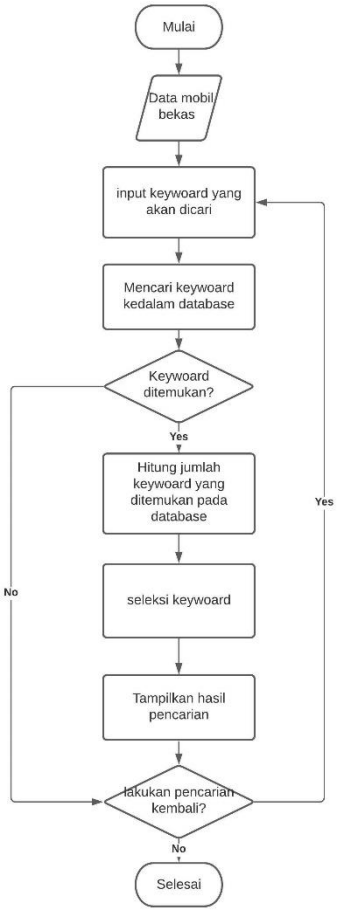
Pengantar

Kendaraan pribadi memiliki peranan penting dalam menjalankan aktivitas sehari-hari seperti bekerja sebagai pengemudi taksi online, mudik, berbelanja, dan lain sebagainya. Namun, calon pembeli kendaraan seringkali merasa ragu karena harga kendaraan yang tidak sesuai dengan kondisi yang diberikan dan mereka bingung dalam memilih kendaraan yang sesuai. Untuk mengatasi masalah ini, teknologi-teknologi terbaru dapat dimanfaatkan dengan membangun aplikasi chatbot yang menggunakan kecerdasan buatan.

Penelitian ini menggunakan 235 data mobil bekas pada tahun 2020 dengan beberapa variabel seperti nama kendaraan, tahun, bahan bakar, transmisi, jarak tempuh, kapasitas mesin, dan harga. Implementasi Natural language programming dengan algoritma levenshtein distance dilakukan secara bertahap melalui beberapa tahap seperti pembersihan data, seleksi data, transformasi data, pengelompokan pertanyaan, penerapan, dan pengujian sistem.

Hasil dari sistem pencarian mobil bekas ini dapat dimanfaatkan oleh pembeli. Metode ini mencapai tingkat akurasi sebesar 69% dengan jumlah 40 kelas pertanyaan dari 2050 data.

