Appendix - Code

Main.py

```
from etl import main_input_processing
import numpy as np
from flask import Flask,render_template,request,jsonify, flash,
url_for, redirect, session
import sqlite3
from nlp import create_recommendation
from etl import eliminate_plural_trivial_words
from bs4 import BeautifulSoup
import requests
import pickle
import os

def get_backend_result(user_inputs):
    clean_keyword_pool = main_input_processing(user_inputs)
    print("user inputs ... "+str(user_inputs))
    print("main processing: ..." +str(clean_keyword_pool))
    c2_relevance,similar_word =
create_recommendation(clean_keyword_pool, user_inputs)
    print("create recommendation: ..." +str(c2_relevance)+"
"+str(similar_word))
    return c2_relevance,similar_word,user_inputs,clean_keyword_pool

def check_user_input(input1,input2,input3):
    error = []
    error_msg = ""
    user_inputs = [input1,input2,input3]
    user_inputs = eliminate_plural_trivial_words(user_inputs)
    if len(user_inputs)<=2:
        is_redirect = True
        print("must enter 3 keywords")
        error_msg = "must enter 3 keywords"
        error.append(error_msg)
        return is_redirect,error,user_inputs

    for input in user_inputs:
```

```
        if input == "":
            is_redirect = True
            error_msg = "Input contains a stopword. Please try again."
            error.append(error_msg)
            return is_redirect,error,user_inputs


    result = {} #dictionary
    for each_input in user_inputs:
        iserror = 0
        base_url = 'https://podcasts.google.com/search/'
        search_url = base_url + each_input
        resp = requests.get(search_url)
        soup = BeautifulSoup(resp.text, 'lxml') #utilizes google
podcast api to search for podcast results
        div_list = soup.find_all('div', class_="O9KIXe") #check if no
podcast found using class property as web-scraping tool
        if len(div_list)!=0: #meaning that within class, there is a
line: "no podcast found". So, the input is invalid
            iserror = 1
        result[each_input] = iserror

    is_redirect = False
    error = []
    for key in result.keys():
        if result[key] == 1:
            error_msg = f"Input {key} is invalid. Please try again."
            error.append(error_msg)
    if error_msg !="":
        is_redirect = True
    return is_redirect, error, user_inputs

if __name__ == '__main__': #removes redundancies of rerunning models
etc. increases efficiency

    app = Flask(__name__)
    app.secret_key = "super secret key"

    @app.route('/')
    def form():
        template_name = 'form.html'
```

```
        conn = sqlite3.connect('data/KEYWORD_MAP.db')
        conn.row_factory = sqlite3.Row  # index-based and
case-insensitive name-based access to columns; converts plain tuple to
more useful object
        print("Connected to the database successfully")
        # Create a cursor for interacting with the database
        cur = conn.cursor()
        cur.execute('CREATE TABLE IF NOT EXISTS KEYWORD(user_input,
keywords)')
        conn.commit()
        # Execute a query to fetch data from the 'KEYWORD' table
        cur.execute('SELECT user_input FROM KEYWORD')
        input_records = cur.fetchall()
        input_records = [input_record[0] for input_record in
input_records]
        return render_template(template_name,
input_records=input_records) #renders the frontend

    @app.route('/previous_inputs', methods = ['POST']) # Route to
display data from the 'KEYWORD' table
    def previous_inputs(): #query the data, grab the data and pass it
on to form.html template
        template_name = 'previous_inputs.html'
        # Connect to the SQLite database
        conn = sqlite3.connect('data/KEYWORD_MAP.db')
        conn.row_factory = sqlite3.Row  # index-based and
case-insensitive name-based access to columns;
        #converts plain tuple to more useful object
        print("Connected to the database successfully")

        # Create a cursor for interacting with the database
        cur = conn.cursor()
        cur.execute('CREATE TABLE IF NOT EXISTS KEYWORD(user_input,
keywords)')
        conn.commit()
        # Execute a query to fetch data from the 'KEYWORD' table
        cur.execute('SELECT user_input, keywords FROM KEYWORD')
        # Fetch all records from the query result and convert to a
list of dictionaries
        records = cur.fetchall()
```

```
        keywords = [dict(user_input=record[0], keywords=record[1]) for
record in records]
        #This makes it easier to work with the data in a more
structured way

        # Close the database connection
        conn.close()

        # Pass the data to the HTML template
        return render_template(template_name, keywords=keywords)

    @app.route('/query_user_input', methods=['POST'])
    def query_user_input():
        request_data = request.get_json()  # Get JSON data from the
request body
        if 'user_query' in request_data:
            search_input = request_data['user_query']
            conn = sqlite3.connect('data/KEYWORD_MAP.db')
