CHATBOT DEMO

with Slack

Revised @ 2021.09.25

목차

01
SETUP
슬랙 설치와 기본적인
셋업을 알아본다

02
BASICS
챗봇의 기본적인 기능을
사용해본다

03
SLACK “BLOCKS”
슬랙의 " block “
기능에 대해 알아본다

04
DEMO : MOVIE BOT
영화정보를 불러오는
챗봇 데모를 알아본다

SETUP

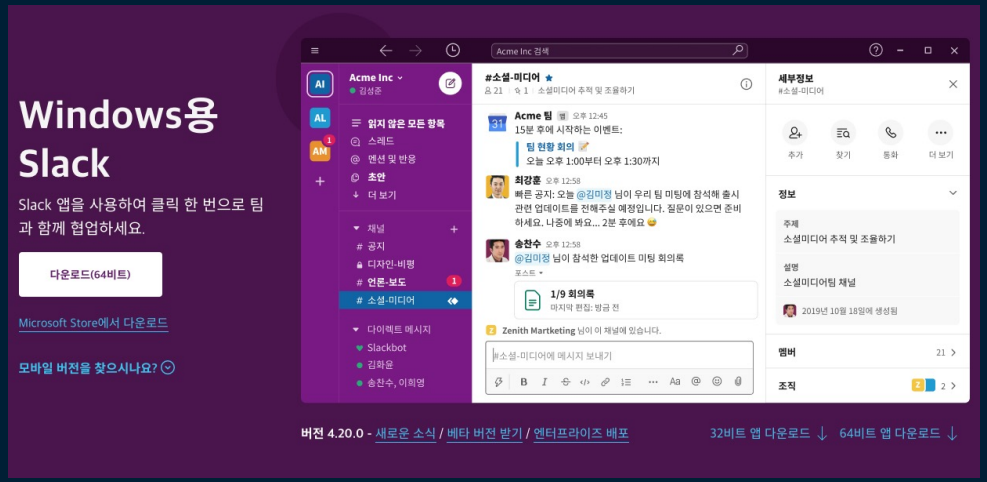
슬랙 설치

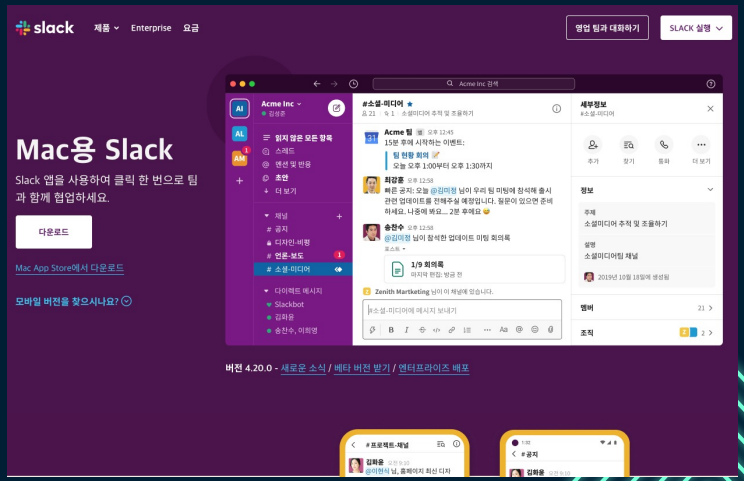
MAC OS:

URL : https://slack.com/intl/ko-kr/downloads/mac?geocode=ko-kr

WINDOWS:

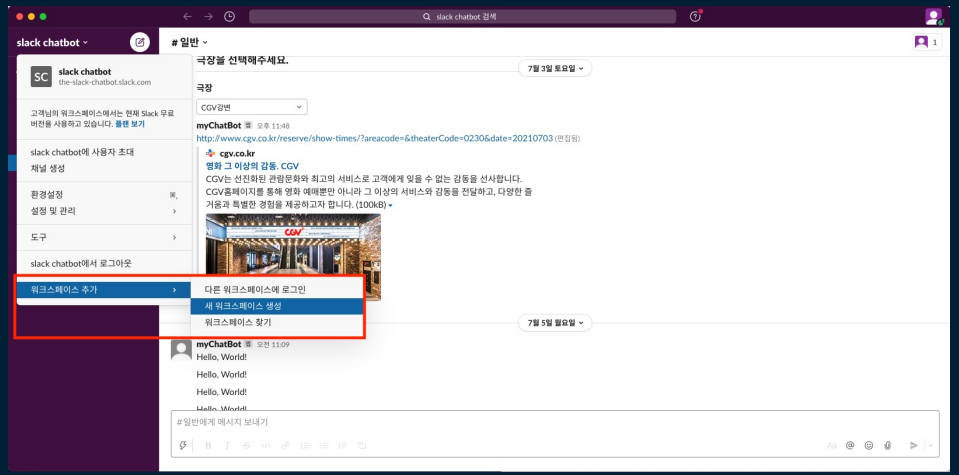
URL : https://slack.com/intl/ko-kr/downloads/windows?geocode=ko-kr