Alur Kerja Aplikasi

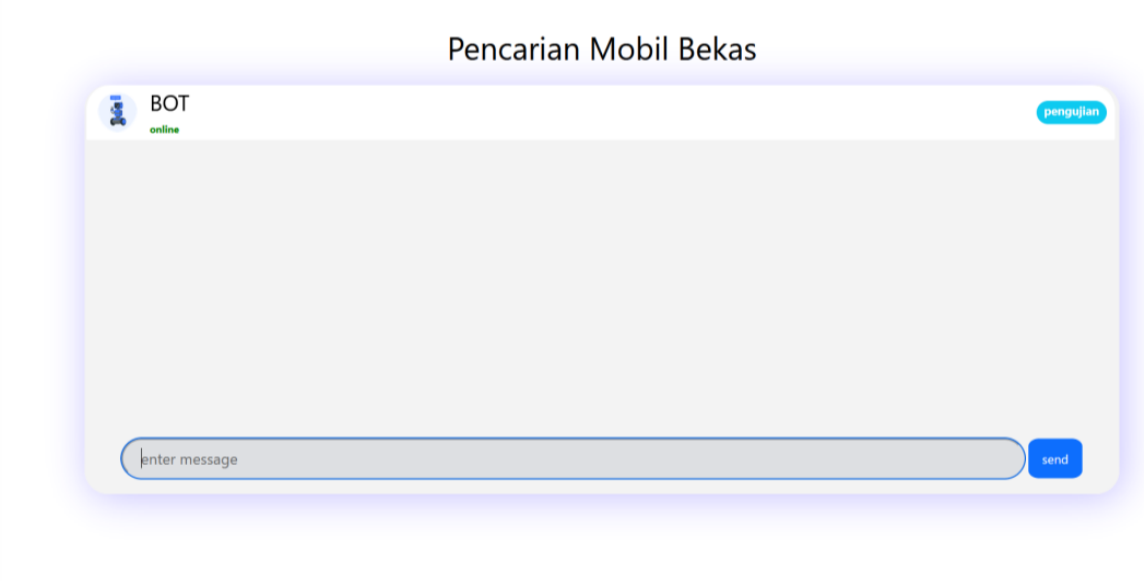


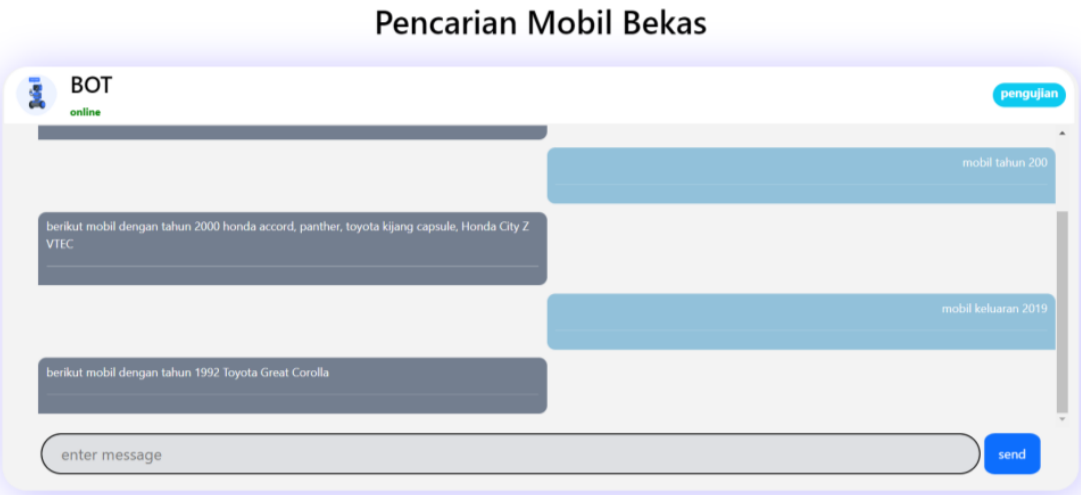
Awalnya, sistem akan membaca data latih dan data uji dari mobil bekas. Kemudian, pengguna memasukkan kata kunci yang sesuai dengan yang mereka cari, dan sistem akan mencari di database untuk mendapatkan hasil pencarian. Jika data yang diminta ditemukan atau memiliki kesamaan dengan kelompok data lain, sistem akan mencocokkan rangkaian kata yang mirip atau dekat dengan

kata yang dimasukkan, dengan menghitung panjang rangkaian kata untuk disesuaikan dengan kelas pada setiap atribut. Jika tidak ada kesamaan yang ditemukan, maka sistem akan menampilkan hasil pencarian yang belum ditemukan. Selanjutnya, sistem akan memeriksa karakter huruf awal dan akhir tanpa menggunakan panjang len, dan jika ditemukan kesamaan, maka sistem akan menampilkan hasil pencarian tersebut. Jika tidak ada kesamaan yang ditemukan atau tidak ada dalam kamus data, maka sistem akan menampilkan hasil pencarian yang sesuai dengan default response di dalam program.