            cur = conn.cursor()
            cur.execute('SELECT keywords FROM KEYWORD WHERE user_input
= ?', (search_input,))
            query_keywords = cur.fetchone()
            query_keywords = query_keywords[0]
            query_keywords = query_keywords.split(",")
            if query_keywords is not None:
                session['keyword'] = query_keywords
                search_input = ''.join(search_input)
                session['searchQuery'] = search_input
                session['route'] = 1
                #keywordVector = word_to_vector(query_keywords)
                return "success"
                #return render_template('embedding_projector.html',
query_keywords = query_keywords)
            else:
                print("No data found for the given input")
                return jsonify({'error': 'No data found for the given
input'}), 404  # Return a 404 Not Found status code
        else:
            print("user_query not in request_data")
            return jsonify({'error': 'No user_query provided'}), 400
# Return a 400 Bad Request status code
```

```
    @app.route('/input_validation', methods = ['POST'])
    def input_validation():
        form_data = request.form #requesting the input data from form
        input_1, input_2, input_3 =
form_data['Input1'],form_data['Input2'],form_data['Input3']
        is_redirect, error, user_inputs =
check_user_input(input_1,input_2,input_3)
        if is_redirect == False and len(user_inputs)==3:
            input_1 = user_inputs[0]
            input_2 = user_inputs[1]
            input_3 = user_inputs[2]
            session['user_input'] = [input_1, input_2, input_3]
#session: datalog for individual users acts as a dictionary
            return redirect(url_for('loading')) #redirects to loading
page
        else:
            for error_msg in error:
                flash(f'{error_msg}')
            #return render_template(template_name) #re enter data
            return redirect(url_for('form'))

    @app.route('/loading')
    def loading():
        template_name = 'loading.html'
        return render_template(template_name)

    @app.route('/backend')
    def backend():
        user_inputs = session.get('user_input') #get input from
session
        c2_relevance,similar_word,user_inputs,clean_keyword_pool =
get_backend_result(user_inputs) #backend processing
        data = {
            "c2_relevance": c2_relevance,
            "similar_word": similar_word,
            "clean_keyword_pool": clean_keyword_pool,
            "user_inputs": user_inputs
        }
        session['result'] = pickle.dumps(data)
        return "Success"
```

```
    @app.route('/result')
    def result():
        template_name = 'result.html'
        result_data = session['result']
        session['route'] = 0
        if result_data:
            # Unpickle the data to get the variables
            try:
                loaded_data = pickle.loads(result_data)
                c2_relevance = loaded_data.get('c2_relevance')
                similar_word = loaded_data.get('similar_word')
                clean_keyword_pool =
loaded_data.get('clean_keyword_pool')
                user_inputs = loaded_data.get('user_inputs')
                # Now you can use these variables

                #flatten list of list into list
                keyword_pool = []
                for i in clean_keyword_pool:
                    keyword_pool.extend(i)
                clean_keyword_pool = keyword_pool

                # Generate the URL for the embedding_projector route
with clean_keyword_pool as a query parameter
                word_cloud_url = url_for('word_cloud')
                session['keyword'] = clean_keyword_pool
                inputs = str(user_inputs[0])+" ,
"+str(user_inputs[1])+" , "+str(user_inputs[2])
                flash(inputs)
                flash(similar_word)
                flash("{:.2%}".format(c2_relevance))
                similar_word = str(similar_word)
                base_url = 'https://podcasts.google.com/search/'
                search_url = base_url + similar_word
                flash(search_url)
                return
render_template(template_name,c2_relevance=c2_relevance,
                                    similar_word=similar_word,
search_url=search_url,bool=[0,1,2,0],
                                    user_inputs=user_inputs,
```

```
                                    word_cloud_url=word_cloud_url)
            except (pickle.UnpicklingError, TypeError, ValueError):
                print("Pickle not pickling...")
                # Handle exceptions during unpickling
                pass

    @app.route('/word_cloud')
    def word_cloud():
        template_name = 'word_cloud.html'
        clean_keyword_pool = session['keyword']
        if session['route'] == 1:
            search_input = session['searchQuery']
            flash(search_input)
            route = 1
        else:
            route = 0
        return render_template(template_name,

keywordsForCloud=clean_keyword_pool,route = route)
    app.run(debug=True)
```