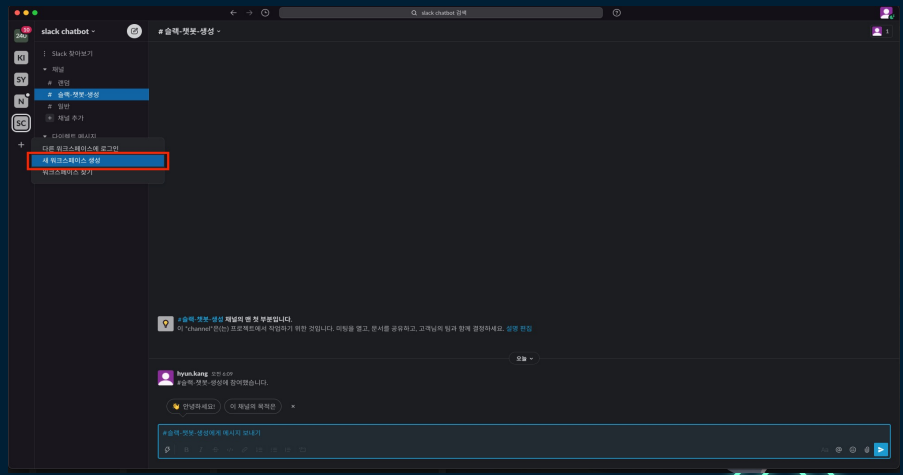


워크스페이스 생성

Side Bar가 없는 경우

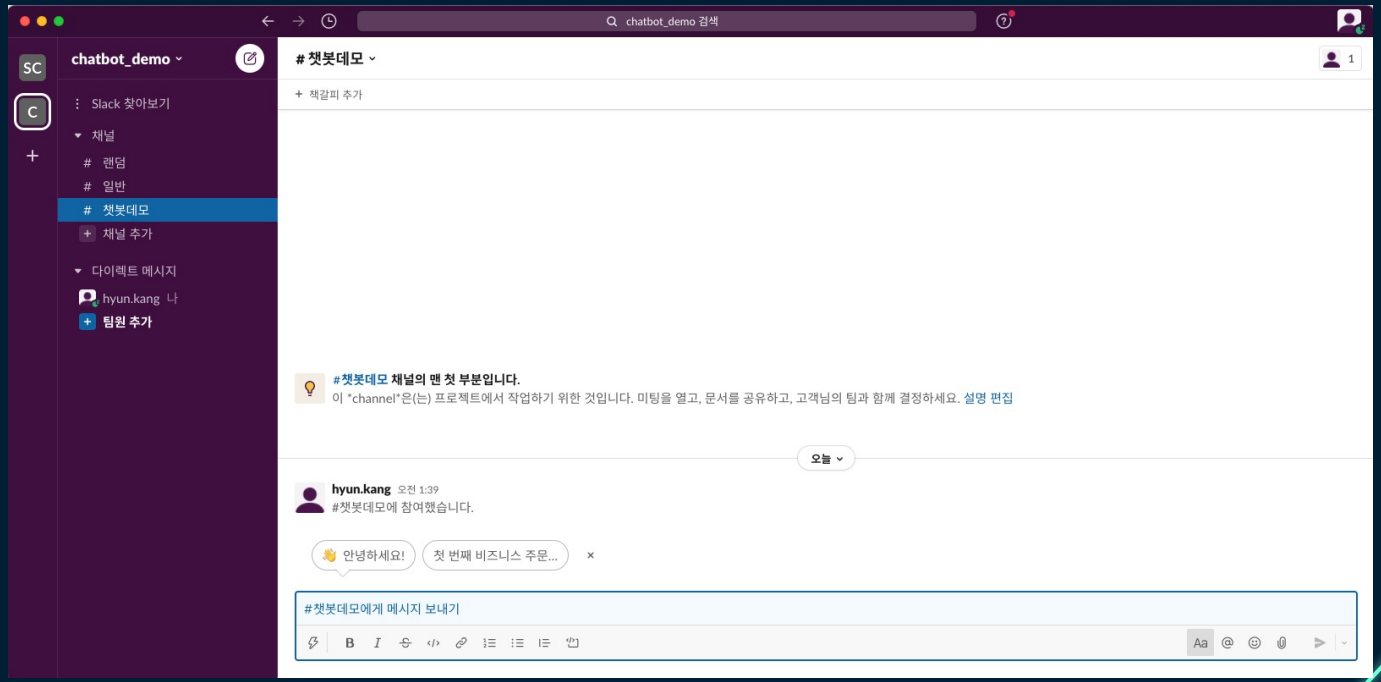


