챗봇 프로젝트 11

개체명 인식 모델 학습

챗봇 엔진에 입력된 문장의 의도가 분류된 후, 문장 내 개체명 인식을 진행한다.
개체명 인식을 위해 양방향 LSTM이라는 모델을 사용하게 된다.
이번 모델을 구현할 때, 인식이 가능한 주요 개체명은 다음과 같다.

개체명	설명
B_FOOD	음식
B_DT, B_TI	날짜, 시간
B_PS	사람
B_OG	조직, 회사
B_LC	지역

train_model.py

models/ner 폴더 안에 코드 작성

버전 충돌로 인해 다운그레이드 필요.
pip uninstall matplotlib / pip uninstall numpy
pip install matplotlib==3.6 / pip install numpy==1.19.5

```
import sys
sys.path.append('../../')
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import preprocessing
from sklearn.model_selection import train_test_split
import numpy as np
from util.Preprocess import Preprocess
```

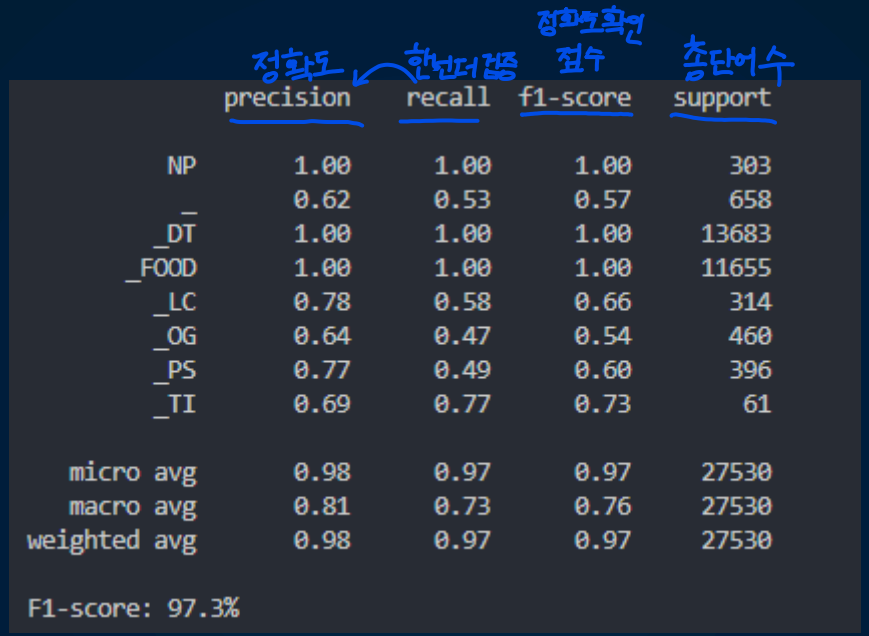
```
def read_file(file_name):
    sents = []
    with open(file_name, 'r', encoding='utf-8') as f:
        lines = f.readlines()
        for idx, l in enumerate(lines):
            if l[0] == ';' and lines[idx + 1][0] == '$':
                this_sent = []
            elif l[0] == '$' and lines[idx - 1][0] == ';':
                continue
            elif l[0] == '\n':
                sents.append(this_sent)
            else:
                this_sent.append(tuple(l.split()))
    return sents

p =
Preprocess(word2index_dic='../../train_tools/dict/chatbot_dict.bi
n',
               userdic='../../util/user_dic.tsv')
```

```
corpus = read_file('ner_train.txt')

sentences, tags = [], []
for t in corpus:
    tagged_sentence = []
    sentence, bio_tag = [], []
    for w in t:
        tagged_sentence.append((w[1], w[3]))
        sentence.append(w[1])
        bio_tag.append(w[3])

    sentences.append(sentence)
    tags.append(bio_tag)


print("샘플 크기 : \n", len(sentences))
print("0번 째 샘플 단어 시퀀스 : \n", sentences[0])
print("0번 째 샘플 bio 태그 : \n", tags[0])
print("샘플 단어 시퀀스 최대 길이 :", max(len(l) for l in sentences))
print("샘플 단어 시퀀스 평균 길이 :", (sum(map(len,
sentences))/len(sentences)))
```

```
tag_tokenizer = preprocessing.text.Tokenizer(lower=False)
tag_tokenizer.fit_on_texts(tags)

vocab_size = len(p.word_index) + 1
tag_size = len(tag_tokenizer.word_index) + 1
print("BIO 태그 사전 크기 :", tag_size)
print("단어 사전 크기 :", vocab_size)

x_train = [p.get_wordidx_sequence(sent) for sent in sentences]
y_train = tag_tokenizer.texts_to_sequences(tags)

index_to_ner = tag_tokenizer.index_word
index_to_ner[0] = 'PAD'

max_len = 40
x_train = preprocessing.sequence.pad_sequences(x_train,
padding='post', maxlen=max_len)
y_train = preprocessing.sequence.pad_sequences(y_train,
padding='post', maxlen=max_len)
```

```
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train,
test_size=.2, random_state=1234)

y_train = tf.keras.utils.to_categorical(y_train, num_classes=tag_size)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=tag_size)

print("학습 샘플 시퀀스 형상 : ", x_train.shape)
print("학습 샘플 레이블 형상 : ", y_train.shape)
print("테스트 샘플 시퀀스 형상 : ", x_test.shape)
print("테스트 샘플 레이블 형상 : ", y_test.shape)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, Dense, TimeDistributed,
Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=30,
input_length=max_len, mask_zero=True))
model.add(Bidirectional(LSTM(200, return_sequences=True, dropout=0.50,
recurrent_dropout=0)))
model.add(TimeDistributed(Dense(tag_size, activation='softmax')))
model.compile(loss='categorical_crossentropy', optimizer=Adam(0.01),
metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=128, epochs=10)
```

```
print("평가 결과 : ", model.evaluate(x_test, y_test)[1])
model.save('ner_model.h5')

def sequences_to_tag(sequences):
    result = []
    for sequence in sequences:
        temp = []
        for pred in sequence:
            pred_index = np.argmax(pred)
            temp.append(index_to_ner[pred_index].replace("PAD", "O"))
        result.append(temp)
    return result

from seqeval.metrics import f1_score, classification_report

y_predicted = model.predict(x_test)
pred_tags = sequences_to_tag(y_predicted)
test_tags = sequences_to_tag(y_test)

print(classification_report(test_tags, pred_tags))
print("F1-score: {:.1%}".format(f1_score(test_tags, pred_tags)))
```



각 개체명의 밀도, 재현율, 검증 점수 가 출력된다.