etl.py

```
from bs4 import BeautifulSoup
import pandas as pd
import podcastparser
import urllib.request
from keybert import KeyBERT
import requests,sqlite3,re
import pickle
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
nltk.download('omw-1.4')

def get_stop_words():
    with open('data/Stopwords.pickle', 'rb') as handle:
        stopwords = pickle.load(handle)
```

```
    return stopwords

def eliminate_plural_trivial_words(word_list):
    # Define a list of trivial words
    trivial_words = get_stop_words()
    lemmatizer = nltk.stem.WordNetLemmatizer()
    singular_words = []
    for word in word_list:
        if word!="":
            singular_words.append(lemmatizer.lemmatize(word.strip()))
    #singular_words = [lemmatizer.lemmatize(word.strip()) for word in
word_list]
    # Remove trivial words
    cleaned_words = []
    for word in singular_words:
        if word.lower() not in trivial_words and word.lower()!='':
            cleaned_words.append(word.lower())
        else:
            cleaned_words.append("")
    print("singular words: ...
"+str(cleaned_words)+str(len(cleaned_words)))
    return cleaned_words

def KeywordExtractor(user_input):
    #reading data from database
    conn = sqlite3.connect('data/KEYWORD_MAP.db')  #connecting to a
database
    cursor = conn.cursor()
    cursor.execute('CREATE TABLE IF NOT EXISTS KEYWORD(user_input,
keywords)')  #creating a database table
    conn.commit()

    df_result = pd.read_sql('SELECT user_input, keywords FROM
KEYWORD', conn)

    #comparing user_input from database with new user_input
    for index in range(df_result['user_input'].count()):
        if(df_result['user_input'][index]==user_input):
            return(user_input,df_result['keywords'][index])

    #if same then return keywords previously found
```

```
    #otherwise, go through webscraping process to find keywords
    base_url = 'https://podcasts.google.com/search/'
    search_url = base_url + user_input
    resp = requests.get(search_url)
    soup = BeautifulSoup(resp.text, 'lxml')
    #utilizes google podcast api to search for podcast results

    podcast_urls = []
    results = soup.find_all('a', {'role': 'listitem'}) #find the
podcasts items inside of the soup content
    domain_google_podcast = 'https://podcasts.google.com/'
    for result in results:
        podcast_url_part = result.get('href')[2:] #get the links of
each podcast item
        podcast_urls.append(domain_google_podcast+podcast_url_part)

    #getting homepage url
    homepage_urls = []
    for i in podcast_urls:
        resp_home = requests.get(i)
        soup_home = BeautifulSoup(resp_home.text, 'lxml')
        home_class = soup_home.find_all('div', {'class': 'Uqdiuc'})
#access the item within homepage class
        for div in home_class:
            homepage_url_part = div.a['href'] #access the homepage URL
of each podcast

homepage_urls.append(domain_google_podcast+homepage_url_part)

    #check if homepage_urls in list are the same (first elimination of
redundant elements)
    new_homepage_urls = list(set(homepage_urls))

    descriptions = {}
    for pc_url in new_homepage_urls:
        google_podcast_url = pc_url
        url_getrssfeed = 'https://getrssfeed.com'
        headers = {'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0
Safari/537.36'}
```

```
        #to get podcast homepage rss url
        r = requests.post(url_getrssfeed,
data={"url":google_podcast_url}, headers=headers)
        soup_getrssafterpost = BeautifulSoup(r.text, 'lxml')
        try:
            rss_url  = soup_getrssafterpost.find('div', {'class':
'mt-4'}).a['href']
        except:
            print(f"Cannot retrieve rss feed from this
{google_podcast_url}")
            continue
        try:
            parsed = podcastparser.parse(rss_url,
urllib.request.urlopen(rss_url))
            #get descriptions for each rss feed episode
            description = ''
            for i in range (len(parsed['episodes'])):
                description = description +
parsed['episodes'][i]['description']

            descriptions[parsed['title']] = description
        except:
            print(f"ERROR in pasring rss url: {rss_url}")

    total_keywords = []
    for i in descriptions.keys():
        kw_model = KeyBERT() #model using tone, word frequency, etc to
find keywords from text
        keywords = kw_model.extract_keywords(descriptions[i])
        total_keywords.append(keywords)

    #making list of list (total_keywords) into string for storing in
database
    total_keywords_flat = []
    for item in total_keywords:
        for item2 in item:
            if item2[0] not in total_keywords_flat:
                total_keywords_flat.append(item2[0])

    total_keywords_string = ",".join(total_keywords_flat)
```

```
    #expanding user inputs to create keyword pool using BERT from
descriptions of relevant podcasts
    d = {'user_input': [user_input],
'keywords':[total_keywords_string]}
    df_result = pd.DataFrame(data=d)
    df_result.to_sql('KEYWORD', conn, if_exists='append', index=False)
#put into database
    return(user_input,total_keywords_string)


def main_input_processing(input_results):
    keyword_pool = []

    print(f"Receive user inputs: {input_results}")

    for input_result in input_results:
        print(f"Start processing user input: {input_result}...")
        user_input, total_keywords_string =
KeywordExtractor(input_result)
        total_keywords_list = total_keywords_string.split(',') #making
string into list of keywords seperated by comma
        keyword_pool.append(total_keywords_list) #list of list of
keywords from each input

    print(f"Finished processing user inputs: {input_results}")

    clean_keyword_pool = []
    for each_keyword in keyword_pool:
        cleaned_words = eliminate_plural_trivial_words(each_keyword)
        if len(cleaned_words) == 0:
            continue
        new_cleaned_words = list(set(cleaned_words))
        clean_keyword_pool.append(new_cleaned_words)

    return clean_keyword_pool
```

