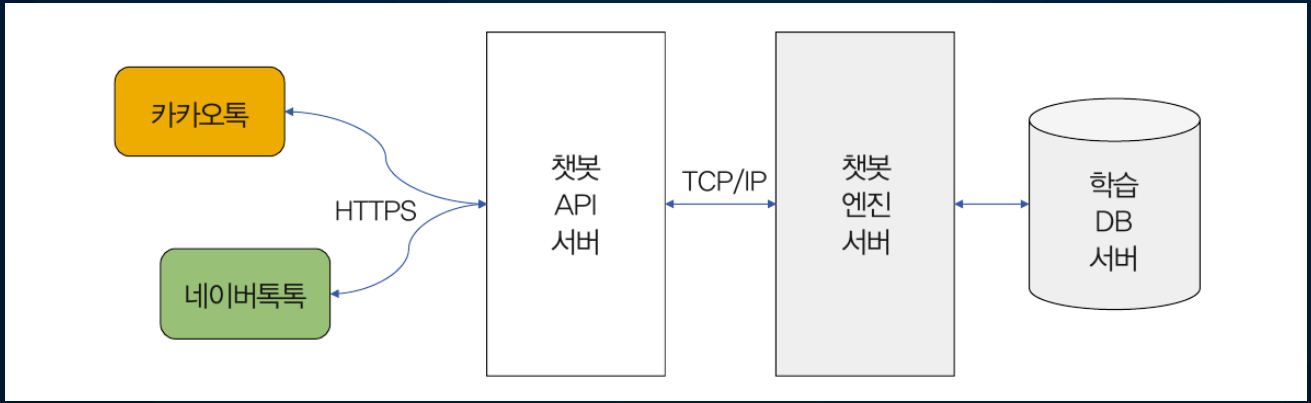
챗봇 프로젝트 12

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.



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

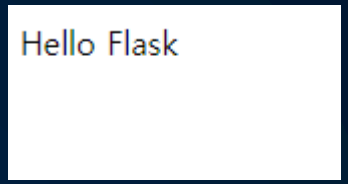
app.py

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
```



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

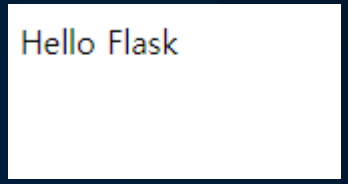
앞에서 작성한 파일에 코드 추가

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

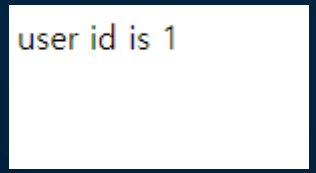
```
from flask import Flask
app = Flask(__name__)

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

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

```
@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

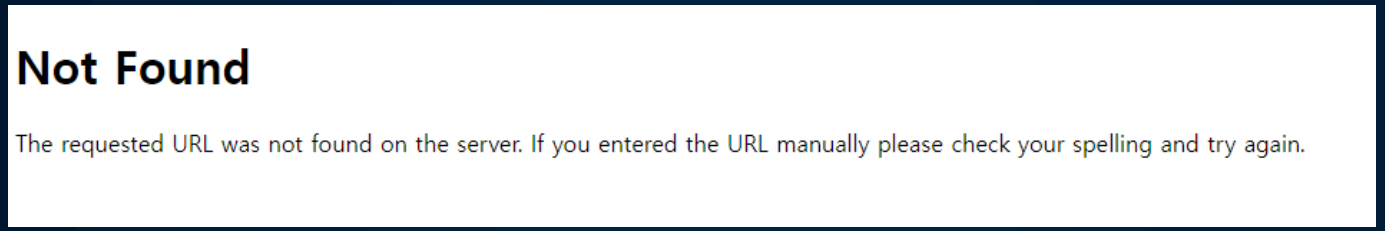
임의의 경로에 main_flask 폴더를 생성 후, 파일 생성

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'])
```