개체명 인식 모델 학습이 완료되면 ner_model.h5 모델 파일 생성

NerModel.py

models/ner 폴더 안에 코드 작성

개체명 인식에 필요한 기능을 구현한 개체명 인식 모듈 작성

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Model, load_model
from tensorflow.keras import preprocessing

class NerModel:
    def __init__(self, model_name, proprocess):

        self.index_to_ner = {1: 'O', 2: 'B_DT', 3: 'B_FOOD', 4: 'I', 5:
'B_OG', 6: 'B_PS', 7: 'B_LC', 8: 'NNP', 9: 'B_TI', 0: 'PAD'}

        self.model = load_model(model_name)

        self.p = proprocess
```

```
    def predict(self, query):
        pos = self.p.pos(query)

        keywords = self.p.get_keywords(pos, without_tag=True)
        sequences = [self.p.get_wordidx_sequence(keywords)]

        max_len = 40
        padded_seqs = preprocessing.sequence.pad_sequences(sequences,
padding="post", value=0, maxlen=max_len)

        predict = self.model.predict(np.array([padded_seqs[0]]))
        predict_class = tf.math.argmax(predict, axis=-1)

        tags = [self.index_to_ner[i] for i in predict_class.numpy()[0]]
        return list(zip(keywords, tags))

    def predict_tags(self, query):
        pos = self.p.pos(query)

        keywords = self.p.get_keywords(pos, without_tag=True)
        sequences = [self.p.get_wordidx_sequence(keywords)]
```

```
max_len = 40
        padded_seqs = preprocessing.sequence.pad_sequences(sequences,
padding="post", value=0, maxlen=max_len)

        predict = self.model.predict(np.array([padded_seqs[0]]))
        predict_class = tf.math.argmax(predict, axis=-1)

        tags = []
        for tag_idx in predict_class.numpy()[0]:
            if tag_idx == 1: continue
            tags.append(self.index_to_ner[tag_idx])

        if len(tags) == 0: return None
        return tags
```

해당 NerModel 클래스를 테스트 하기 위해서 추가 코드 작성 필요. (출력X)

model_ner_test.py

test 폴더 안에 코드 작성

개체명 인식에 필요한 기능을 구현한 개체명 인식 모듈 작성

```
import sys
sys.path.append('../')
from util.Preprocess import Preprocess
from models.ner.NerModel import NerModel

p = Preprocess(word2index_dic='../train_tools/dict/chatbot_dict.bin',
               userdic='../util/user_dic.tsv')


ner = NerModel(model_name='../models/ner/ner_model.h5', proprocess=p)
query = '오늘 오전 13시 2분에 탕수육 주문 하고 싶어요'
predicts = ner.predict(query)
tags = ner.predict_tags(query)
print(predicts)
print(tags)
```

```
[('오늘', 'B_DT'), ('오전', 'B_DT'), ('13시', 'B_DT'), ('2분', 'B_DT'), ('탕수육', 'B_FOOD'), ('주문', 'O'), ('하', 'O'), ('싶', 'O')]
['B_DT', 'B_DT', 'B_DT', 'B_DT', 'B_FOOD']
```

코드에 넣은 테스트 문장의 개체명이 출력된다.

학습 데이터와 유사한 유형의 문장을 입력해서 개체명을 잘 인식했지만,
데이터와 많이 다른 문장을 넣으면 개체명 인식 정확도가 떨어질 수 있음.

답변 검색

입력된 문장의 전처리, 의도 분류, 개체명 인식 과정을 거쳐 해석된 데이터를 기반으로

적절한 답변을 학습 DB로부터 검색하는 기능을 구현.

검색 기능은 네이버 같은 포털사이트의 검색엔진

Database.py

util 폴더에 작성

답변 검색 기능을 사용할 때, 데이터베이스 접근과 제어를 쉽게 할 수 있도록,
해당 모듈을 먼저 생성한다.

```python
import pymysql
import pymysql.cursors
import logging

class Database:
    def __init__(self, host, user, password, db_name, charset='utf8'):
        self.host = host
        self.user = user
        self.password = password
        self.charset = charset
        self.db_name = db_name
        self.conn = None
```

```python
# DB 연결
def connect(self):
    if self.conn != None:
        return

    self.conn = pymysql.connect(
        host=self.host,
        user=self.user,
        password=self.password,
        db=self.db_name,
        charset=self.charset
    )

# DB 연결 닫기
def close(self):
    if self.conn is None:
        return

    if not self.conn.open:
        self.conn = None
        return
    self.conn.close()
    self.conn = None
```

```
    # SQL 구문 실행
    def execute(self, sql):
        last_row_id = -1
        try:
            with self.conn.cursor() as cursor:
                cursor.execute(sql)
            self.conn.commit()
            last_row_id = cursor.lastrowid
            # logging.debug("excute last_row_id : %d", last_row_id)
        except Exception as ex:
            logging.error(ex)

        finally:
            return last_row_id
    # SELECT 구문 실행 후, 단 1개의 데이터 ROW만 불러옴
    def select_one(self, sql):
        result = None

        try:
            with self.conn.cursor(pymysql.cursors.DictCursor) as cursor:
                cursor.execute(sql)
                result = cursor.fetchone()
        except Exception as ex:
            logging.error(ex)