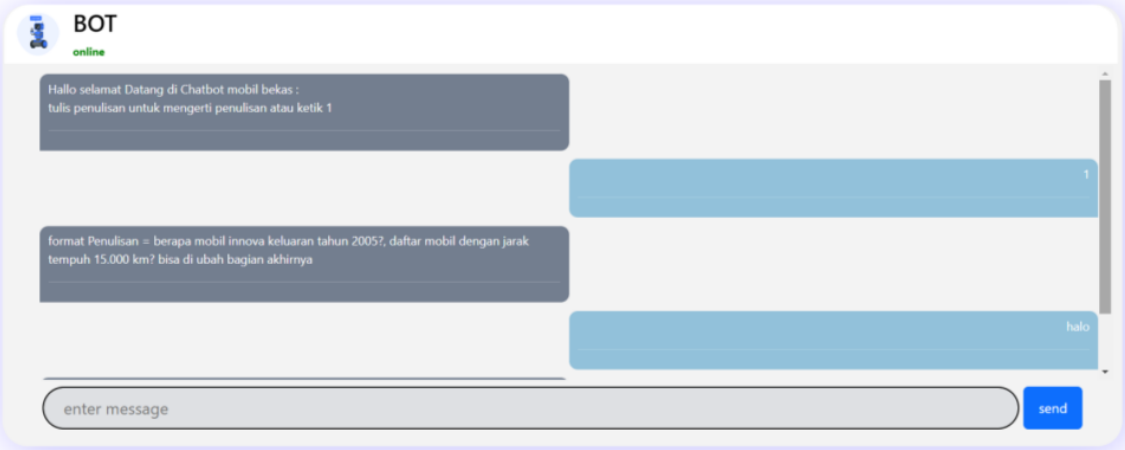
ScreenShoot

Tampilan ini bisa langsung digunakan untuk melakukan komunikasi secara langsung dengan sistem tanpa perlu login, yang membantu pengguna dalam melakukan pencarian mobil bekas.





