left;
    }

```

→ summarizerStyle.css (client (in 'stylesheets' directory))

```

.start {
    color: #1FB28D;
    font-size: 15px;
    font-family: Arial;
    margin-left: 30px;
    text-align: left;
}

```

```

.title {
    margin-left: 10px;
    color: #1F8DB2;
    font-family: Arial;
}

```

```

.explain {
    margin-left: 10px;
    color: #1F8DB2;
    font-family: Arial;
}

```

```

#textinput {
    width: 40%;
    height: 150px;
}

```

```

padding: 12px 20px;
box-sizing: border-box;
border: 2px solid #20B2AA;
border-radius: 4px;
background-color: #ffe6e9;
font-size: 16px;
margin-top: 10px;
margin-left: 50px;
}

```

```

#translate {
margin-top: 50px;
margin-left: 30px;
color: #20B2AA;
font-size: 15px;
font-family: Arial;
text-align: left;
}

```

→ style.css (client (in 'stylesheets' directory))

```

.bg-3 {
background-color: #ffffff;
color: #555555;
}
.container-fluid {
padding-top: 70px;
padding-bottom: 70px;
}
.navbar {
padding-top: 15px;
padding-bottom: 15px;
border: 0;
border-radius: 0;
margin-bottom: 0;
font-size: 12px;
letter-spacing: 5px;
}

```

```
}  
.navbar-nav li a:hover {  
    color: #1abc9c !important;  
}  
.active>a {  
    color: #1abc9c !important;;  
}
```