        finally:
            return result
```

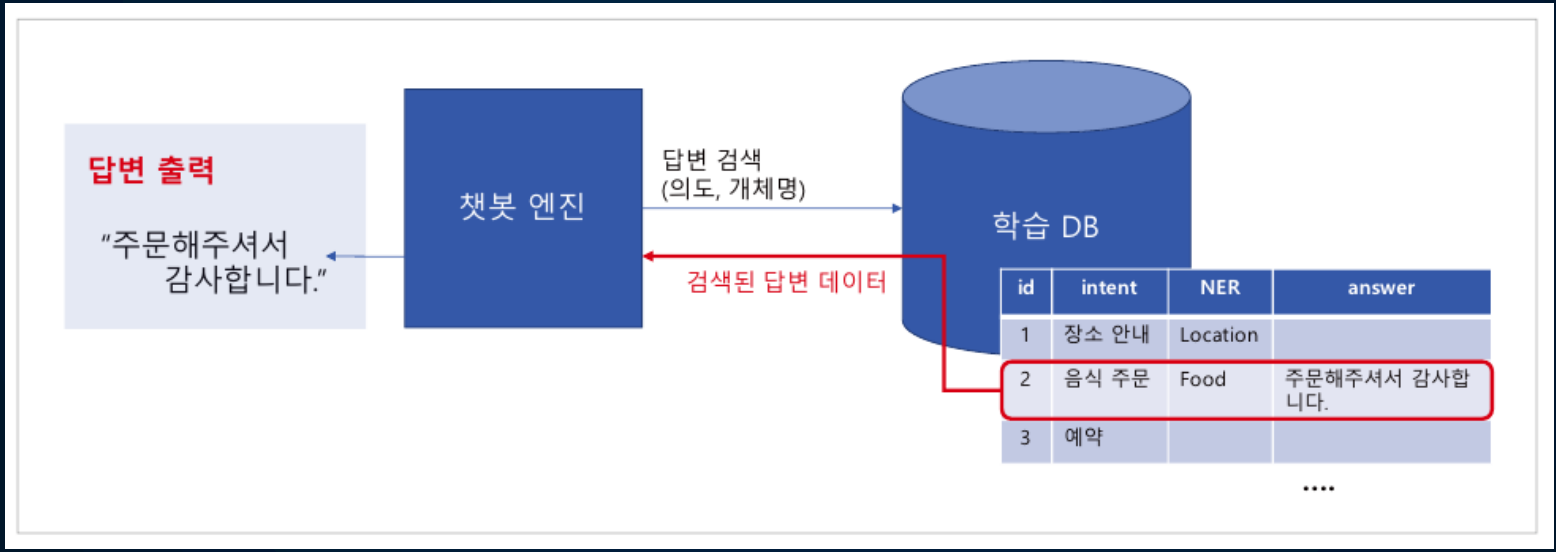
```python
# SELECT 구문 실행 후, 전체 데이터 ROW만 불러옴
def select_all(self, sql):
    result = None

    try:
        with self.conn.cursor(pymysql.cursors.DictCursor) as cursor:
            cursor.execute(sql)
            result = cursor.fetchall()
    except Exception as ex:
        logging.error(ex)

    finally:
        return result
```

해당 코드는 데이터베이스에 접근하고 전달받은 SQL문을 실행하는 제어 역할만 하기 때문에

검색 기능 구현을 위해서는 추가 모듈 구현 필요

답변 검색

전처리, 의도 분류, 개체명 인식 과정을 거쳐서 나온 자연어 해석 결과를 이용해
학습 DB에서 적절한 답변을 검색한다.
이 중에서 의도명과 개체명 2가지 항목을 가지고 답변을 검색하는 기능을 구현한다.



findAnswer.py

util 폴더에 작성

답변 검색에 필요한 기능을 제공하는 FindAnswer클래스를 생성한다.

```
class FindAnswer:
    def __init__(self, db):
        self.db = db

    # 검색 쿼리 생성
    def _make_query(self, intent_name, ner_tags):
        sql = "select * from chatbot_train_data"
        if intent_name != None and ner_tags == None:
            sql = sql + " where intent='{}' ".format(intent_name)

        elif intent_name != None and ner_tags != None:
            where = ' where intent="%s" ' % intent_name
            if (len(ner_tags) > 0):
                where += 'and ('
                for ne in ner_tags:
                    where += " ner like '%{}%' or ".format(ne)
                where = where[:-3] + ')'
            sql = sql + where

        # 동일한 답변이 2개 이상인 경우, 랜덤으로 선택
        sql = sql + " order by rand() limit 1"
        return sql
```

```python
    # 답변 검색
    def search(self, intent_name, ner_tags):
        # 의도명, 개체명으로 답변 검색
        sql = self._make_query(intent_name, ner_tags)
        answer = self.db.select_one(sql)

        # 검색되는 답변이 없으면 의도명만 검색
        if answer is None:
            sql = self._make_query(intent_name, None)
            answer = self.db.select_one(sql)

        return (answer['answer'], answer['answer_image'])

    # NER 태그를 실제 입력된 단어로 변환
    def tag_to_word(self, ner_predicts, answer):
        for word, tag in ner_predicts:

            # 변환해야하는 태그가 있는 경우 추가
            if tag == 'B_FOOD' or tag == 'B_DT' or tag == 'B_TI':
                answer = answer.replace(tag, word)

        answer = answer.replace('{', '')
        answer = answer.replace('}', '')
        return answer
```

chatbot_test.py

test 폴더에 작성

findAnswer로 구현한 검색 기능 클래스를 테스트 하는 코드를 작성
해당 코드가 입력받은 문장을 기반으로 답변을 찾아 출력하는 실질적인 챗봇의 엔진 역할을 한다.

```
import sys
sys.path.append('../')
from config.DatabaseConfig import *
from util.Database import Database
from util.Preprocess import Preprocess

# 전처리 객체 생성
p = Preprocess(word2index_dic='../train_tools/dict/chatbot_dict.bin',
               userdic='../util/user_dic.tsv')

# 질문/답변 학습 디비 연결 객체 생성
db = Database(
    host=DB_HOST, user=DB_USER, password=DB_PASSWORD, db_name=DB_NAME
)
db.connect()    # 디비 연결
```

```
query = "자장면 주문할게요"

# 의도 파악
from models.intent.IntentModel import IntentModel
intent = IntentModel(model_name='../models/intent/intent_model.h5',
proprocess=p)
predict = intent.predict_class(query)
intent_name = intent.labels[predict]

# 개체명 인식
from models.ner.NerModel import NerModel
ner = NerModel(model_name='../models/ner/ner_model.h5', proprocess=p)
predicts = ner.predict(query)
ner_tags = ner.predict_tags(query)

print("질문 : ", query)
print("=" * 100)
print("의도 파악 : ", intent_name)
print("개체명 인식 : ", predicts)
print("답변 검색에 필요한 NER 태그 : ", ner_tags)
print("=" * 100)
```