nlp.py

```
from gensim import models
import numpy as np
import os
file_path = os.path.join(os.getcwd(), 'data', 'glove.6B.300d.txt')
model = models.KeyedVectors.load_word2vec_format(file_path,
binary=False, no_header=True)

def word_to_vector(keyword_pool):
    each_keyword_vector_pool = []
    if keyword_pool is None:
        return []  # Return an empty list if keyword_pool is None
    for each_keyword in keyword_pool:
        try:
            # 'model' is your pre-trained Word2Vec model
            vector = model[each_keyword]
            each_keyword_vector_pool.append(vector)
        except KeyError:
            # Handle the case where the keyword is not in the model's
vocabulary
            continue
    return each_keyword_vector_pool

def get_centroid_1(keyword_pool):
    keyword_pool_flat = []
    for i in keyword_pool:
        keyword_pool_flat.extend(i) #makes list flat by concatenating
list of keywords from each input together

    vector_pool = []
    for i in keyword_pool_flat:
        try:
            vector_pool.append(model[i]) #make into 1x300x(number of
keywords) array
        except:
            continue
    #mean
    vector_pool = np.array(vector_pool, dtype=object)
    print(vector_pool.shape)
    centroid_1 = vector_pool.mean(axis=0)
```

```
    distance=0
    for vector in vector_pool:
        distance += np.sqrt(sum((vector-centroid_1)**2))
    avg_distance = distance/len(vector_pool) #average distance
    print(centroid_1.shape,avg_distance,vector_pool.shape)
    return(centroid_1,avg_distance,vector_pool)

def get_centroid_2(keyword_pool):
    total_keyword_vector_pool = []
    pre_centroid_arr = []
    for each_keyword_pool in range(len(keyword_pool)): #loop through
keyword pool from each homepage
        each_keyword_vector_pool = []
        for each_keyword in keyword_pool[each_keyword_pool]: #loop
through keyword in each keyword pool
            try:
                each_keyword_vector_pool.append(model[each_keyword])
#list of list
            except:
                continue
        each_keyword_vector_pool = np.array(each_keyword_vector_pool,
dtype=object) #keyword vectors for each input in a numpy array
        total_keyword_vector_pool.append(each_keyword_vector_pool)
#array stored in a nested list for all 3 inputs
    total_keyword_vector_pool = np.array(total_keyword_vector_pool,
dtype=object)
    for keyword_vector_pool in total_keyword_vector_pool:
        pre_centroid_2 = keyword_vector_pool.mean(axis=0) #centroid
for each keyword pool of each input
        #print(f"pre_centroid_2: {pre_centroid_2.shape}")
        pre_centroid_arr.append(pre_centroid_2) #store in an array
    print(len(pre_centroid_arr))
    pre_centroid_arr = np.array(pre_centroid_arr, dtype=object)
    centroid_2 = pre_centroid_arr.mean(axis=0)
    distance = 0
    for arr in range(len(pre_centroid_arr)):
        distance +=
np.sqrt(sum((pre_centroid_arr[arr]-centroid_2)**2))
    avg_distance = distance/len(pre_centroid_arr)
    print(centroid_2.shape,avg_distance,pre_centroid_arr.shape)
    return(centroid_2,avg_distance,pre_centroid_arr)
```

```
def create_recommendation(clean_keyword_pool,user_inputs):
    print("centroid 1: ")
    c1,distance1,vector_pool1=get_centroid_1(clean_keyword_pool)
    print("centroid 2: ")
    c2,distance2,vector_pool2=get_centroid_2(clean_keyword_pool)

    C1_min = 5.697671953672131
    C1_max = 6.826386969517547
    C2_min = 0.9865800996310438
    C2_max = 4.104674641199845

    def C1_min_max_normalisation(C_dis):
        return 1-((C_dis-C1_min)/(C1_max-C1_min)) #normalisation to
0-1, the larger the more relevant

    def C2_min_max_normalisation(C_dis):
        return 1-((C_dis-C2_min)/(C2_max-C2_min))

    c1_relevance,c2_relevance =
C1_min_max_normalisation(distance1),C2_min_max_normalisation(distance2
)

    final_centroid = c2.reshape(300)
    centroid_input1 = model.similar_by_vector(final_centroid)
#matching centroid vector with list of similar words
    centroid_input1 = np.array(centroid_input1)

    #check if centroid similar word includes user input
    for user_input in user_inputs:
        index = 0
        for (similar_word,_) in centroid_input1: #checking if similar
word includes user input word
            if(user_input in similar_word):
                centroid_input1[index][0]=-1 #eliminate if similar
word does
            index+=1 #next input check

    for (similar_word,_) in centroid_input1:
        if similar_word !='-1':
            print(similar_word) #recommended word
```

```
            break

    return c2_relevance,similar_word
```