Side Bar가 있는 경우



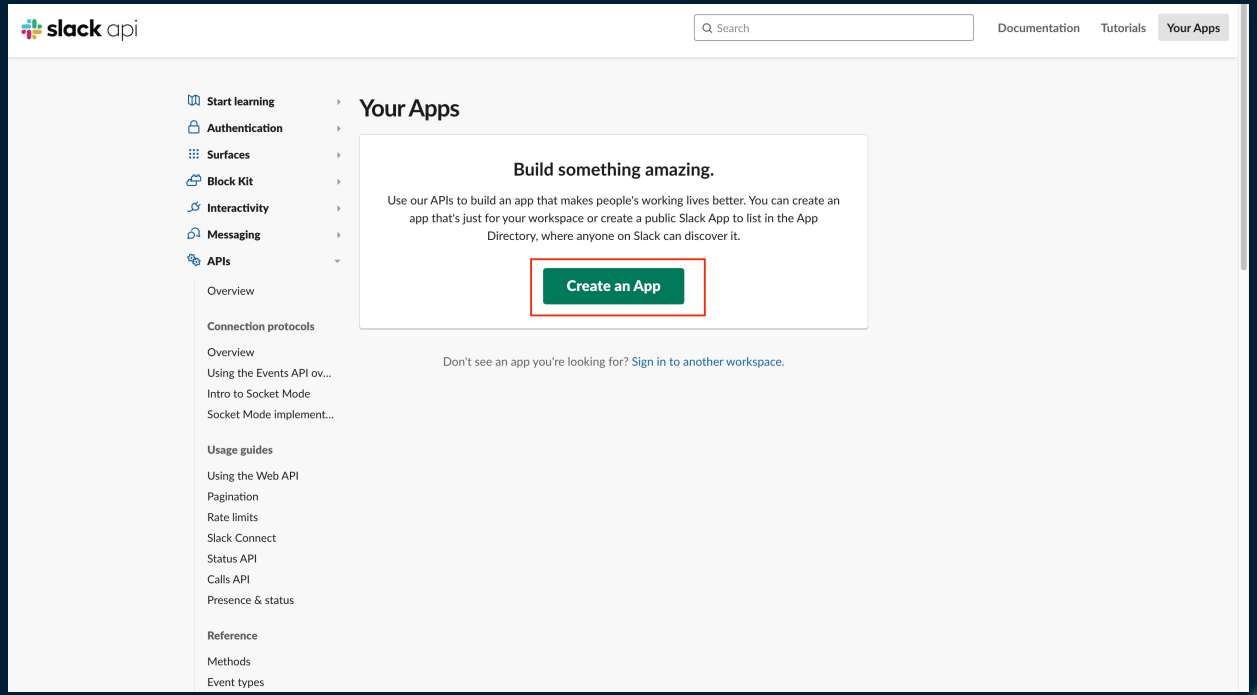
워크스페이스 생성

완성 화면 :