```python
# 답변 검색
from util.findAnswer import FindAnswer

try:
    f = FindAnswer(db)
    answer_text, answer_image = f.search(intent_name, ner_tags)
    answer = f.tag_to_word(predicts, answer_text)
except:
    answer = "죄송해요 무슨 말인지 모르겠어요"

print("답변 : ", answer)

db.close() # 디비 연결 끊음
```

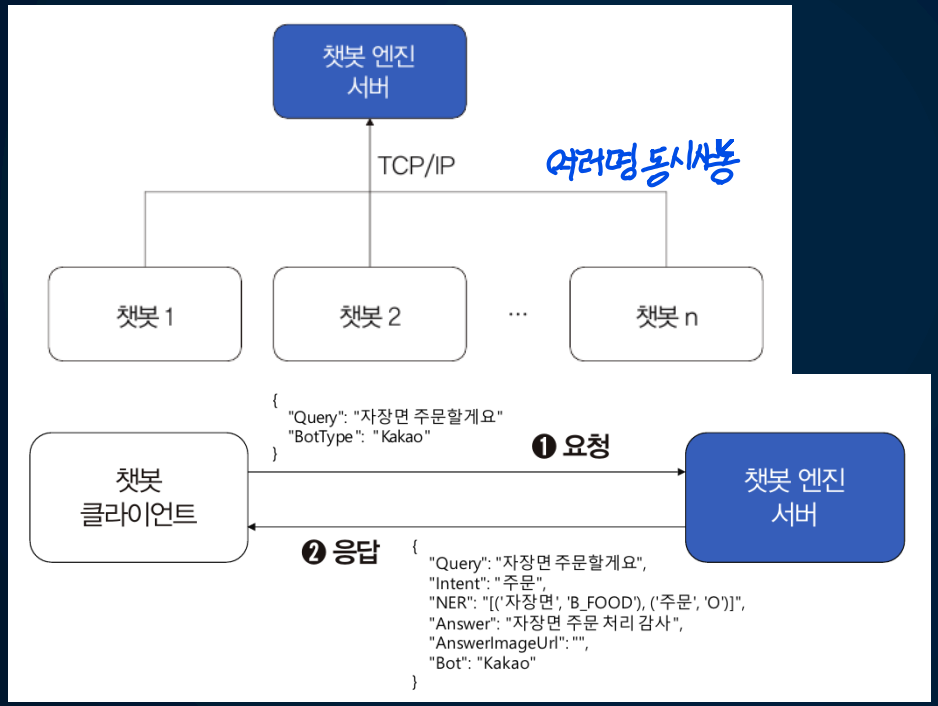
실행 시,
질문, 답변 검색에 필요한
의도, 개체명 추출.
해당 질문에 맞는 답변을 검색하여 출력

```
질문 :  자장면 주문할게요
==========================================================
의도 파악 :  주문
개체명 인식 :  [('자장면', 'B_FOOD'), ('주문', 'O')]
답변 검색에 필요한 NER 태그 :  ['B_FOOD']
==========================================================
답변 :  자장면 주문 처리 감사!!
```

서버 구현

챗봇 엔진을 실제로 다양한 플랫폼에서 적용하기 위해 서버통신을 위한 기능을 구현.

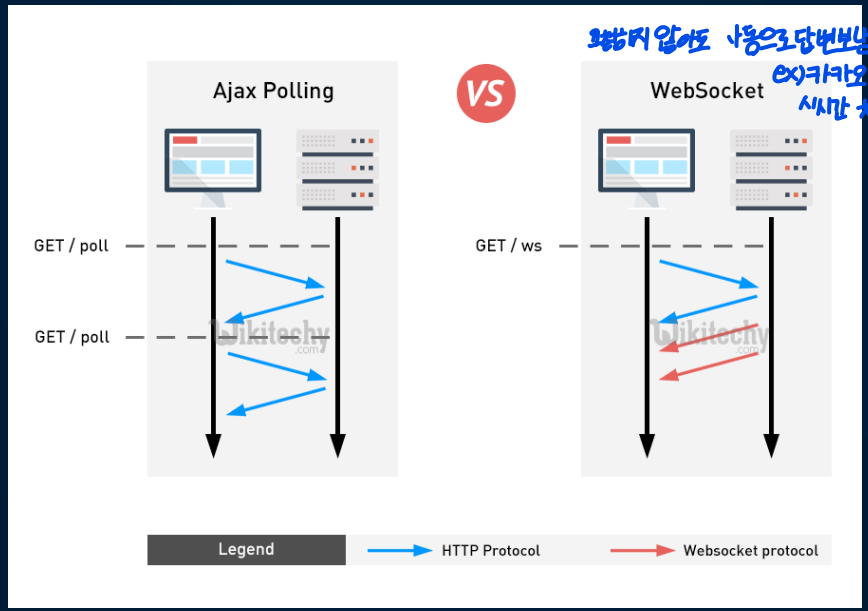
소켓서버



소켓 서버

서버와 클라이언트가 특정 포트를 통해 실시간으로 양방향 통신을 하는 방식

HTTP통신 VS 소켓통신



BotServer.py

util 폴더에 작성

TCP 소켓 서버를 관리하는 모듈을 먼저 구현한다.
해당 모듈은 서버에 접속하는 클라이언트 소켓을 생성하고 처리하는 기능을 담당한다.

```python
import socket

class BotServer:
    def __init__(self, srv_port, listen_num):
        self.port = srv_port
        self.listen = listen_num
        self.mySock = None

    def create_sock(self):
        self.mySock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.mySock.bind(("0.0.0.0", int(self.port)))
        self.mySock.listen(int(self.listen))
        return self.mySock

    def ready_for_client(self):
        return self.mySock.accept()

    def get_sock(self):
        return self.mySock
```

Bot.py

chatbot 폴더에 작성

최종적으로 실제 챗봇 기능을 담당하는 메인 프로그램을 구현한다.
챗봇 엔진 동작을 서버 환경에서 작동하기 위한 BotServer클래스,
여러 개의 클라이언트에서 동작될 수 있도록 멀티 스레드 모듈을 이용한다.