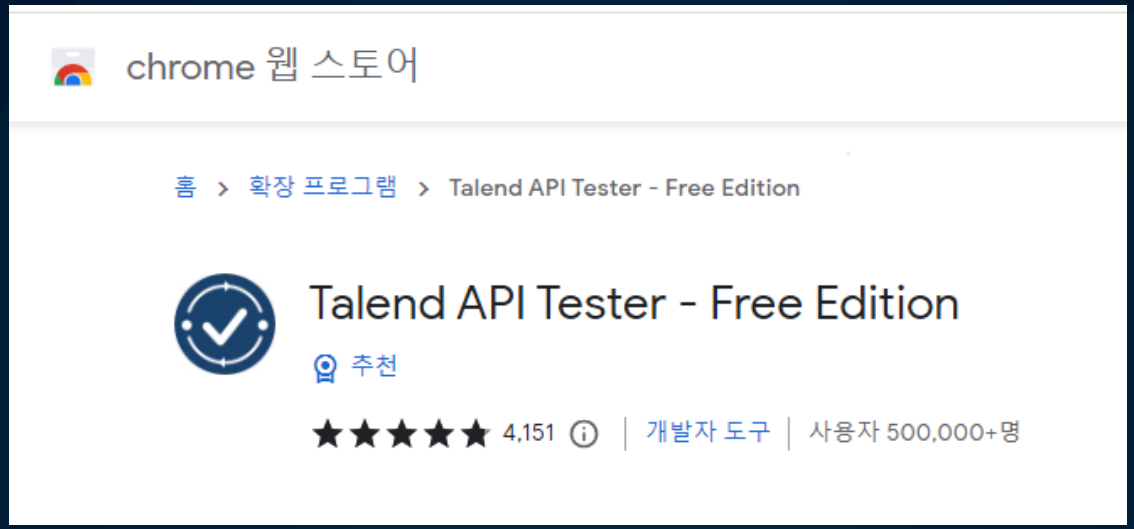
```python
def get_user(user_id):
    for user in resource:
        if user['user_id'] is user_id:
            return jsonify(user)

    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
```



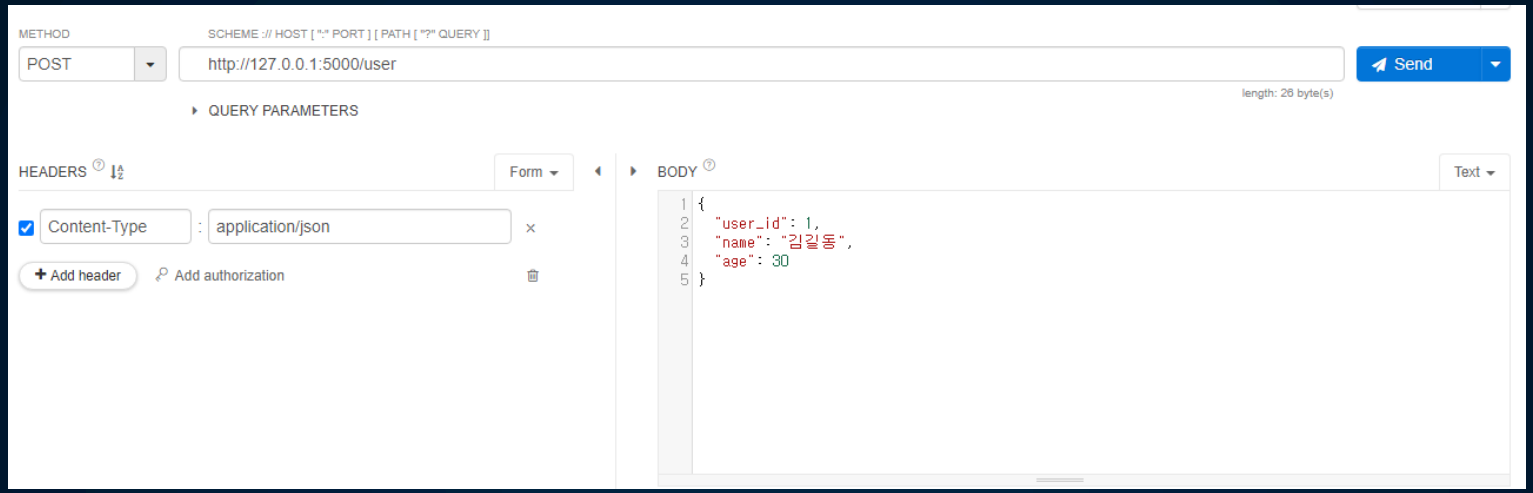
사용자 정보를 조회하는 기능을 구현했지만, 사용자 정보가 존재하지 않기 때문에
Not Found 오류 문구를 출력한다.

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.