API 세팅

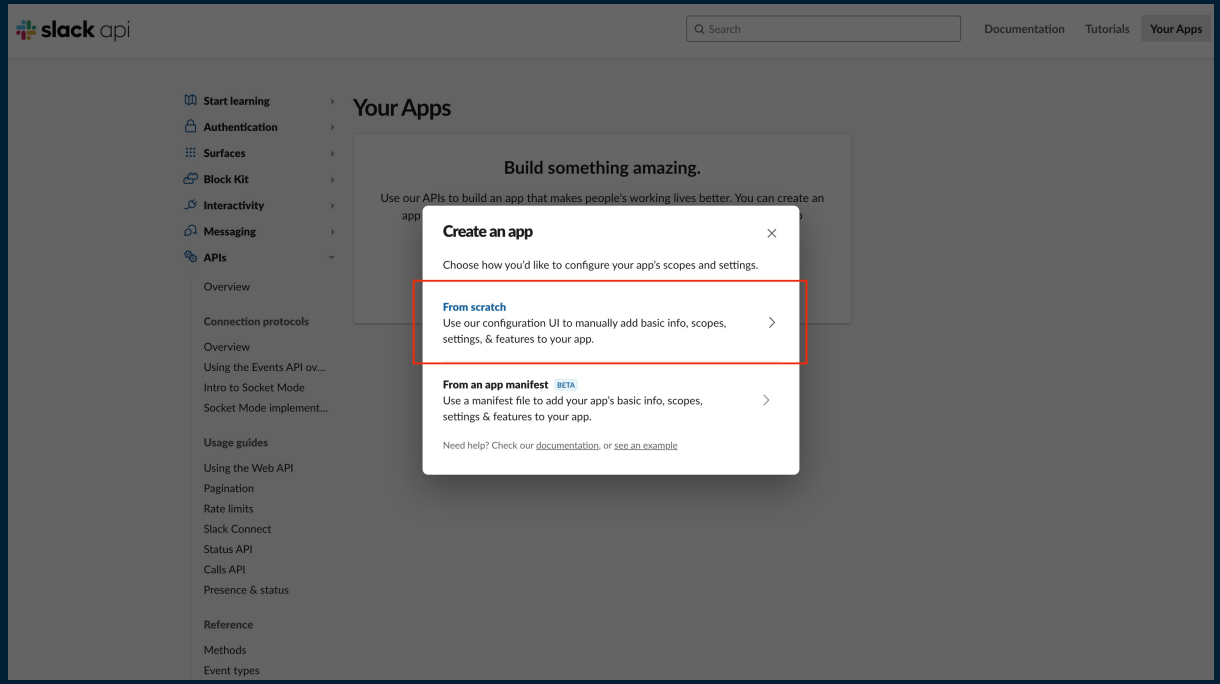
1. SLACK API 홈페이지 접속



URL : https://api.slack.com/apps

API 세팅

2. SLACK APP 생성



URL : https://api.slack.com/apps

API 세팅

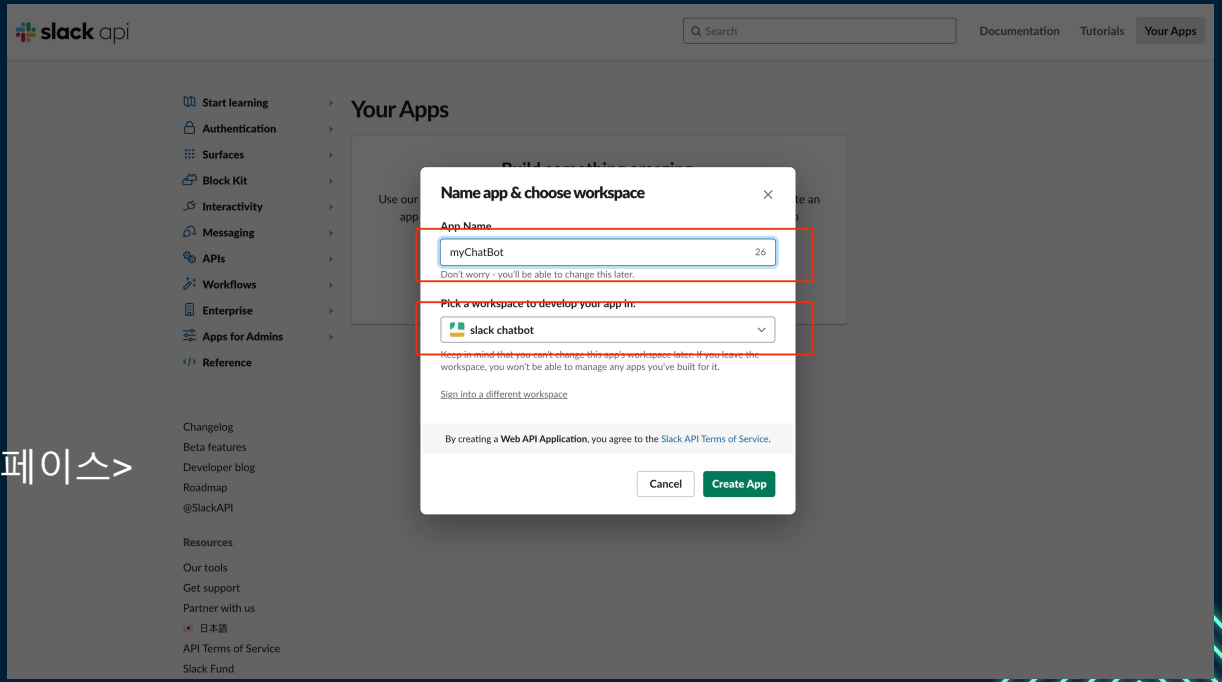
3. SLACK APP 생성

APP NAME :

<임의의 이름>

WORKSPACE :