templates/form.html

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="static/form_display.css">
        <link rel="stylesheet" href="static/main.css">
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.js"></s
cript>
        <script
src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8.16/jquery-ui.j
s"></script>
        <link
href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.16/themes/ui-l
ightness/jquery-ui.css"
            rel="stylesheet" type="text/css" />
    </head>
    <body>
        <div class="button-container">
            <a href="http://127.0.0.1:5000"
class="home-button">Home</a>
            <form action="/previous_inputs" method="POST">
                <div class="home-button">
                    <input type="submit" value="Previous Inputs"/>
                </div>
            </form>
        </div>
        <hr>

        <div class="login-box">
            <h2>Give me your keywords</h2>
            <form action="/input_validation" method = "POST">
                <div class="user-box">
```

```
            <input type = "text" name = "Input1" id="tags1"/>
            <label>Input1</label>
        </div>
        <div class="user-box">
            <input type = "text" name = "Input2" id="tags2"/>
            <label>Input2</label>
        </div>
        <div class="user-box">
            <input type = "text" name = "Input3" id="tags3"/>
            <label>Input3</label>
        </div>
        <div class="home-button">
            <input type = "submit" value = "Submit"/>
        </div>
    </form>
</div>

<div class="flash">
    {% with messages = get_flashed_messages() %}
        {% for message in messages %}
            {{ message}}
        {% endfor %}
    {% endwith %}
</div>

<script>
    $( function() {
      var availableTags1 = [
          {% for input_record in input_records %}
              "{{input_record}}",
          {% endfor %}
      ];
      $( "#tags1" ).autocomplete({
        source: availableTags1
      });
    } );

    $( function() {
      var availableTags2 = [
          {% for input_record in input_records %}
              "{{input_record}}",
```

```
                {% endfor %}
            ];
            $( "#tags2" ).autocomplete({
              source: availableTags2
            });
          } );

          $( function() {
            var availableTags3 = [
                {% for input_record in input_records %}
                    "{{input_record}}",
                {% endfor %}
            ];
            $( "#tags3" ).autocomplete({
              source: availableTags3
            });
          } );
        </script>

    </body>
</html>
```

templates/loading.html

```
<head>
    <link rel="stylesheet" href="static/loading.css">

    <link rel="stylesheet" href="static/main.css">
    <script>
        function navigate() {
            window.location.href = 'result';  // Redirects user to the
/map route when 'create_map' is finished
        }
        fetch('backend').then(navigate); // Performing 'backend' and
then calls navigate() function, declared above
    </script>
```