```
import threading
import json

from config.DatabaseConfig import *
from util.Database import Database
from util.BotServer import BotServer
from util.Preprocess import Preprocess
from models.intent.IntentModel import IntentModel
from models.ner.NerModel import NerModel
from util.findAnswer import FindAnswer

p = Preprocess(word2index_dic='train_tools/dict/chatbot_dict.bin',
               userdic='util/user_dic.tsv')

intent = IntentModel(model_name='models/intent/intent_model.h5', proprocess=p)


ner = NerModel(model_name='models/ner/ner_model.h5', proprocess=p)
```

```python
def to_client(conn, addr, params):
    db = params['db']

    try:
        db.connect()  # 디비 연결

        # 데이터 수신
        read = conn.recv(2048)  # 수신 데이터가 있을 때 까지 블로킹
        print('==========================')
        print('Connection from: %s' % str(addr))

        if read is None or not read:
            # 클라이언트 연결이 끊어지거나, 오류가 있는 경우
            print('클라이언트 연결 끊어짐')
            exit(0)


        # json 데이터로 변환
        recv_json_data = json.loads(read.decode())
        print("데이터 수신 : ", recv_json_data)
        query = recv_json_data['Query']

        # 의도 파악
        intent_predict = intent.predict_class(query)
        intent_name = intent.labels[intent_predict]
```

```
# 개체명 파악
ner_predicts = ner.predict(query)
ner_tags = ner.predict_tags(query)


# 답변 검색
try:
    f = FindAnswer(db)
    answer_text, answer_image = f.search(intent_name, ner_tags)
    answer = f.tag_to_word(ner_predicts, answer_text)

except:
    answer = "죄송해요 무슨 말인지 모르겠어요. 조금 더 공부 할게요."
    answer_image = None

send_json_data_str = {
    "Query" : query,
    "Answer": answer,
    "AnswerImageUrl" : answer_image,
    "Intent": intent_name,
    "NER": str(ner_predicts)
}
message = json.dumps(send_json_data_str)
conn.send(message.encode())
```

```
    except Exception as ex:
        print(ex)

    finally:
        if db is not None: # db 연결 끊기
            db.close()
        conn.close()


if __name__ == '__main__':

    # 질문/답변 학습 디비 연결 객체 생성
    db = Database(
        host=DB_HOST, user=DB_USER, password=DB_PASSWORD, db_name=DB_NAME
    )
    print("DB 접속")

    port = 5050
    listen = 100
```

```python
# 봇 서버 동작
    bot = BotServer(port, listen)
    bot.create_sock()
    print("bot start")

    while True:
        conn, addr = bot.ready_for_client()
        params = {
            "db": db
        }

        client = threading.Thread(target=to_client, args=(
            conn,
            addr,
            params
        ))
        client.start()
```

해당 파일을 실행하면 서버가 활성화되고, 실행이 되는 중이라면 연결된 클라이언트에서 해당 챗봇의 기능을 사용할 수 있다.

```
DB 접속
bot start
```

chatbot_client_test.py test 폴더에 작성

챗봇의 엔진을 담당하는 서버 프로그램을 구현했지만, 실제로 작동이 되는지를 확인하기 위해서 테스트 코드를 작성하여, 질문을 보냈을 때 그에 따른 적절한 답변이 나오는지 확인해본다.

```
import socket
import json

host = "127.0.0.1"
port = 5050

# 클라이언트 프로그램 시작
while True:
    print("질문 : ")
    query = input()
    if(query == "exit"):
        exit(0)
    print("-" * 40)

    # 챗봇 엔진 서버 연결
    mySocket = socket.socket()
    mySocket.connect((host, port))
```

```python
# 챗봇 엔진 질의 요청
    json_data = {
        'Query': query,
        'BotType': "MyService"
    }
    message = json.dumps(json_data)
    mySocket.send(message.encode())

    # 챗봇 엔진 답변 출력
    data = mySocket.recv(2048).decode()
    ret_data = json.loads(data)
    print("답변 : ")
    print(ret_data['Answer'])
    print("\n")

    # 챗봇 엔진 서버 연결 소켓 닫기
    mySocket.close()
```

클라이언트

```
질문 :
짜장면 배달 되나요
----------------------------------------
답변 :
짜장면 주문 처리 감사!!
```

서버

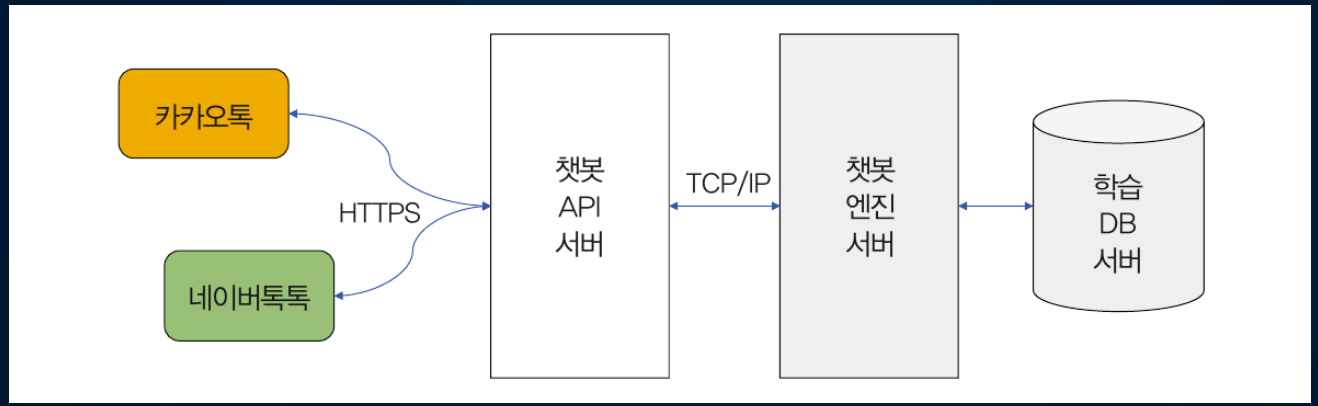
```
DB 접속
bot start
==========================
Connection from: ('127.0.0.1', 13066)
데이터 수신 :  {'Query': '짜장면 배달 되나요', 'BotType': 'MyService'}
```

메신저와 연동

지금까지의 단계에서는 화자의 질의를 해석하여 알맞은 답변을 제공하는 챗봇 엔진 구현에 집중했다면,

이번에는 다양한 메신저 플랫폼과 어떻게 통신을 해서, 챗봇 엔진의 결과물을 카카오톡같은 메신저 상의 말풍선으로 보여주는지 알아본다.

ex) 카카오톡



FLASK

간단한 웹 사이트, 혹은 간단한 API 서버를 만드는 데 특화된 프레임워크.
장점으로는 쉽고 간단하게 웹 서버를 구현할 수 있고 배포도 간단하다는 점이 있다.
하지만 복잡한 서비스를 만들기에는 어려움이 있다.

Flask 프레임워크로 다른 채팅 서비스와 통신을 위한 API를 만들어본다.

Flask 설치
pip install flask

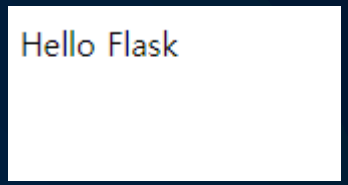
app.py

임의의 경로에 ex_flask 폴더를 생성 후, 파일 생성

Flask의 동작 방식을 이해하기 위해 간단한 웹 페이지를 구현한다.
“Hello Flask”를 띄우는 페이지를 구현

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello Flask'

if __name__ == '__main__':
    app.run()
```

```
(chatbot2) C:\Users\yubeen\python-chatbot\chatbot\flask>python app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production d
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```



로컬호스트 웹페이지작성

클릭

Ctrl

실행 후, Running on http:// { ip 주소 } 출력 시 웹 페이지 구현 성공

해당 주소로 이동 시, Hello Flask가 출력되는 것을 확인 가능.

app.py

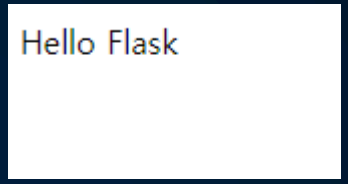
앞에서 작성한 파일에 코드 추가

HTTP 메서드 (GET, POST, DELETE, PUT)에 따라
URI 호출을 통해 동적 변수를 처리하는 기능을 추가해본다

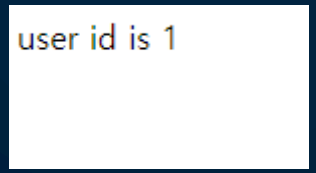
```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello Flask'