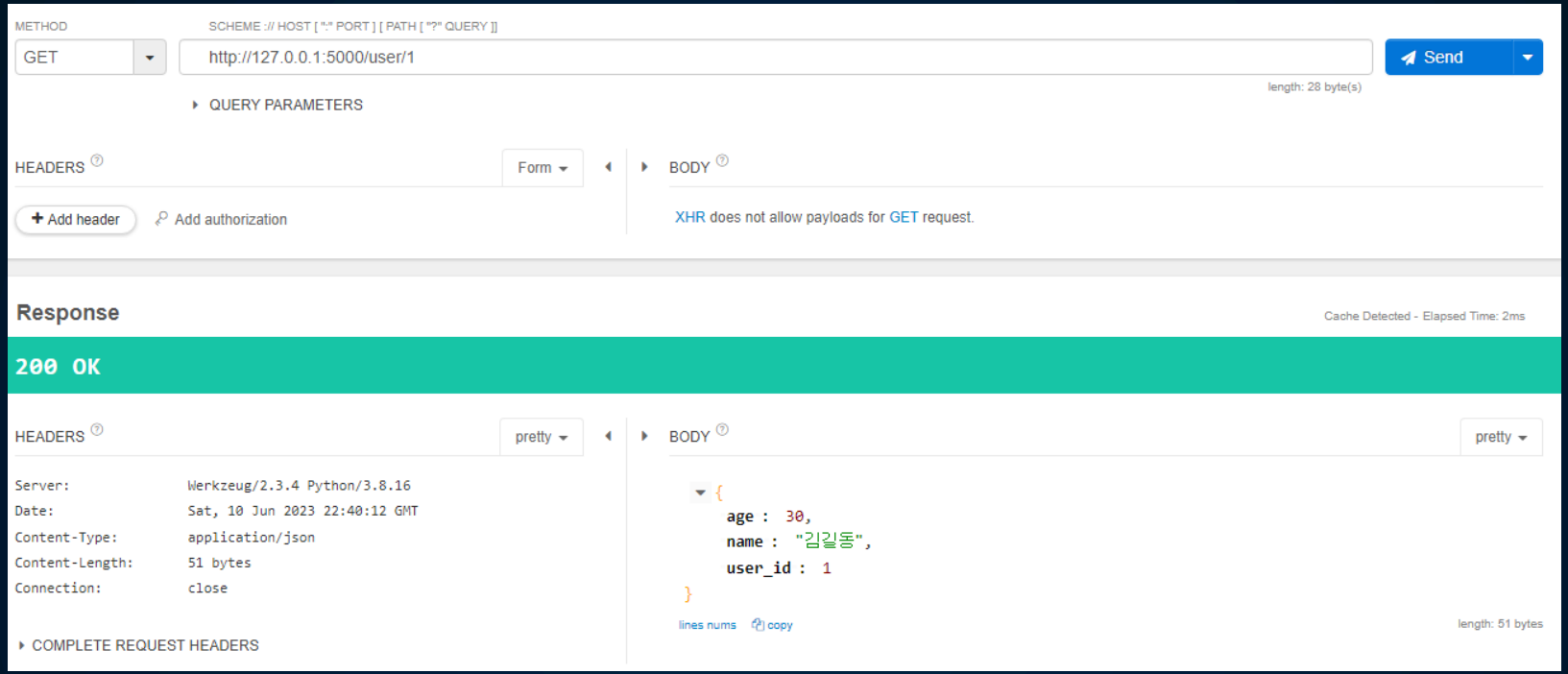
Talend API Tester 실행,

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 네이버톡톡 메신저와 연동할 것을 고려한 코드도 있다.