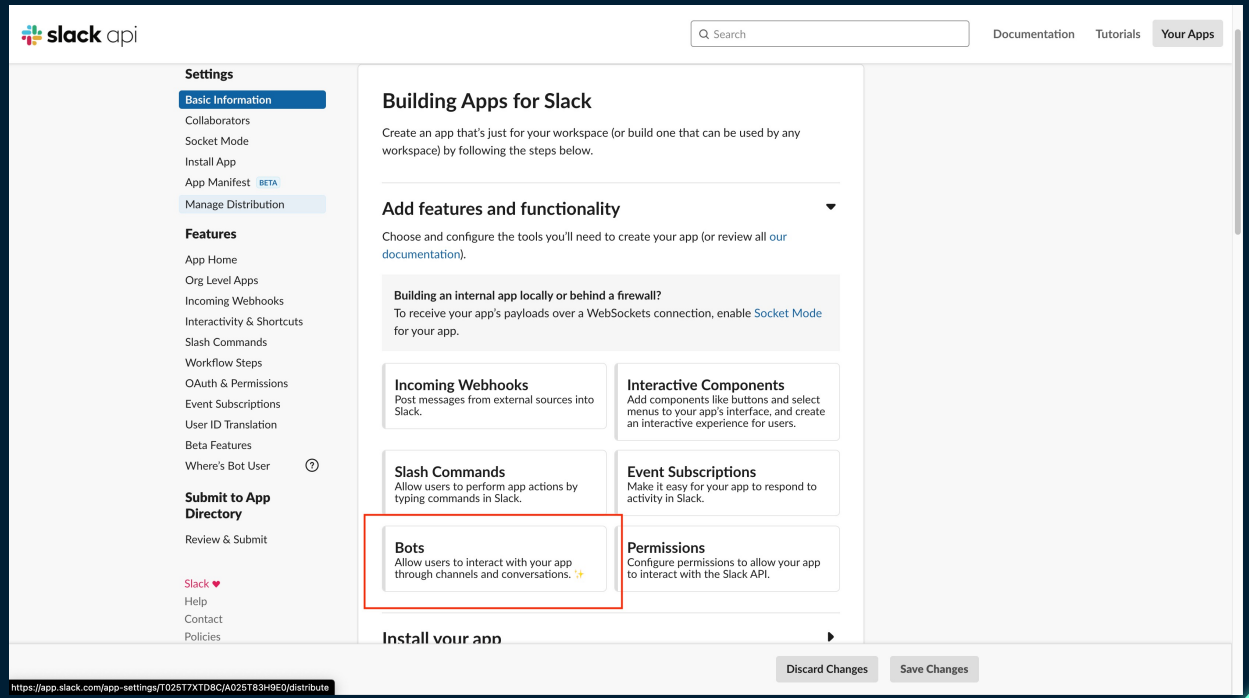
<이전 단계에서 생성한 워크스페이스>



URL : https://api.slack.com/apps

API 세팅

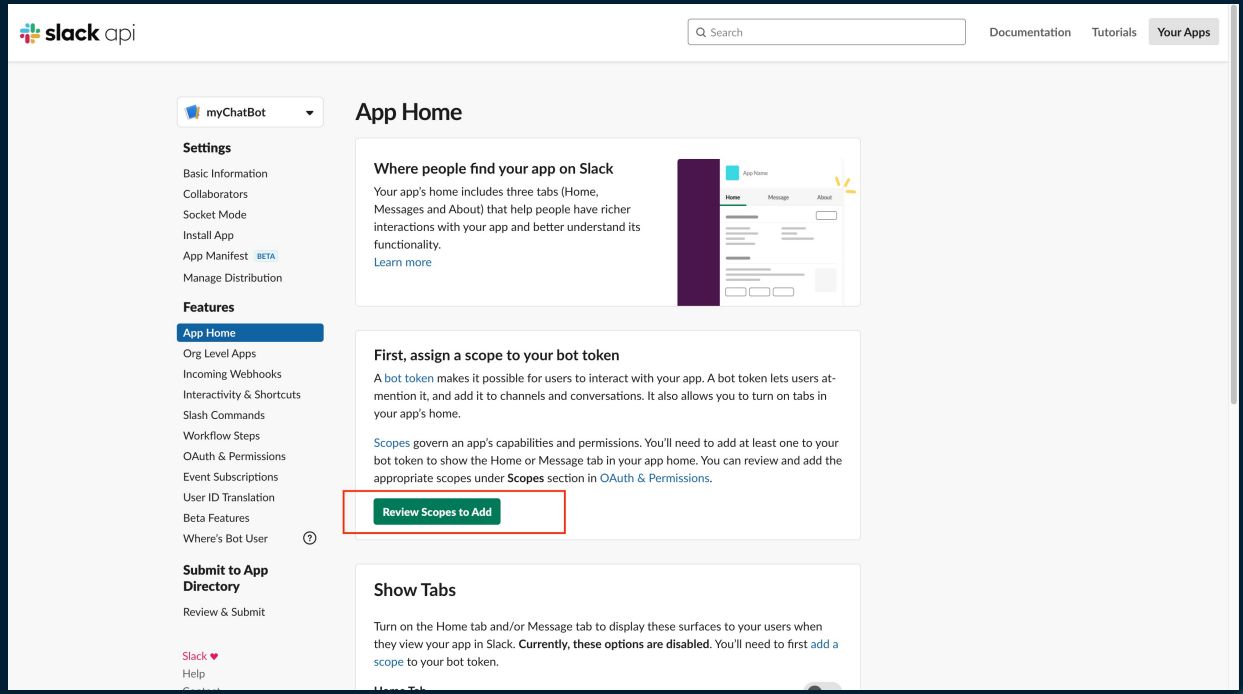
4. BOTS 세팅



URL : https://api.slack.com/apps

API 세팅

4. BOTS 세팅