```
</head>

<body>
    <div class="sk-wander center">
        <div class="sk-wander-cube"></div>
        <div class="sk-wander-cube"></div>
        <div class="sk-wander-cube"></div>
        <div class="sk-wander-cube"></div>
    </div>
</body>
```

templates/result.html

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="static/result_style.css">
        <link rel="stylesheet" href="static/main.css">
        <link rel="stylesheet" href="static/word_cloud.css">
    </head>
    <body>
        <a href="http://127.0.0.1:5000" class="home-button">Home</a>
        <div>
            <h1>Podcast Recommendation</h1>
            <hr>
        </div>
        {% with messages = get_flashed_messages() %}
            {% if messages %}
                <div class="container"> <!-- wrap both the ul and div
class inside another class so that they horizontally align -->
                    <ul class="flash"> <!-- flash class displaying
relevance, inputs, etc.-->
                        {% for message in messages %}
                            <li>
                                {% if bool[loop.index0] == 2 %}
<!--represents the zero-based index of the current iteration -->
                                    Relevance of input: {{ message }}
                                {% elif bool[loop.index0] == 1 %}
                                    Recommended Keyword: {{message}}
                                {% else %}
                                    {{message}}
```

```
                                {% endif %}
                            </li>
                        {% endfor %}
                    </ul>
                    <div class="results">
                        <p>Click the link below to view search
results:</p>
                        <a href="{{ search_url }}"
target="_blank">View Search Results on Google Podcast</a>
                    </div>
                    <!-- <div class="iframe-container"> div class for
embedded frame
                        <iframe src="{{ search_url }}" title="Google
Podcast" style = "width: 200%; height: 700px;"></iframe>
                    </div> -->
                    <div class="wordcloud">
                        <iframe src="{{ word_cloud_url }}"
title="wordcloud" style = "width: 210%; height: 500px;"
scrolling="yes"></iframe>
                    </div>
                </div>
            {% endif %}
        {% endwith %}
    </body>
</html>
```

templates/previous_inputs.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Previous Inputs</title>
  <link href="https://unpkg.com/gridjs/dist/theme/mermaid.min.css"
rel="stylesheet">
  <link rel="stylesheet" href="static/main.css">
  <link rel="stylesheet" href="static/button.css">
</head>
<body>
```

```
  <a href="http://127.0.0.1:5000" class="home-button">Home</a>
  <div>
    <h1>Previous Inputs</h1>
    <hr>
    <div class="database_vis">
        <div id="database"></div>
    </div>
  </div>
  <script src="https://unpkg.com/gridjs/dist/gridjs.umd.js"></script>
  <script>
    const userData = {{ keywords | tojson | safe }};
    new gridjs.Grid({
      columns: [
        { id: 'user_input', name: 'User Input' },
        { id: 'keywords', name: 'Keywords' },
      ],
      data: userData,
      search: {
        selector: (cell, rowIndex, cellIndex) => [0,
1].includes(cellIndex) ? cell : null,
      },
      sort: true,
      pagination: true,
    }).render(document.getElementById('database'));

    // Function to perform a POST request when Enter key is pressed
    function getUserInputOnSearch() {
      var user_query = document.querySelector('.gridjs-search-input');
      var user_query = user_query.value;

      function navigate() {
            window.location.href = 'word_cloud';  // Redirects user to
the /map route when 'create_map' is finished
        }
        // Send the user input to the server via a POST request
        fetch('/query_user_input', {
            method: 'POST',
            headers: {
            'Content-Type': 'application/json',
            },
            body: JSON.stringify({ user_query: user_query }),
```

```
        }).then(navigate);
    }
    const searchInput = document.getElementById('searchInput');

    const searchButton_1 =
document.querySelector('.gridjs-search-input');

    searchButton_1.addEventListener('keyup', function(event) {
      if (event.key === 'Enter') {
        getUserInputOnSearch();
      }
    });

    searchInput.addEventListener('keyup', function(event) {
      if (event.key === 'Enter') {
        getUserInputOnSearch();
      }
    });
  </script>
</body>
</html>
```

templates/word_cloud.html

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="static/word_cloud.css">
        <link rel="stylesheet" href="static/main.css">
    </head>
    <body>
        {% if route == 1 %}
            {% with message = get_flashed_messages() %}
                <a href="http://127.0.0.1:5000"
class="home-button">Home</a>
                <h1>Search Query</h1>
                <hr>
                <div class="searchQuery">
```

```
                    Displayed Keywords for: {{ message }}
                </div>
            {% endwith %}
        {% endif %}
        <script
src="https://cdn.jsdelivr.net/npm/TagCloud@2.2.0/dist/TagCloud.min.js"
></script>
        <span class="contents"></span>
        <script>
            const myTags = JSON.parse('{{ keywordsForCloud | tojson |
safe }}');
            var tagCloud = TagCloud('.contents', myTags, {
                radius: 270,
                // animation speed
                maxSpeed: "fast",
                initSpeed: "fast",
                direction: 135,
                left: 0,
                // interact with cursor movement
                keep: true,
            });
            // To change the color of text randomly
            var colors = [
            "#F9F9F9",
            "#FFE0AC",
            "#FFACB7",
            "#FCFFA6",
            "#FFEDED",
            ];
            var random_color = colors[Math.floor(Math.random() *
colors.length)];
            document.querySelector(".contents").style.color =
random_color;
        </script>
    </body>
</html>
```