app.py

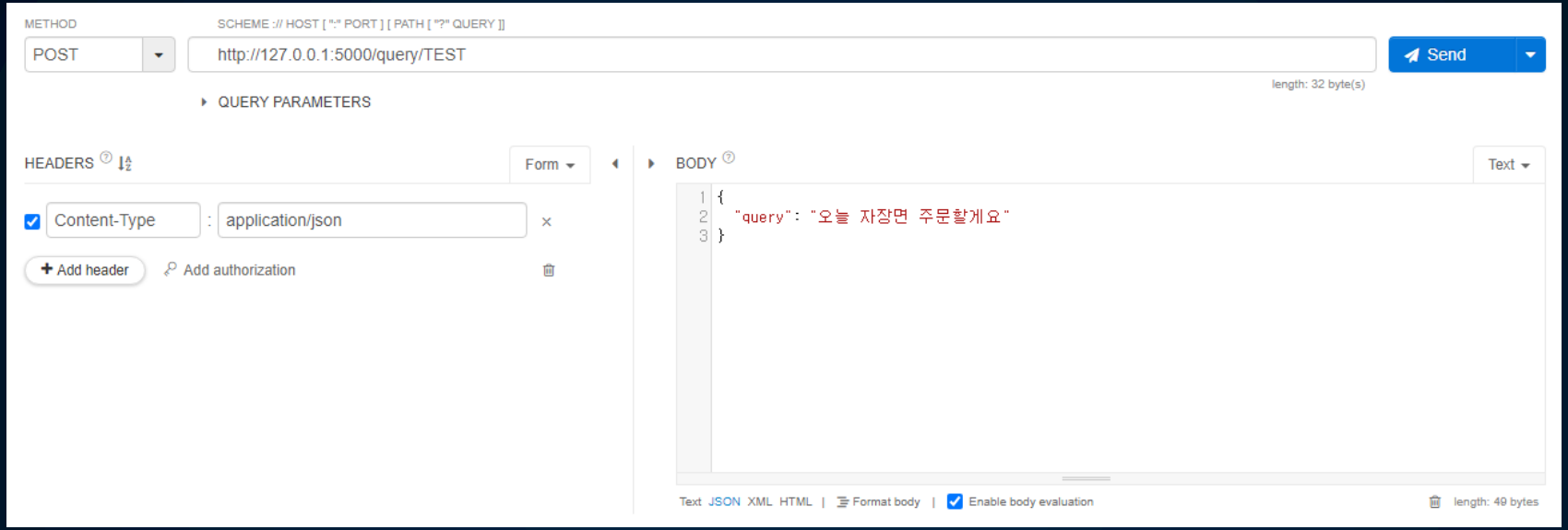
chatbot/chatbot_api폴더 생성
chatbot_api 폴더 안에 파일 생성

```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 == "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.

네이버톡톡 API 설정

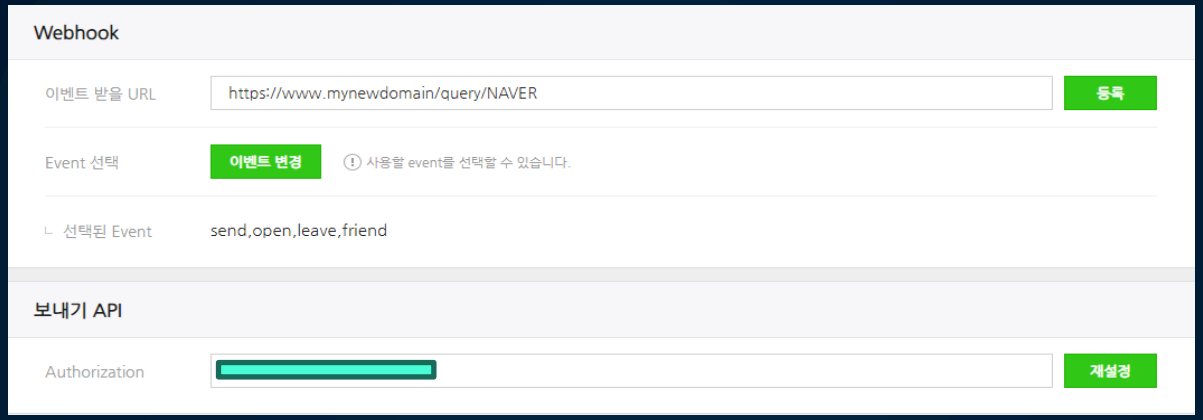
partner.talk.naver.com 접속

좌측 메뉴 〉 개발자도구 〉 챗봇API 설정

최초 접속 시, API 활용 신청을 위해

업체명, 업체홈페이지, 사용목적, 이름, 전화번호 등의 정보를 입력해야 되지만
내용을 적당히 채워넣으면 따로 승인 절차 없이 사용이 가능함.

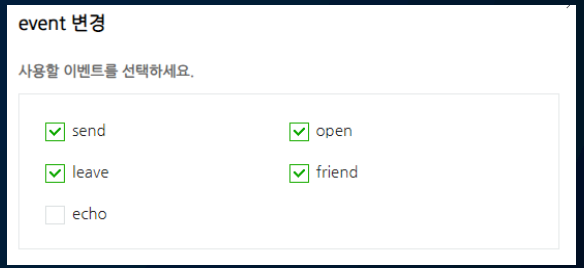
Webhook 설정



URL 받을 링크
일단 비워두거나, 적당한 주소를 입력해둔다.

이벤트 변경 클릭 후,
send, open, leave, friend 이벤트 선택 저장

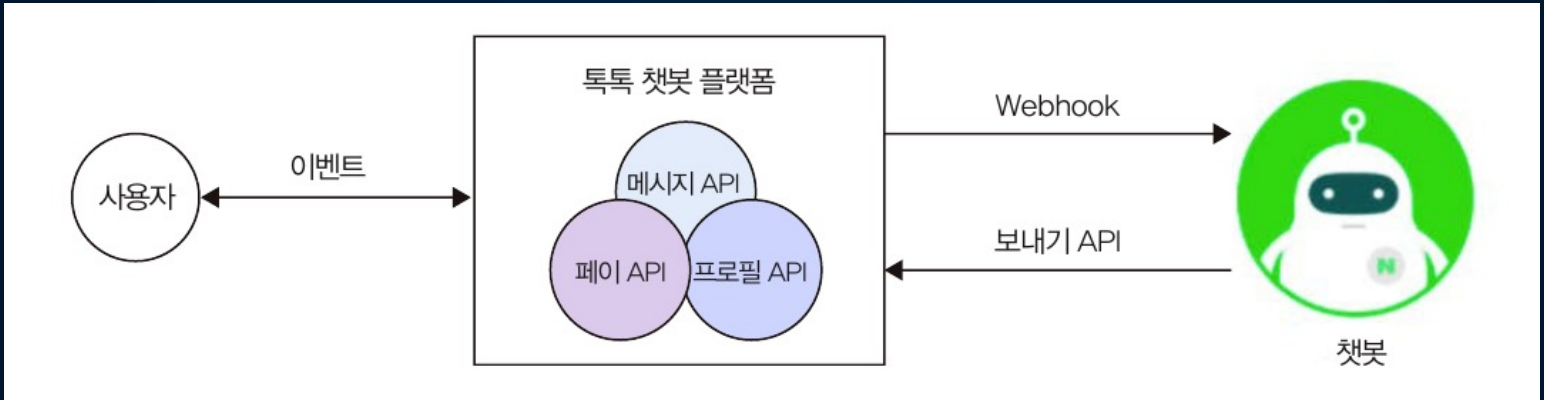