@app.route('/info/<name>')
def get_name(name):
    return "hello {}".format(name)
```

```python
@app.route('/user/<int:id>')
def get_user(id):
    return "user id is {}".format(id)

@app.route('/json/<int:dest_id>/<message>')
@app.route('/JSON/<int:dest_id>/<message>')
def send_message(dest_id, message):
    json = {
    "bot_id": dest_id,
    "message": message
    }
    return json

if __name__ == '__main__':
    app.run()
```

```
(chatbot2) C:\Users\yubeen\python-chatbot\chatbot\flask>python app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production d
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```



실행하면 방금과 같은 페이지가 출력됨.
주소를 다음과 같이 입력하면 입력한 값에 따라 페이지의 내용이 동적으로 변한다.
ex) http://127.0.0.1:5000/user/1
http:// { ip주소 } / { 유저명 } / { 유저id }



REST API

REST API는 기능에 따라 GET, POST, DELETE, PUT의 HTTP 메서드를 사용한다.
클라이언트로부터 요청이 들어왔을 때 HTTP 메서드별로 함수를 정의한다.
CRUD 동작이 어떤 HTTP 메서드와 연결되어 있는지는 아래의 표를 확인한다.

HTTP 메서드	CRUD 동작	설명
POST	Create	서버 리소스를 생성한다
GET	Read	서버 리소스를 읽어온다
PUT	Update	서버 리소스를 수정한다
DELETE	Delete	서버 리소스를 삭제한다

4가지의 메서드가 있지만 POST, GET만 적용을 한다.

app.py

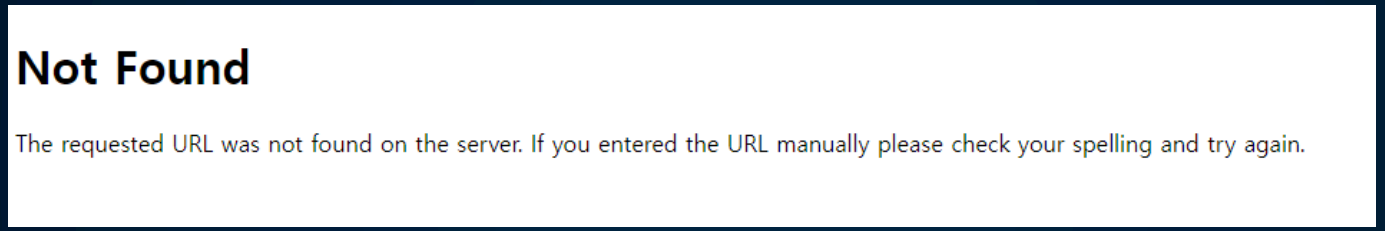
임의의 경로에 main_flask 폴더를 생성 후, 파일 생성

Flask의 동작 방식을 이해하기 위해 간단한 웹 페이지를 구현한다.
“Hello Flask”를 띄우는 페이지를 구현

```
from flask import Flask, request, jsonify
app = Flask(__name__)

# 서버 리소스
resource = []

# 사용자 정보 조회
@app.route('/user/<int:user_id>', methods=['GET'])
```