static/main.css

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins&display=swap');

html {
    height: 150%;
    width: 100%;
}
body {
    margin-top:5px;
    font-family: 'Poppins', sans-serif;
    background: linear-gradient(#141e30, #243b55);
}
h1 {
  color: white;
  padding-top: 2rem;
}

.home-button {
  position: relative;
  top: 20px;
  left: 10px;
  color: whitesmoke;
  display: inline-block;
  padding: 10px 20px;
  background-color: #0074D9;
  text-decoration: none;
  border: 1px solid #0074D9;
  border-radius: 5px; /* Rounded corners */
  font-weight: bold;
  margin-right:50px
}

.home-button:hover {
  background-color: #0056b3;
}

.home-button input[type="submit"] {
  background-color: #0074D9;
  border: none;
```

```
  color: white;
  font-family:'Poppins', sans-serif;
  font-weight: bold;
}

.button-container {
  display: flex; /* Use flexbox to align items horizontally */
  align-items: center; /* Center items vertically */
  margin-bottom: 30px;
}
```

static/form_display.css

```
.login-box {
    margin-top: 40px;
    margin-bottom: 20px;
    position: absolute;
    top: 50%;
    left: 50%;
    width: 400px;
    padding: 40px;
    transform: translate(-50%, -50%);
    background: rgba(0,0,0,.5);
    box-sizing: border-box;
    box-shadow: 0 15px 25px rgba(0,0,0,.6);
    border-radius: 10px;
  }

  .login-box h2 {
    margin: 0 0 30px;
    padding: 0;
    color: #fff;
    text-align: center;
  }

  .login-box .user-box {
```

```
  position: relative;
}

.login-box .user-box input{
  width: 100%;
  padding: 10px 0;
  font-size: 30px;
  color: #fff;
  margin-bottom: 30px;
  border: none;
  border-bottom: 1px solid #fff;
  outline: none;
  background: transparent;
}
.login-box .user-box label {
  position: absolute;
  top:0;
  left: 0;
  padding: 10px 0;
  font-size: 16px;
  color: #fff;
  pointer-events: none;
  transition: .5s;
}

.login-box .user-box input:focus ~ label,
.login-box .user-box input:valid ~ label {
  top: -20px;
  left: 0;
  color: #03e9f4;
  font-size: 12px;
}

.login-box form a {
  position: relative;
  display: inline-block;
  padding: 10px 20px;
  color: #03e9f4;
  font-size: 16px;
  text-decoration: none;
  text-transform: uppercase;
```

```
  overflow: hidden;
  transition: .5s;
  margin-top: 40px;
  letter-spacing: 4px
}

.login-box a:hover {
  background: #03e9f4;
  color: #fff;
  border-radius: 5px;
  box-shadow: 0 0 5px #03e9f4,
              0 0 25px #03e9f4,
              0 0 50px #03e9f4,
              0 0 100px #03e9f4;
}

.login-box a span {
  position: absolute;
  display: block;
}

.login-box a span:nth-child(1) {
  top: 0;
  left: -100%;
  width: 100%;
  height: 2px;
  background: linear-gradient(90deg, transparent, #03e9f4);
  animation: btn-anim1 1s linear infinite;
}

@keyframes btn-anim1 {
  0% {
    left: -100%;
  }
  50%,100% {
    left: 100%;
  }
}

.login-box a span:nth-child(2) {
  top: -100%;
```

```
  right: 0;
  width: 2px;
  height: 100%;
  background: linear-gradient(180deg, transparent, #03e9f4);
  animation: btn-anim2 1s linear infinite;
  animation-delay: .25s
}

@keyframes btn-anim2 {
  0% {
    top: -100%;
  }
  50%,100% {
    top: 100%;
  }
}

.login-box a span:nth-child(3) {
  bottom: 0;
  right: -100%;
  width: 100%;
  height: 2px;
  background: linear-gradient(270deg, transparent, #03e9f4);
  animation: btn-anim3 1s linear infinite;
  animation-delay: .5s
}