보내기 API Authorization 생성



이벤트	설명
Open	사용자가 채팅창에 진입했을 때 전달되는 이벤트, open 이벤트의 options필드에 사용자 유입 정보가 포함되어 있다.
Leave	사용자가 채팅창에서 나가기를 누르면 발생하는 이벤트. 사용자가 채팅방을 나가는 시점에서 확인 할 수 있다.
Friend	사용자가 톡톡 계정에서 친구추가 또는 친구철회를 누르면 발생하는 이벤트. friend 이벤트의 options필드에 친구추가 및 친구철회 정보가 포함되어 있다.
send	Send 이벤트는 사용자와 챗봇이 메시지를 전송할 때 발생하는 이벤트다. Send 이벤트의 메시지 타입에 따라 챗봇에 표현되는 말풍선 모양이 달라진다.

톡톡에서 이벤트가 발생했을 때
API로 전달되는 데이터는 다음과 같다.

네이버톡톡 챗봇 API



네이버 톡톡에서 제공하는 챗봇 API의 구조.

사용자로부터 이벤트가 발생할 때마다 챗봇 플랫폼은 지정한 Webhook을 호출한다.
챗봇은 보내기 API를 통해 사용자에게 자유롭게 메시지를 전달할 수 있다.

톡톡에서 이벤트가 발생했을 때
API로 전달되는 데이터의 예시는 다음과 같다.