Pencarian Mobil Bekas



Listing Program

chatterbot.py

```
import logging
from chatterbot.storage import StorageAdapter
from chatterbot.logic import LogicAdapter
from chatterbot.search import TextSearch, IndexedTextSearch
from chatterbot import utils


class ChatBot(object):
    """
    A conversational dialog chat bot.
    """

    def __init__(self, name, **kwargs):
        self.name = name

        storage_adapter = kwargs.get('storage_adapter',
'chatterbot.storage.SQLStorageAdapter')

        logic_adapters = kwargs.get('logic_adapters', [
            'chatterbot.logic.BestMatch'
        ])

        # Check that each adapter is a valid subclass of it's
respective parent
        utils.validate_adapter_class(storage_adapter, StorageAdapter)

        # Logic adapters used by the chat bot
        self.logic_adapters = []

        self.storage = utils.initialize_class(storage_adapter,
**kwargs)

        primary_search_algorithm = IndexedTextSearch(self, **kwargs)
        text_search_algorithm = TextSearch(self, **kwargs)

        self.search_algorithms = {
            primary_search_algorithm.name: primary_search_algorithm,
            text_search_algorithm.name: text_search_algorithm
        }

        for adapter in logic_adapters:
            utils.validate_adapter_class(adapter, LogicAdapter)
            logic_adapter = utils.initialize_class(adapter, self,
**kwargs)
            self.logic_adapters.append(logic_adapter)

        preprocessors = kwargs.get(
            'preprocessors', [
                'chatterbot.preprocessors.clean_whitespace'
            ]
        )

        self.preprocessors = []

        for preprocessor in preprocessors:

self.preprocessors.append(utils.import_module(preprocessor))

        self.logger = kwargs.get('logger', logging.getLogger(__name__))
```

```
        # Allow the bot to save input it receives so that it can learn
        self.read_only = kwargs.get('read_only', False)

    def get_response(self, statement=None, **kwargs):
        """
        Return the bot's response based on the input.

        :param statement: An statement object or string.
        :returns: A response to the input.
        :rtype: Statement

        :param additional_response_selection_parameters: Parameters to
pass to the
            chat bot's logic adapters to control response selection.
        :type additional_response_selection_parameters: dict

        :param persist_values_to_response: Values that should be saved
to the response
            that the chat bot generates.
        :type persist_values_to_response: dict
        """
        Statement = self.storage.get_object('statement')

        additional_response_selection_parameters =
kwargs.pop('additional_response_selection_parameters', {})

        persist_values_to_response =
kwargs.pop('persist_values_to_response', {})

        if isinstance(statement, str):
            kwargs['text'] = statement

        if isinstance(statement, dict):
            kwargs.update(statement)

        if statement is None and 'text' not in kwargs:
            raise self.ChatBotException(
                'Either a statement object or a "text" keyword '
                'argument is required. Neither was provided.'
            )

        if hasattr(statement, 'serialize'):
            kwargs.update(**statement.serialize())

        tags = kwargs.pop('tags', [])

        text = kwargs.pop('text')

        input_statement = Statement(text=text, **kwargs)

        input_statement.add_tags(*tags)

        # Preprocess the input statement
        for preprocessor in self.preprocessors:
            input_statement = preprocessor(input_statement)

        # Make sure the input statement has its search text saved

        if not input_statement.search_text:
```

```
            input_statement.search_text =
self.storage.tagger.get_text_index_string(input_statement.text)

        if not input_statement.search_in_response_to and
input_statement.in_response_to:
            input_statement.search_in_response_to =
self.storage.tagger.get_text_index_string(input_statement.in_response_t
o)

        response = self.generate_response(input_statement,
additional_response_selection_parameters)

        # Update any response data that needs to be changed
        if persist_values_to_response:
            for response_key in persist_values_to_response:
                response_value =
persist_values_to_response[response_key]
                if response_key == 'tags':
                    input_statement.add_tags(*response_value)
                    response.add_tags(*response_value)
                else:
                    setattr(input_statement, response_key,
response_value)
                    setattr(response, response_key, response_value)

        if not self.read_only:
            self.learn_response(input_statement)

            # Save the response generated for the input
            self.storage.create(**response.serialize())

        return response

    def generate_response(self, input_statement,