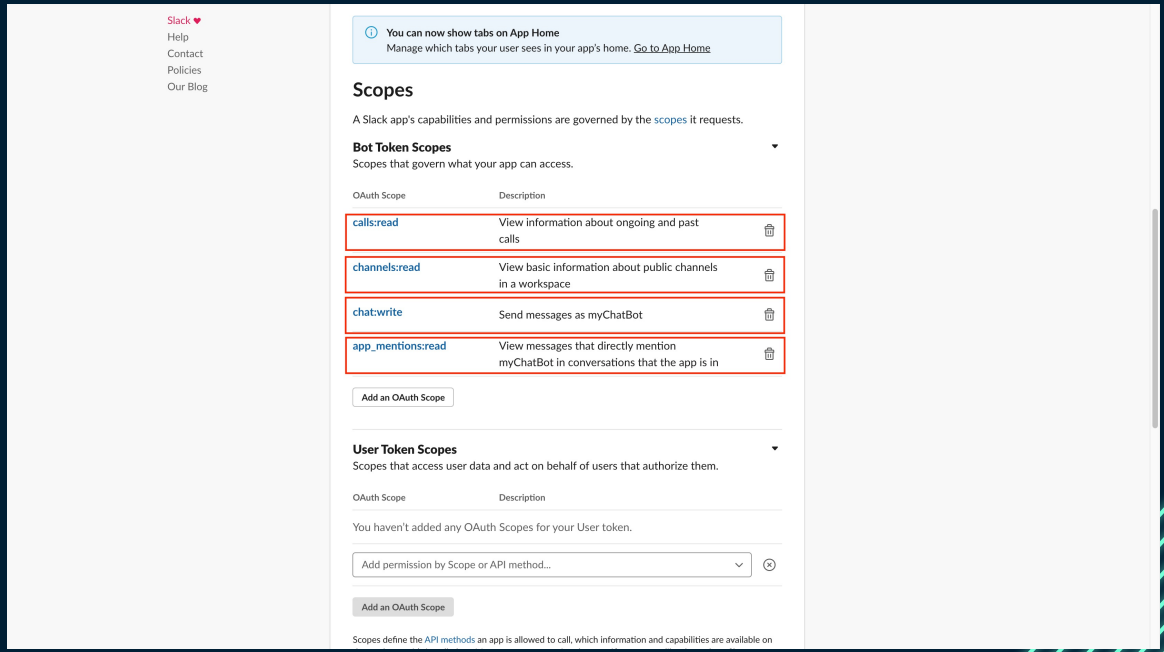
URL : https://api.slack.com/apps

API 세팅

4. BOTS 세팅

BOT TOKEN SCOPES:

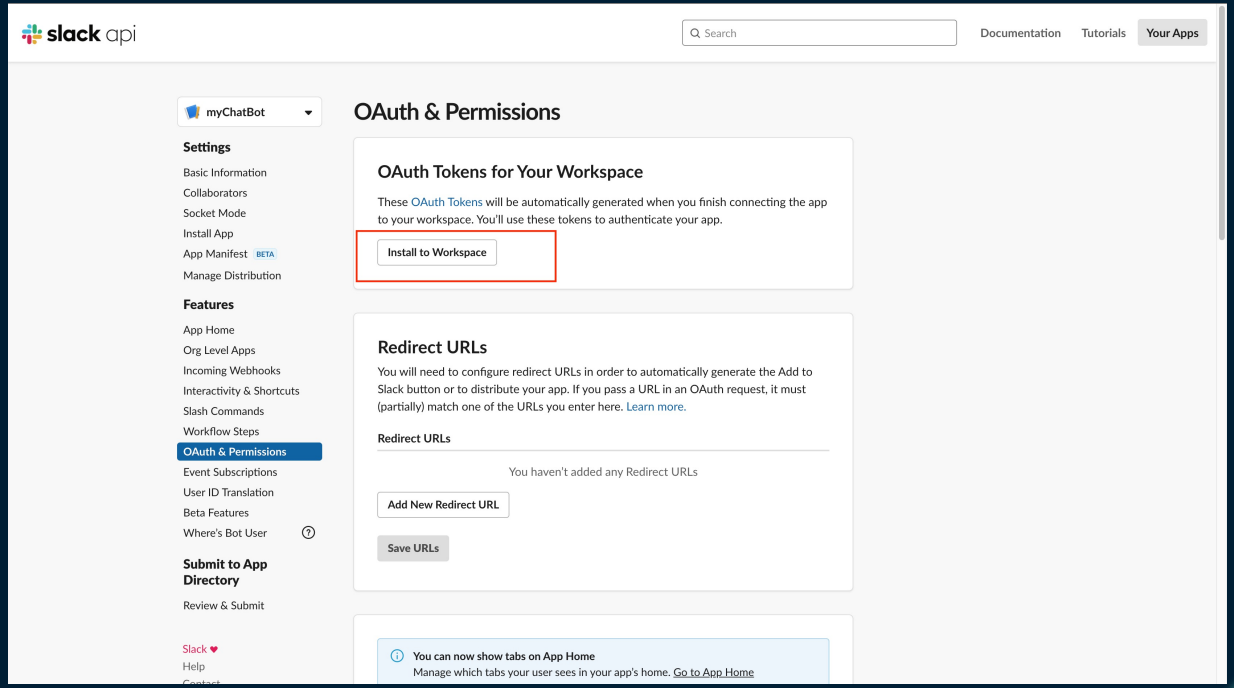
1. calls:read
2. channels:read
3. chat:write
4. app_mentions:read