```
{
    "event": "send",
    "user": "<사용자 식별값>",
    "textContent": {
        "text": "안녕",
        "inputType": "typing "
    }
}
```

사용자 -> 챗봇
텍스트 메시지 전송

```
{
    "event": "send",
    "textContent": {
        "text": "네, 저도 반가워요.",
    }
}
```

챗봇 -> 사용자
텍스트 메시지 전송

```
{
    "event": "send",
    "user": "<사용자 식별값>",
    "imageContent": {
        "imageUrl": "<네이버 서버에 저장된 이미지 URL>",
        "width": "<이미지 너비 >",
        "height": "<이미지 높이 >",
    }
}
```

사용자 -> 챗봇
이미지 메시지 전송

```
{
    "event": "send",
    " imageContent ": {
        " imageUrl": "<사용자에게 전송할 이미지 URL>",
    }
}
```

챗봇 -> 사용자
이미지 메시지 전송

```
{
  "event": "open",
  "user": "al-2eGuGr5WQOnco1_V-FQ",
  "options": {
      "inflow": "list",
      "referer": "https://talk.naver.com/",
      "friend": false,
      "under14": false,
      "under19": false
  }
}
```


채팅창에 진입한 경우


```
{
  "event": "send",
  "user": "al-2eGuGr5WQOnco1_V-FQ",
  "textContent": {
      "text": "hello world",
      "inputType": "typing"
  }
}
```


채탕창에 hello world 메시지를 보낸 경우

톡톡 API 연동

네이버톡톡 이벤트를 처리하는 기능을 챗봇 API 서버에 추가

네이버톡톡 응답 이벤트를 생성하는 모듈을 생성한다.
해당 모듈은 톡톡 챗봇 플랫폼으로 전달할 응답 메시지(JSON)를 생성하는 기능을 가지게 된다.

NaverEvent.py

chatbot_api 폴더 안에 파일 생성

```python
import requests, json

class NaverEvent:
    def __init__(self, authorization):
        self.authorization_key = authorization

    # 텍스트 컨텐트 출력 요소
    def textContentComponent(self, text):
        return {
            "textContent": {
                "text": text
            }
        }
```

```
        # 보내기 API 호출
        message = json.dumps(data)  # JSON 문자열 변경
        return requests.post(
            'https://gw.talk.naver.com/chatbot/v1/event',
            headers=headers,
            data=message)

    # 사용자에게 응답 전송
    def send_response(self, dst_user_key, bot_resp):
        # 이미지 답변이 텍스트 답변보다 먼저 출력 됨
        # 이미지 답변이 있는 경우
        if bot_resp['AnswerImageUrl'] is not None:
            image = self.imageContentComponent(bot_resp['AnswerImageUrl'])
            self.send_message(user_key=dst_user_key, component=image)

        # 텍스트 답변이 있는 경우
        if bot_resp['Answer'] is not None:
            text = self.textContentComponent(bot_resp['Answer'])
            self.send_message(user_key=dst_user_key, component=text)

        return json.dumps({}), 200
```

```python
# 이미지 컨텐트 출력 요소
def imageContentComponent(self, imageUrl):
    return {
        "imageContent": {
            "imageUrl": imageUrl
        }
    }

# 보내기 API로 메시지 전송
def send_message(self, user_key, component):
    headers = {
        'Content-Type': 'application/json;charset=UTF-8',
        'Authorization': self.authorization_key,
    }

    data = {
        "event": "send",
        "user": user_key,
    }
    data.update(component)
```

app.py

chatbot_api 폴더 안에 만들었던, API서버를 구현한 app.py에
네이버톡톡 이벤트 처리 코드를 추가한다.

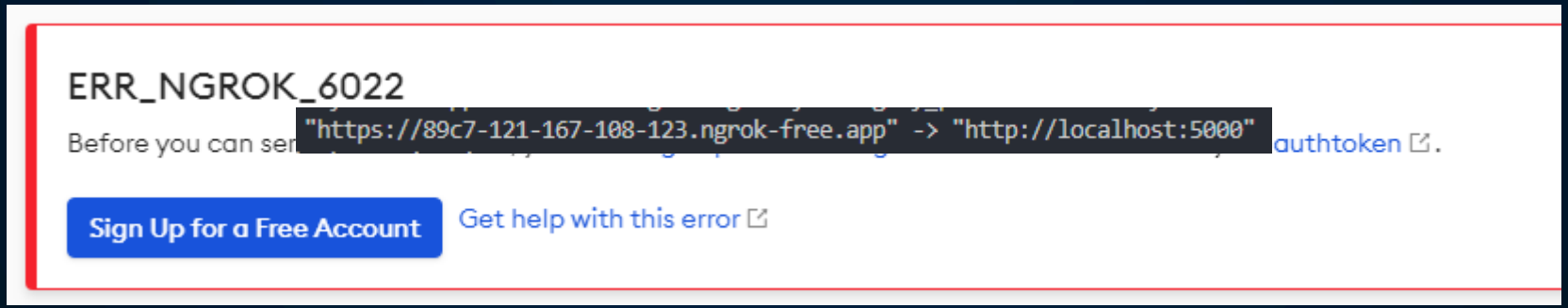
```
    try:
        if bot_type == 'TEST':
            # 챗봇 API 테스트
            ret = get_answer_from_engine(bottype=bot_type,
query=body['query'])
            return jsonify(ret)