additional_response_selection_parameters=None):
        """
        Return a response based on a given input statement.

        :param input_statement: The input statement to be processed.
        """
        Statement = self.storage.get_object('statement')

        results = []
        result = None
        max_confidence = -1

        for adapter in self.logic_adapters:
            if adapter.can_process(input_statement):

                output = adapter.process(input_statement,
additional_response_selection_parameters)
                results.append(output)

                self.logger.info(
                    '{} selected "{}" as a response with a confidence
of {}'.format(
                        adapter.class_name, output.text,
output.confidence
                    )
                )
```

```
                if output.confidence > max_confidence:
                    result = output
                    max_confidence = output.confidence
            else:
                self.logger.info(
                    'Not processing the statement using
{}'.format(adapter.class_name)
                )

        class ResultOption:
            def __init__(self, statement, count=1):
                self.statement = statement
                self.count = count

        # If multiple adapters agree on the same statement,
        # then that statement is more likely to be the correct response
        if len(results) >= 3:
            result_options = {}
            for result_option in results:
                result_string = result_option.text + ':' +
(result_option.in_response_to or '')

                if result_string in result_options:
                    result_options[result_string].count += 1
                    if
result_options[result_string].statement.confidence <
result_option.confidence:
                        result_options[result_string].statement =
result_option
                else:
                    result_options[result_string] = ResultOption(
                        result_option
                    )

            most_common = list(result_options.values())[0]

            for result_option in result_options.values():
                if result_option.count > most_common.count:
                    most_common = result_option

            if most_common.count > 1:
                result = most_common.statement

        response = Statement(
            text=result.text,
            in_response_to=input_statement.text,
            conversation=input_statement.conversation,
            persona='bot:' + self.name
        )

        response.confidence = result.confidence

        return response

    def learn_response(self, statement, previous_statement=None):
        """
        Learn that the statement provided is a valid response.
        """
        if not previous_statement:
            previous_statement = statement.in_response_to
```

```
        if not previous_statement:
            previous_statement =
self.get_latest_response(statement.conversation)
            if previous_statement:
                previous_statement = previous_statement.text

        previous_statement_text = previous_statement

        if not isinstance(previous_statement, (str, type(None), )):
            statement.in_response_to = previous_statement.text
        elif isinstance(previous_statement, str):
            statement.in_response_to = previous_statement

        self.logger.info('Adding "{}" as a response to "{}"'.format(
            statement.text,
            previous_statement_text
        ))

        # Save the input statement
        return self.storage.create(**statement.serialize())

    def get_latest_response(self, conversation):
        """
        Returns the latest response in a conversation if it exists.
        Returns None if a matching conversation cannot be found.
        """
        from chatterbot.conversation import Statement as
StatementObject

        conversation_statements = list(self.storage.filter(
            conversation=conversation,
            order_by=['id']
        ))

        # Get the most recent statement in the conversation if one
exists
        latest_statement = conversation_statements[-1] if
conversation_statements else None

        if latest_statement:
            if latest_statement.in_response_to:

                response_statements = list(self.storage.filter(
                    conversation=conversation,
                    text=latest_statement.in_response_to,
                    order_by=['id']
                ))

                if response_statements:
                    return response_statements[-1]
                else:
                    return StatementObject(
                        text=latest_statement.in_response_to,
                        conversation=conversation
                    )
            else:
                # The case that the latest statement is not in response
to another statement
                return latest_statement

        return None
```

```
    class ChatBotException(Exception):
        pass
```