@keyframes btn-anim3 {
  0% {
    right: -100%;
  }
  50%,100% {
    right: 100%;
  }
}


.login-box .button form a {
  position: relative;
  display: inline-block;
  padding: 10px 20px;
```

```
    color: #03e9f4;
    font-size: 16px;
    text-decoration: none;
    text-transform: uppercase;
    overflow: hidden;
    transition: .5s;
    margin-top: 40px;
    letter-spacing: 4px;
}

.flash
{
  color:rgba(254, 108, 108, 0.737);
  font-weight: 550;
  font-size: 35px;
```

static/loading.css

```
/* Config */
:root {
  --sk-size: 40px;
  --sk-color: white;
}

.sk-wander.center {
  margin: 250px 750px;
  display: flex;
  justify-content: center; /* Horizontal centering */
  align-items: center; /* Vertical centering */
  height: 100vh; /* Optional: This ensures that it spans the entire
viewport height. */
}

.sk-wander {
  width: var(--sk-size);
  height: var(--sk-size);
  position: relative;
}
```

```
.sk-wander-cube {
  background-color: var(--sk-color);
  width: 20%;
  height: 20%;
  position: absolute;
  top: 0;
  left: 0;
  --sk-wander-distance: calc(var(--sk-size) * 0.75);
  animation: sk-wander 2.0s ease-in-out -2.0s infinite both;
}
.sk-wander-cube:nth-child(2) { animation-delay: -0.5s; }
.sk-wander-cube:nth-child(3) { animation-delay: -1.0s; }
.sk-wander-cube:nth-child(4) { animation-delay: -1.5s; }

@keyframes sk-wander {
  0% {
    transform: rotate(0deg);
  } 25% {
    transform: translateX(var(--sk-wander-distance)) rotate(-90deg)
scale(0.6);
  } 50% { /* Make FF rotate in the right direction */
    transform: translateX(var(--sk-wander-distance))
translateY(var(--sk-wander-distance)) rotate(-179deg);
  } 50.1% {
    transform: translateX(var(--sk-wander-distance))
translateY(var(--sk-wander-distance)) rotate(-180deg);
  } 75% {
    transform: translateX(0) translateY(var(--sk-wander-distance))
rotate(-270deg) scale(0.6);
  } 100% {
    transform: rotate(-360deg);
  }
}
```

static/result_style.css

```
.container {
    display: flex; /*align items horizontally*/
}

.iframe-container {
    position: relative;
    margin-bottom: 30px;
    top: 50px
}
.wordcloud{

    position: relative;
    margin-left: -1605px;
    margin-top: 250px;
}
ul.flash {
    margin-right: 150px;
    padding: 20px;
    font-size: 16px;
    border: 2px solid #0074D9;
    background-color: #E5E7E9;
    color: #333;
    list-style-type: disc; /* Change the list style */
    text-align: left;
    width: 600px;
    position: relative;
    top: 30px;
    left: 100px;
    border-radius: 5px;
    height: 140px;
}

ul.flash li {
    margin-bottom: 10px; /* Add space between list items */
    font-weight: 400;
}

.results {
    margin-top: 300px;
```

```
    text-align: center;
    margin-right: 10%;
}
.results p, .results a {
    color:azure;
    font-weight: 550;
    font-size: 35px;
}
.results a {
    color: hsl(206, 28%, 48%); /* Adjust the color as needed */
    text-decoration: darkcyan;
}
.results a:hover {
    color: skyblue;
    text-decoration: underline;
}
```

static/word_cloud.css

```
body {
  background-color: #000;
}

.tagcloud {
    font-family: 'Poppins', sans-serif;
    font-size: 20px;
    margin: auto;
    width: 50%;
  }

.tagcloud--item:hover {
color: white;
}

.searchQuery
{
```

```
    color: white;
    padding: 50px;
    font-size: 25px;
}
```

static/word_cloud.js

```
// Define a function to initialize word cloud with receivedKeywords
function wordcloudInit(receivedKeywords) {
    const myTags = receivedKeywords;

    var tagCloud = tagCloud('.content', myTags, {
        // Configuration options for TagCloud
        radius: 250,
        maxSpeed: 'fast',
        initSpeed: 'fast',
        direction: 135,
        keep: true,
    });

    // To change the color of text randomly
    var colors = ['#34A853', '#FBBC05', '#4285F4', '#7FBC00',
'FFBA01', '01A6F0'];
    var random_color = colors[Math.floor(Math.random() *
colors.length)];
    document.querySelector('.content').style.color = random_color;
}
```