        elif bot_type == "NAVER":
            # 네이버톡톡 이벤트 처리
            body = request.get_json()
            user_key = body['user']
            event = body['event']

            from NaverEvent import NaverEvent
            authorization_key = '<보내기 API 인증키>'
            naverEvent = NaverEvent(authorization_key)
```

```
if event == "open":
                # 사용자가 채팅방 들어왔을 때 처리
                print("채팅방에 유저가 들어왔습니다.")
                return json.dumps({}), 200

            elif event == "leave":
                # 사용자가 채팅방 나갔을 때 처리
                print("채팅방에 유저가 나갔습니다.")
                return json.dumps({}), 200

            elif event == "send":
                # 사용자가 챗봇에게 send 이벤트를 전송했을 때
                user_text = body['textContent']['text']
                ret = get_answer_from_engine(bottype=bot_type,
query=user_text)
                return naverEvent.send_response(user_key, ret)

            return json.dumps({}), 200
```

pyngrok

구현한 챗봇은 아직 localhost 환경에서만 구현이 되기 때문에
이것을 실제 서비스와 연동하기 위해서는 임시 도메인을 얻는 작업이 필요하다.
pyngrok 서비스를 이용하면 이를 임시적으로 해결이 가능하다.

pip install pyngrok



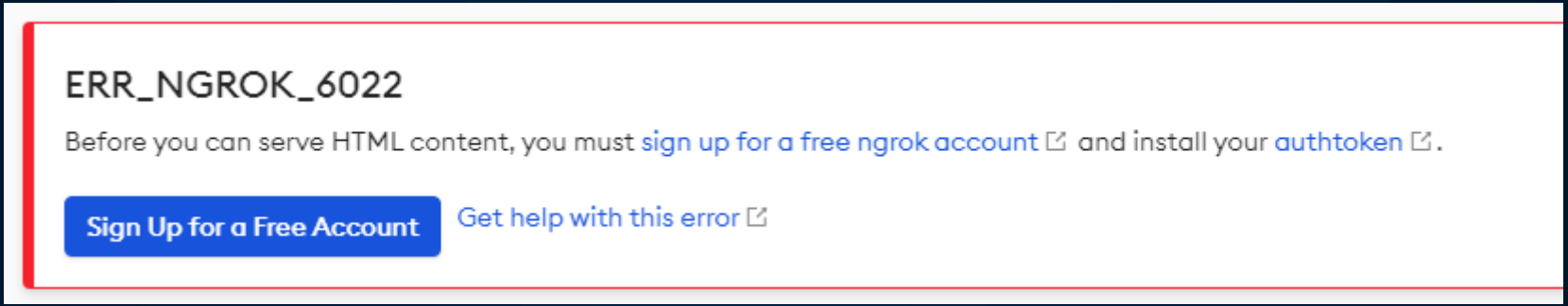
app.py

코드 수정

```
from flask import Flask, request, jsonify, abort
from pyngrok import conf, ngrok
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__)

conf.get_default().auth_token = {본인의 인증키}
http_tunnel = ngrok.connect(5000) # ngrok 시작 및 Port 번호 전달
tunnels = ngrok.get_tunnels() # ngrok forwording 정보