pengujian.py

```
# library dataframe
import pandas as pd
import numpy as np

# load library sklearn
from sklearn.impute import SimpleImputer
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score,
recall_score, precision_score, classification_report
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

from difflib import SequenceMatcher

# library Ipython
from IPython.display import Image

# library
from subprocess import check_call
from datetime import datetime

# library flask
from flask import Flask, render_template, request, jsonify

#library chatterbot
from chatterbot import ChatBot
import chatterbot
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
from chatterbot.response_selection import get_random_response
from chatterbot.comparisons import LevenshteinDistance

# load library matplotlib
import matplotlib.pyplot as plt
import itertools

def uji():
    # load dataset
    data =
pd.read_csv(r'E:/NOVIAN/KULIAH/S7/METOPEN/Datas/allqna(shorted-50) -
40 kelas.csv')
    data = data.iloc[:, 0:2]
    data.head()

    #
    le = preprocessing.LabelEncoder()
    le.fit(data['answers'])
    classes = le.classes_
    y_new = le.transform(data['answers'])

    res = le.inverse_transform([1])
```

```
    data['label_answers'] = y_new
    tabel1 = data.head()

    X = data['question'].values
    y = data['label_answers'].values

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=149)

    # class Comparator:

    #     def __init__(self, language):

    #         self.language = language

    #     def __call__(self, statement_a, statement_b):
    #         return self.compare(statement_a, statement_b)

    #     def compare(self, statement_a, statement_b):
    #         return 0

    #     def match_ratio(statement_a, statement_b):
    #         s = SequenceMatcher(None, statement_a, statement_b)
    #         return s.ratio()

    # class LevenshteinDistance(Comparator):
    #     def compare(self, statement_a, statement_b):

    #         # Return 0 if either statement has a falsy text value
    #         if not statement_a.text or not statement_b.text:
    #             return 0

    #         # Get the lowercase version of both strings
    #         statement_a_text = str(statement_a.text.lower())
    #         statement_b_text = str(statement_b.text.lower())

    #         # similarity = Comparator.get_basic_fuzzy_matches(None,
statement_a_text, statement_b_text)
    #         # print(similarity)
    #         # return similarity
    #         similarity = SequenceMatcher(None, statement_a_text,
statement_b_text)
    #         # Calculate a decimal percent of the similarity
    #         percent = round(similarity.ratio(), 2)
    #         print(percent)
    #         return percent


    # levenshtein_distance = LevenshteinDistance('language')

    bot_lev = ChatBot('ChatBot',
                # response_selection_method=get_random_response,
                statement_comparison_function=LevenshteinDistance,
                logic_adapters = [
                    {
                        'import_path': 'chatterbot.logic.BestMatch',
                        # 'maximum_similarity_threshold': 0.90
                        # 'statement_comparison_function':
levenshtein_distance,
                    }
                ],
```

```
                read_only = True,
                preprocessors=[

'chatterbot.preprocessors.clean_whitespace',
                               'chatterbot.preprocessors.unescape_html',

'chatterbot.preprocessors.convert_to_ascii'
                               ]
                )

    corpusQnA = []
    for i in range(len(X_train)):
        tempQnA = []
        tempQnA.append(str(X_train[i]))
        tempQnA.append(str(y_train[i]))
        # print(x_train[i], " : ", y_train[i])

        corpusQnA.append(tempQnA)

    botv1 = bot_lev
    trainer_manual = ListTrainer(botv1)

    for i in range(len(corpusQnA)):
        trainer_manual.train(corpusQnA[i])

    # userText = 'berapa harga mobil l300'
    y_pred = []
    for uText in X_test:
        bot_response = botv1.get_response(uText)
        # print(bot_response)
        y_pred.append(str(bot_response))

    y_pred_int = []
    for i in y_pred:
        y_pred_int.append(int(i))

    # cek
    example = []
    for i in range(len(y_pred)):
        temp = {}
        temp["1-q"] = X_test[i]
        temp["2-a"] = le.inverse_transform([y_test[i]])[0]
        temp["3-pred"] = le.inverse_transform([y_pred_int[i]])[0]

        example.append(temp)

    # target_names = list(set(y_train))
    list_target = list(set(y_test))

    target_names = []
    for i in list_target:
        target = {}
        target['key'] = "A"+str(i)
        target['value'] = le.inverse_transform([int(str(i))])

        target_names.append(target)

    target_key = []
    for t in target_names:
        target_key.append(t['key'])
```

```
    len(y_pred_int)

    print(classification_report(y_test, y_pred_int, target_names =
target_key))

    cf = confusion_matrix(y_test, y_pred_int)

    #To get better visual of the confusion matrix:
    def plot_confusion_matrix(cm, classes,
            normalize=False,
            title='Confusion matrix',
            cmap=plt.cm.Blues):

        #Add Normalization Option
        # prints pretty confusion metric with normalization option
        if normalize:
            cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
            print('Normalized confusion matrix')
        else:
            print('Confusion matrix, without normalization')

        # print(cm)
        plt.subplots(figsize=(20, 20))
        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=45)
        plt.yticks(tick_marks, classes)

        fmt = '.2f' if normalize else 'd'
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt),
horizontalalignment='center', color='white' if cm[i, j] > thresh else
'black')

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')

    plot_confusion_matrix(cf, target_key, normalize=False)
```