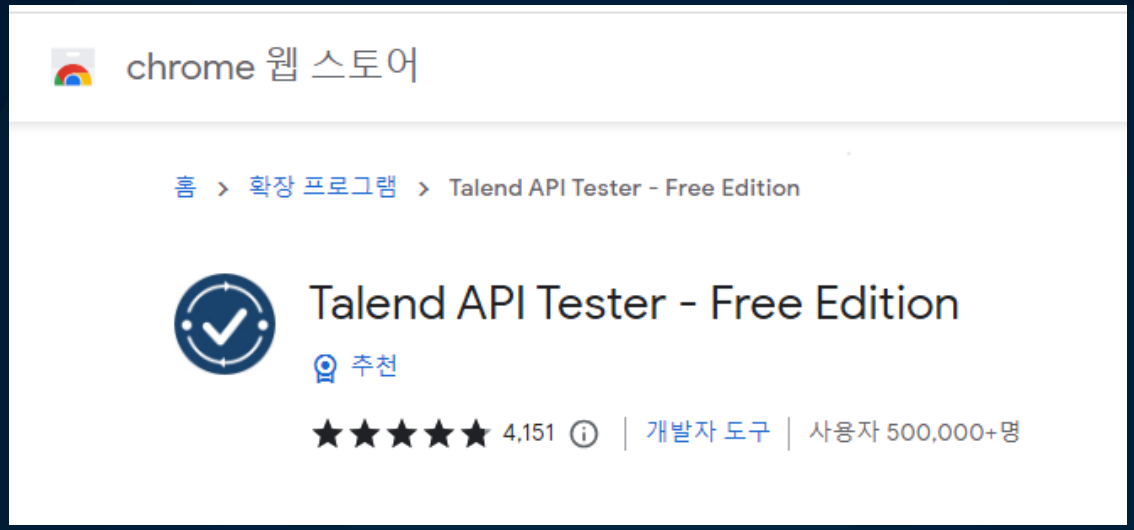
```python
def get_user(user_id):
    for user in resource:
        if user['user_id'] is user_id:
            return jsonify(user)

    return jsonify(None)


# 사용자 추가
@app.route('/user', methods=['POST'])
def add_user():
    user = request.get_json()
    resource.append(user)
    return jsonify(resource)


if __name__ == '__main__':
    app.run()
```

```
(chatbot2) C:\Users\yubeen\python-chatbot\chatbot\flask>python app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production d
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```



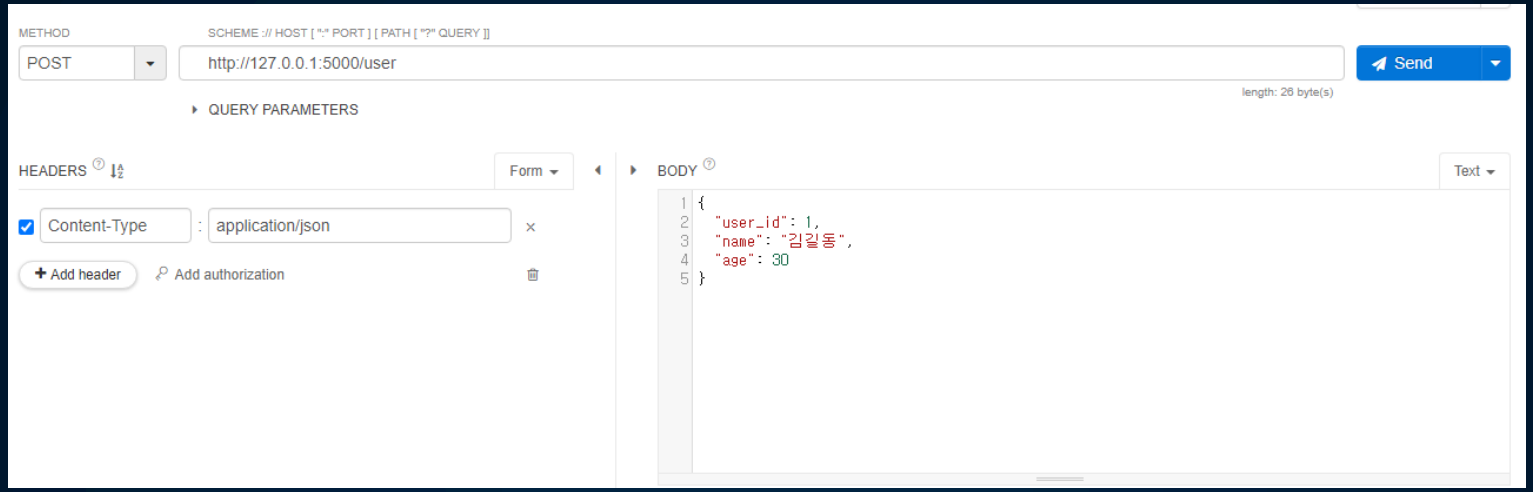
사용자 정보를 조회하는 기능을 구현했지만, 사용자 정보가 존재하지 않기 때문에
Not Found 오류 문구를 출력한다.

REST API를 테스트하기 위해서 추가적인 작업이 필요하다.



크롬 확장프로그램 – Talend API Tester 다운로드

GET 메서드의 경우 브라우저 상에서 해당 주소로 접속하면 작동 결과를 확인할 수 있지만,
POST 메서드의 경우 POST 전송 어플리케이션을 만들지 않으면 테스트를 할 수 없다.
그래서 REST API를 간단하게 테스트할 수 있게 해주는 툴을 사용.



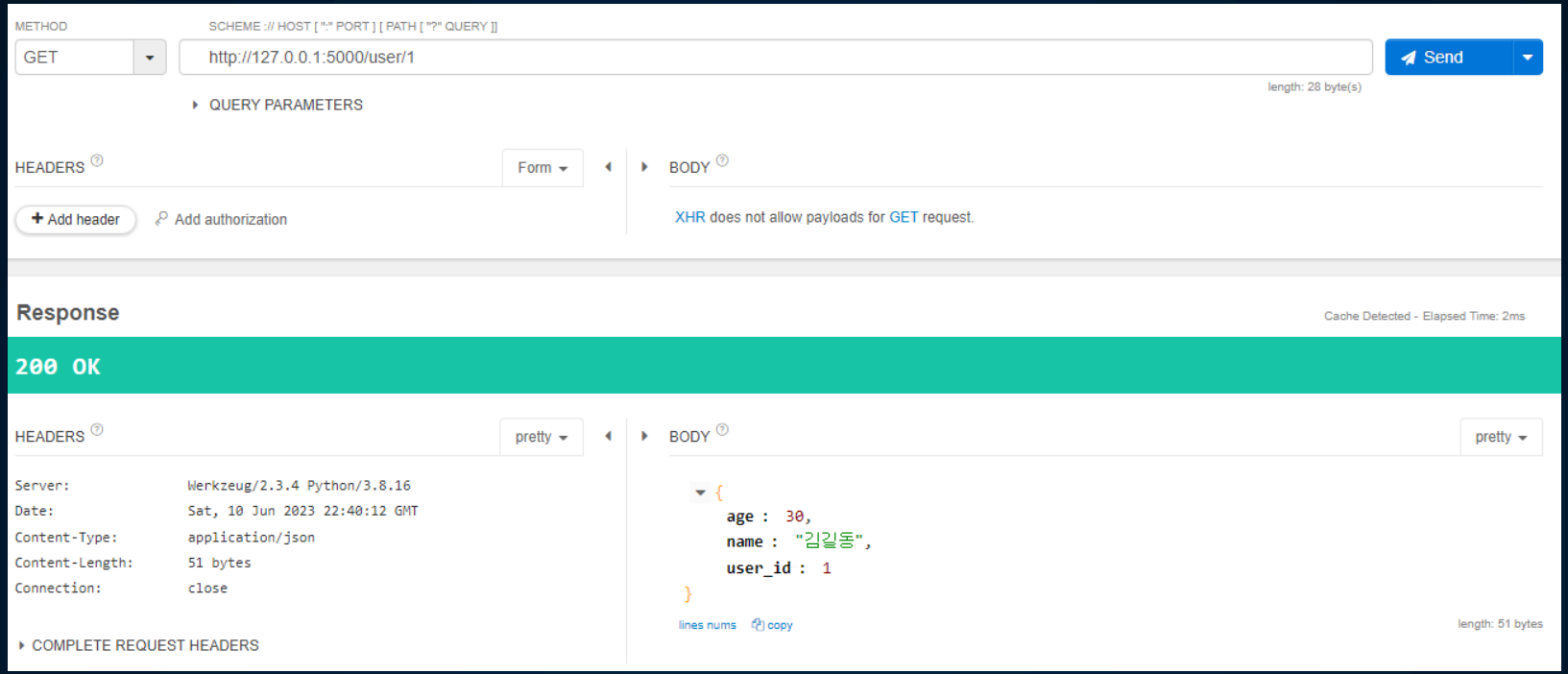
Talend API Tester 실행,

METHOD – POST
URI – http://127.0.0.1:5000/user

BODY 항목은 사진과 같이 입력

POST 메서드 실행 후, 페이지의 아래쪽을 보면

REST API 서버에서 받은 응답을 보여준다.



이번엔 추가한 회원정보를 조회하는 API를 호출한다.
METHOD – GET
URI – http://127.0.0.1:5000/user/1

여기까지 파이썬에서 기본적인 REST API 서버를 구현하는 방법을 알아보았다.

REST API 호출 시 챗봇 엔진 서버에 소켓 통신으로 접속해 질의에 대한 답변을 받아오는 API 서버를 만들어보자.

구현 내용에는 (카카오톡, 네이버톡톡) 의 메신저와 연동할 것을 고려한 코드도 있다.

app.py

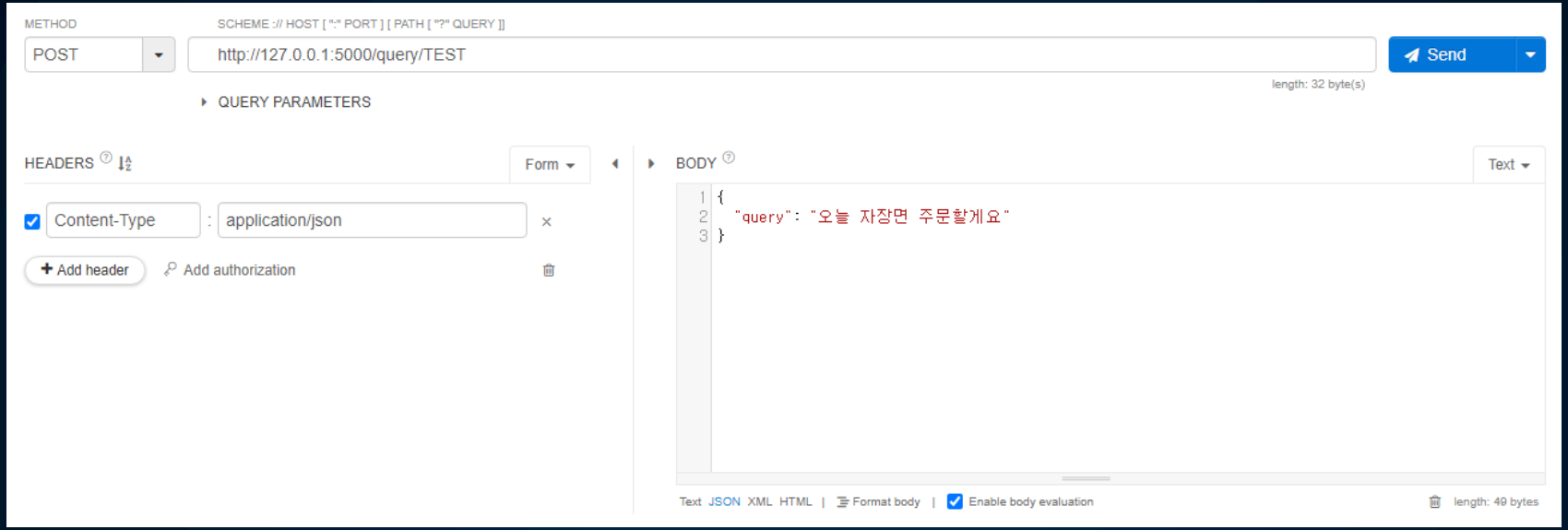
chatbot/chatbot_api폴더 생성
chatbot_api 폴더 안에 파일 생성