for kk in tunnels: # Forwording 정보 출력
    print(kk)
```

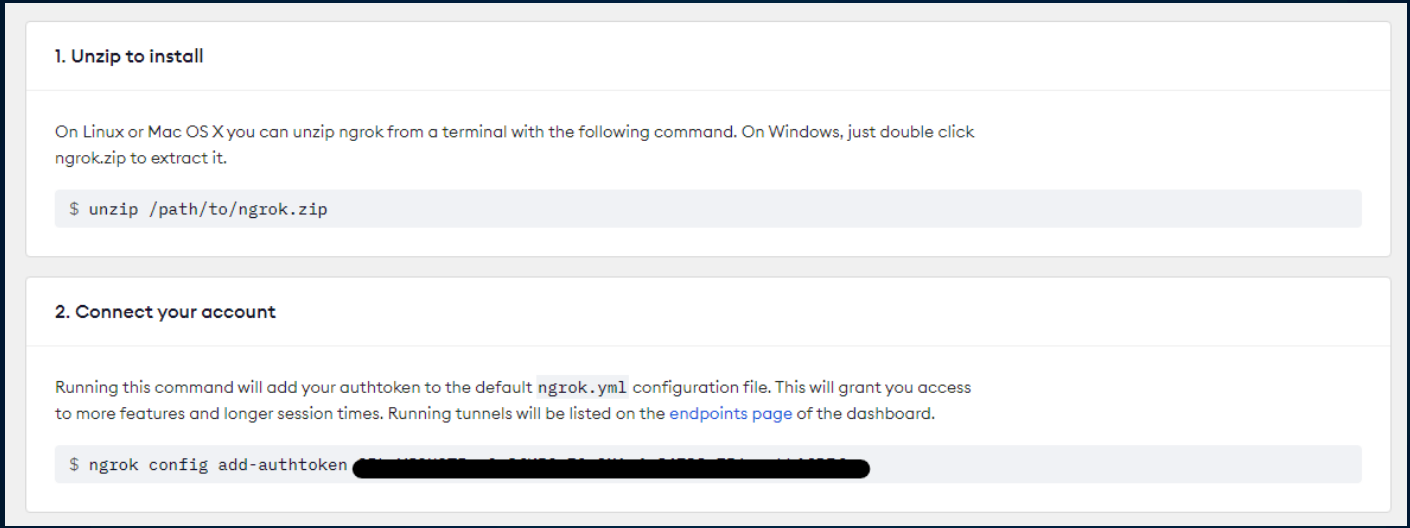
```
"https://89c7-121-167-108-123.ngrok-free.app" -> "http://localhost:5000"
```

코드 수정 후 실행 시, http://localhost:5000를 대체하는 임시 주소 생성



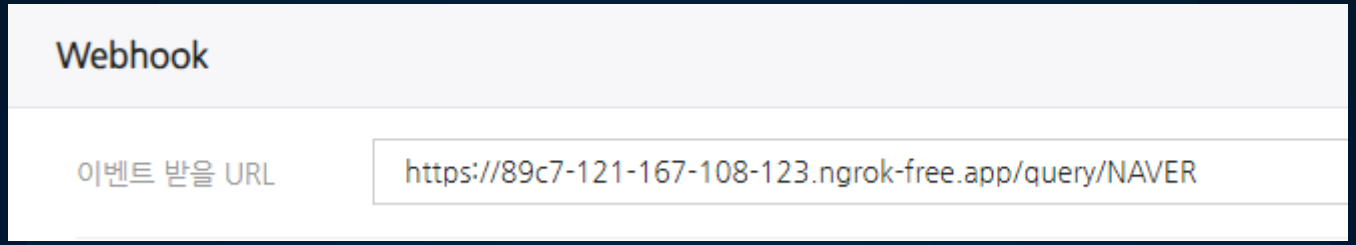
접속 후 초기 1회 회원 가입이 필요하다.

회원가입 완료 시 메인 2번 항목 Connect your account에
authroken을 복사.



app.py에 추가한
`conf.get_default().auth_token = {본인의 인증키}`
부분에 인증토큰 붙여넣기 // { } 는 제외

수정한 후 app.py를 재실행한다.



실행 후, 새롭게 발급받은 주소를 Webhook URL에 등록한다.

등록이 완료되면,

chatbot/bot.py 실행
chatbot/chatbot_api/app.py 실행

챗봇엔진과 톡톡 연동을 위한 api가 작동중인 상태

https://talk.naver.com/{본인챗봇ID} – 주소 접속
챗봇ID : 네이버톡톡 파트너센터 프로필, 챗봇 이름 옆 6글자 숫자+영문자 조합

챗봇엔진과 연동 성공 시, 등록된 메시지가 정상적으로 출력된다.