app.py

```
# from pengujian import *
import pandas as pd
from chatterbot import ChatBot
import chatterbot
from flask import Flask, render_template, request, jsonify
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
from chatterbot.response_selection import get_random_response
from chatterbot.comparisons import LevenshteinDistance
from requests import get
from difflib import SequenceMatcher
import os
import spacy
import en_core_web_sm
from pengujian import uji
```

```
# nlp= spacy.load('en_core_web_sm')
nlp = en_core_web_sm.load()

app = Flask(__name__)

def remove_hyphens(statement):
    """
    Remove hypnens.
    """
    statement.text = statement.text.replace('-', '')

    return statement

# class Comparator:

#     def __init__(self, language):

#         self.language = language

#     def __call__(self, statement_a, statement_b):
#         return self.compare(statement_a, statement_b)

#     def compare(self, statement_a, statement_b):
#         return 0

#     def match_ratio(statement_a, statement_b):
#         s = SequenceMatcher(None, statement_a, statement_b)
#         return s.ratio()

# class LevenshteinDistance(Comparator):
#     def compare(self, statement_a, statement_b):

#         # Return 0 if either statement has a falsy text value
#         if not statement_a.text or not statement_b.text:
#             return 0

#         # Get the lowercase version of both strings
#         statement_a_text = str(statement_a.text.lower())
#         statement_b_text = str(statement_b.text.lower())

#         # similarity = Comparator.get_basic_fuzzy_matches(None,
statement_a_text, statement_b_text)
#         # print(similarity)
#         # return similarity
#         similarity = SequenceMatcher(None, statement_a_text,
statement_b_text)
#         # Calculate a decimal percent of the similarity
#         percent = round(similarity.ratio(), 2)
#         print(percent)
#         return percent


# levenshtein_distance = LevenshteinDistance('language')

bot_lev = ChatBot('ChatBot',
            # response_selection_method=get_random_response,
            # statement_comparison_function=levenshtein_distance,
            statement_comparison_function=LevenshteinDistance,
            logic_adapters = [
                {
```

```
                    'import_path': 'chatterbot.logic.BestMatch',
                    # 'default_response': 'Maaf saya tidak mengerti',
                    'maximum_similarity_threshold': 0.90
                }
            ],
            read_only = True,
            preprocessors=[

'chatterbot.preprocessors.clean_whitespace',
                            'chatterbot.preprocessors.unescape_html',

'chatterbot.preprocessors.convert_to_ascii'
                            ]
            )

trainer = ListTrainer(bot_lev)

# for file in os.listdir('E:/NOVIAN/KULIAH/S7/METOPEN/Chatbot/chatbot-
new/yml/'):

#     chats = open('E:/NOVIAN/KULIAH/S7/METOPEN/Chatbot/chatbot-
new/yml/' + file, 'r').readlines()

bot_lev.preprocessors.append(
    remove_hyphens
)

chats = open('allqna.yml', 'r').readlines()
trainer.train(chats)

@app.route("/")
def home():
    return render_template("index.html")

@app.route("/get", methods=['POST'])
def get_bot_response():
    # userText = request.args.get('msg')
    # return bot.get_response(userText)
    userText = str(request.form['msg'])
    bot_response = bot_lev.get_response(userText)
    # breakpoint()
    while True:

        if bot_response.confidence > 0.1:

            bot_response = str(bot_response)
            print(bot_response)
            return jsonify({'status':'OK','answer':bot_response})

        if userText == ("bye"):

            bot_response='Hope to see you soon'

            print(bot_response)
            return jsonify({'status':'OK','answer':bot_response})

            break

        else:
                bot_response = 'Sorry i have no idea about that.'
```

```
                print(bot_response)
                return jsonify({'status':'OK','answer':bot_response})

@app.route('/pengujian',methods =['GET'])
def pengujian():
    if request.method == 'GET':
        confusion_matrix = uji()

        return render_template('pengujian.html', confusion_matrix)

    return render_template('pengujian.html')

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

index.html