```python
from flask import Flask, request, jsonify, abort
import socket
import json

# 챗봇 엔진 서버 접속 정보
host = "127.0.0.1"  # 챗봇 엔진 서버 IP 주소
port = 5050  # 챗봇 엔진 서버 통신 포트

# Flask 어플리케이션
app = Flask(__name__)
```

```python
# 챗봇 엔진 서버와 통신
def get_answer_from_engine(bottype, query):
    # 챗봇 엔진 서버 연결
    mySocket = socket.socket()
    mySocket.connect((host, port))

    # 챗봇 엔진 질의 요청
    json_data = {
        'Query': query,
        'BotType': bottype
    }
    message = json.dumps(json_data)
    mySocket.send(message.encode())

    # 챗봇 엔진 답변 출력
    data = mySocket.recv(2048).decode()
    ret_data = json.loads(data)

    # 챗봇 엔진 서버 연결 소켓 닫기
    mySocket.close()

    return ret_data
```

```python
@app.route('/', methods=['GET'])
def index():
    print('hello')

# 챗봇 엔진 query 전송 API
@app.route('/query/<bot_type>', methods=['POST'])
def query(bot_type):
    body = request.get_json()

    try:
        if bot_type == 'TEST':
            # 챗봇 API 테스트
            ret = get_answer_from_engine(bottype=bot_type,
query=body['query'])
            return jsonify(ret)

        elif bot_type == "KAKAO":
            # 카카오톡 스킬 처리
            pass
```

```
        elif bot_type == "NAVER":
            # 네이버톡톡 Web hook 처리
            pass
        else:
            # 정의되지 않은 bot type인 경우 404 오류
            abort(404)

    except Exception as ex:
        # 오류 발생시 500 오류
        abort(500)


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



기존에 구현했던 챗봇 엔진 (bot.py)도 실행을 시켜둔 상태에서

METHOD – POST
URI – http://127.0.0.1:5000/query/TEST
BODY
{
“ query”: “오늘 자장면 주문할게요”
}

입력한 값에 따른 의도, 개체명, 답변을 출력한다.