```
<!DOCTYPE html>
<html>
  <head>

    <!-- Bootstrap core CSS -->

    <link rel="stylesheet" type="text/css" href="/static/style.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
</script>
    <script src="https://kit.fontawesome.com/f0d1683abb.js"
crossorigin="anonymous"></script>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVor"
crossorigin="anonymous">
    <title>Chatbot</title>
  </head>
  <body>
    <h1>Pencarian Mobil Bekas</h1>
    <div class="container">
      <div class="head_section">
        <img src="{{url_for('static',
filename='/assets/img/profil.png')}}" alt="profile" />
        <div class="bot_info">
          <h3>BOT</h3>
          <p>online</p>
        </div>
      </div>

      <!-- main chat -->
      <div id="chatbox" class="chats">
        <ul class="media-list" id="list">
          <li class="botText"><div class="media-body"><div
class="media"><div style = "color : white" class="media-body">Hallo
selamat Datang di Chatbot mobil bekas : <br> tulis penulisan untuk
mengerti penulisan atau ketik 1<hr/></div></div></div></li>
        </ul>
      </div>
      <!-- end chat -->

      <!-- input field -->
```

```
      <form action="post" id="chatbot-form">
        <div id="userInput" class="chat_input col-md-12">
          <input id="textInput" type="text" name="msg"
placeholder="enter message" autofocus autocomplete="off"/>
          <button id="buttonInput" class="btn send-btn btn-primary"
value="Send">send<i data-feather="circle"></i></button>
        </div>
      </form>
    </div>
    <script src="https://code.jquery.com/jquery-
1.12.4.min.js"></script>

    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.5/dist/umd/popper
.min.js" integrity="sha384-
Xe+8cL9oJa6tN/veChSP7q+mnSPaj5Bcu9mPX5F5xIGE0DVittaqT5lorf0EI7Vk"
crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/js/bootstrap.min.js" integrity="sha384-
kjU+l4N0Yf4ZOJErLsIcvOU2qSb74wXpOhqTvwVx3OElZRweTnQ6d31fXEoRD1Jy"
crossorigin="anonymous"></script>

    <!-- <script src="https://unpkg.com/browse/speech-to-
text@2.9.1/lib/index.js"></script> -->

    <script>
        var exports = {};
    </script>
    <script>
      function getBotResponse() {
        var rawText = $("#textInput").val();
        var userHtml = '<p class="userText"><span>' + rawText +
'</span></p>';
        $("#textInput").val("");
        $("#chatbox").append(userHtml);
        document.getElementById('userInput').scrollIntoView({block:
'start', behavior: 'smooth'});
        $("#userInput").scroll()
        $.get("/get", { msg: rawText }).done(function(data) {
          var botHtml = '<p class="botText"><span>' + data +
'</span></p>';
          $("#chatbox").append(botHtml);
          document.getElementById('userInput').scrollIntoView({block:
'start', behavior: 'smooth'});
        });
      }
      $('#chatbot-form').submit(function (e) {
        e.preventDefault();
        var message = $('#textInput').val();
        $(".media-list").append(
            '<li class="userText"><div class="media-body"><div
class="media"><div style = "text-align:right; color : white"
class="media-body">' +
            message + '<hr/></div></div></div></li>');
        $.ajax({
            type: "POST",
            url: "/get",
            data: $(this).serialize(),
            success: function (response) {
                $('#textInput').val('');
                var answer = response.answer;
```

```
                const chatbox = document.getElementById("chatbox");
                $(".media-list").append(
                    '<li class="botText"><div class="media-body"><div
class="media"><div style = "color : white" class="media-body">' +
                    answer + '<hr/></div></div></div></li>');
                $(".botText").stop().animate({
                    scrollTop: $(".botText")[0].scrollHeight
                }, 1000);
                msg.text = answer;
                speechSynthesis.speak(msg);
            },
            error: function (error) {
                console.log(error);
            }
        });
      });
    </script>

    <script src="https://code.jquery.com/jquery-3.6.0.js"
integrity="sha256-H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk="
crossorigin="anonymous"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/js/bootstrap.bundle.min.js" integrity="sha384-
pprn3073KE6tl6bjs2QrFaJGz5/SUsLqktiwsUTF55Jfv3qYSDhgCecCxMW52nD2"
crossorigin="anonymous"></script>
  </body>
</html>
```

keterangan.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Keterangan</title>
</head>
<body>
    <h1>Pencarian Mobil Bekas</h1>
    <div>
        <p>Hasil Nilai = {{ tabel1 }}</p>
    </div>
</body>
</html>
```