URL : https://api.slack.com/apps

API 세팅

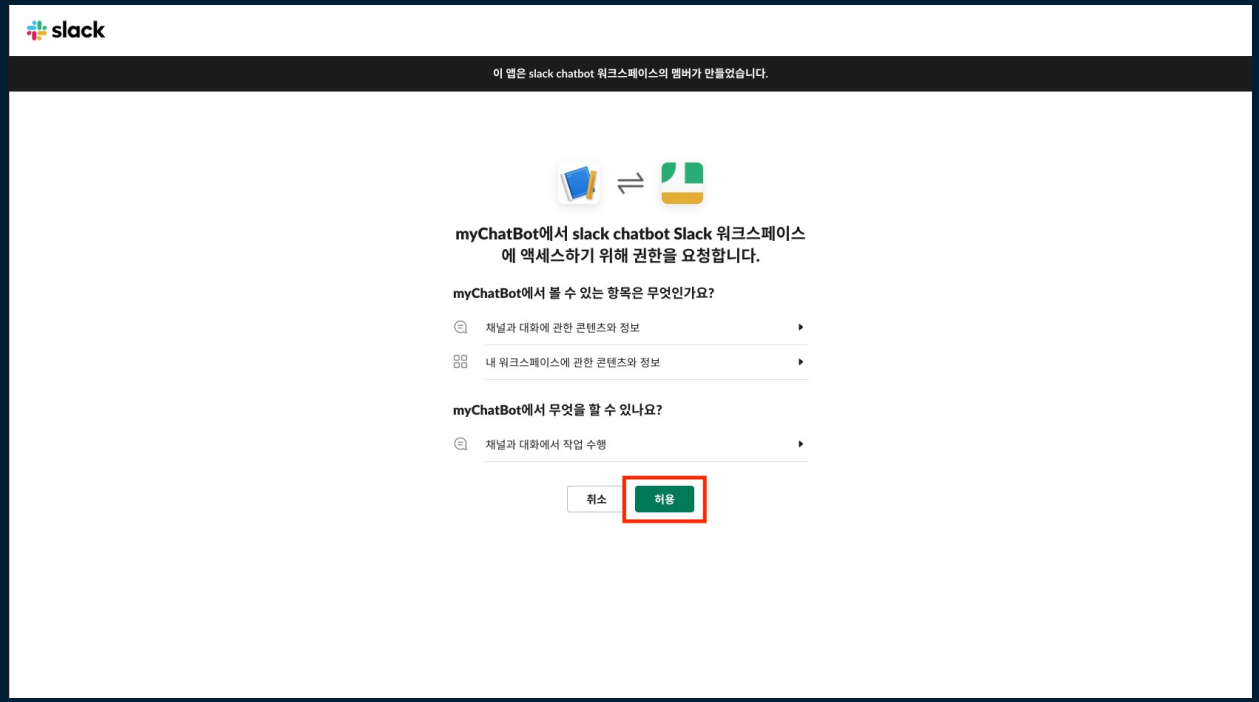
4. BOTS 세팅



URL : https://api.slack.com/apps

API 세팅

4. BOTS 세팅

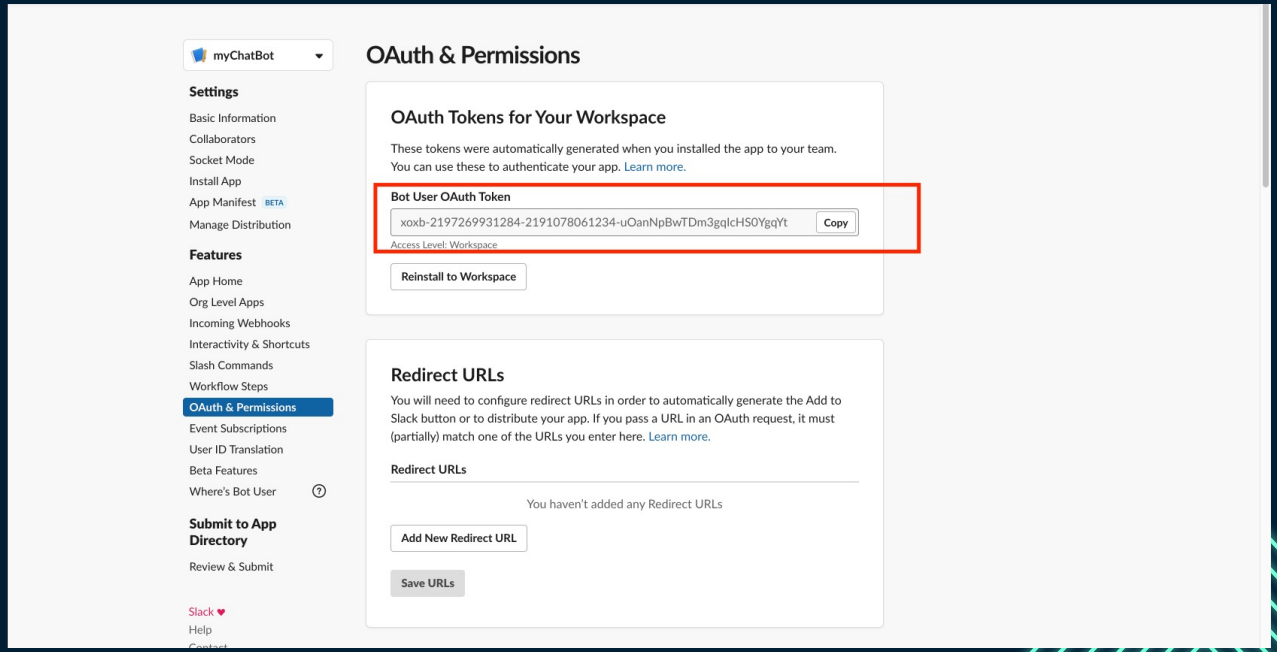


URL : https://api.slack.com/apps

API 세팅

4. BOTS 세팅

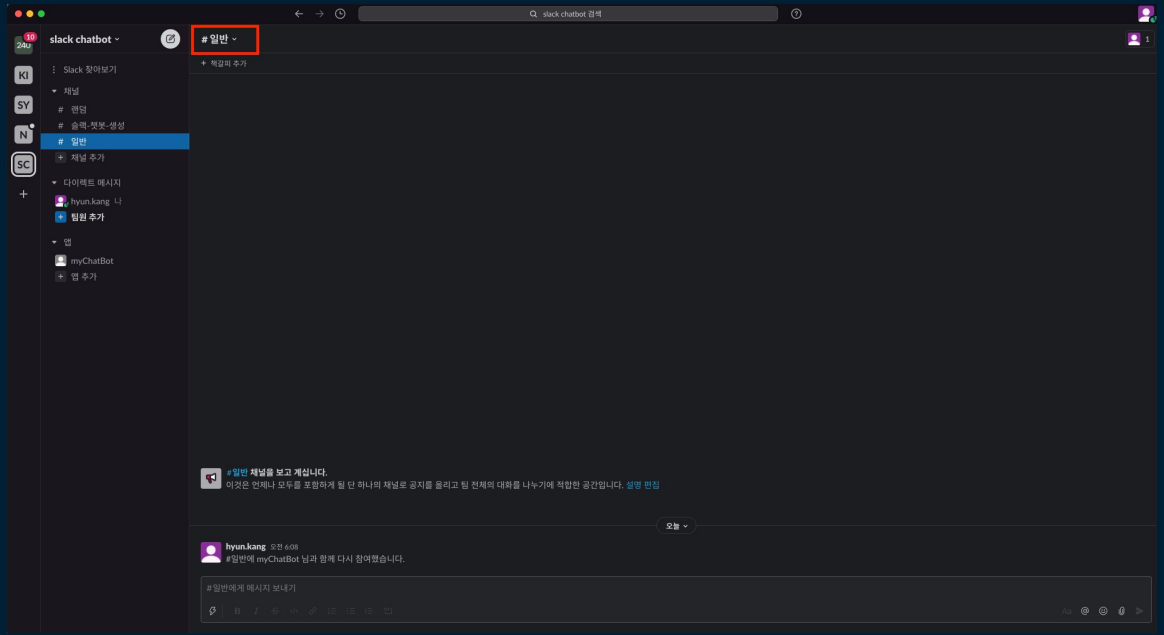
Bot User OAuth Token 저장



URL : https://api.slack.com/apps

API 세팅

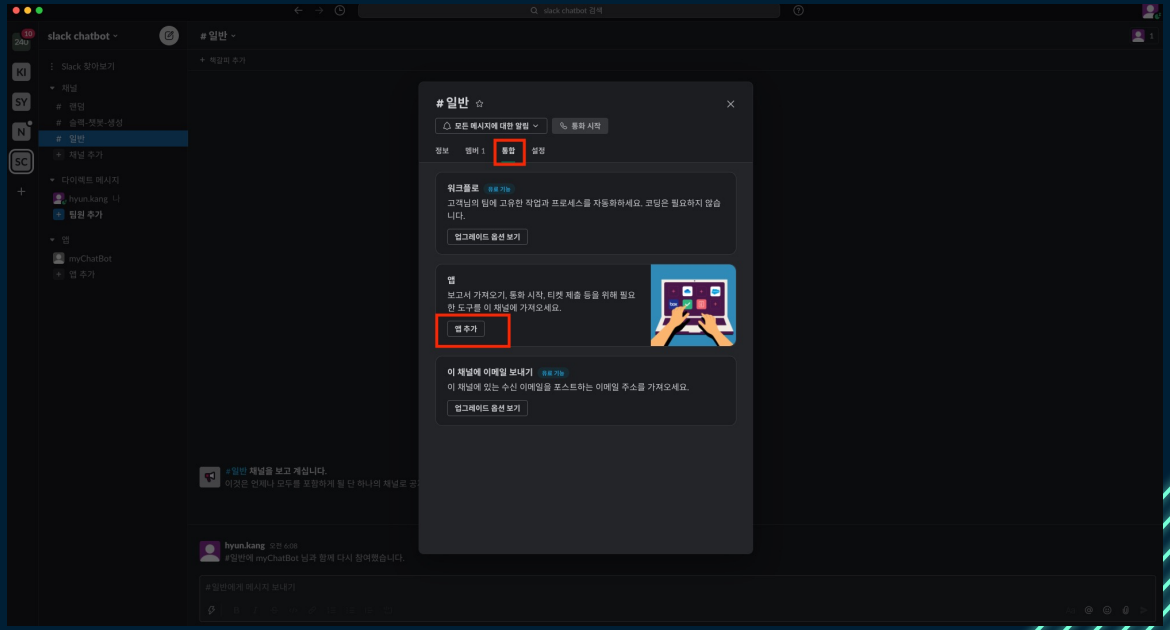
5. Slack App 추가



API 세팅

5. Slack App 추가

1. 통합 탭 클릭
2. 앱 추가
3. 생성한 앱 선택 후 추가



BASICS

라이브러리 설치

URL :
https://drive.google.com/drive/folders/18aAVylX4ku9hefhgJU4KKWpJzfcLUKhK?usp=sharing

```
$ pip install -r requirements.txt
```

환경변수 설정

REFERENCE : .env

.env

```
#.env 파일

# 환경변수를 독립적으로 관리하는 파일

# Development Setting
# 슬랙에서 발급받은 OAuth Tokens을 환경변수로 지정

SLACK_TOKEN = BOT_USER_OAUTH_TOKEN

```

통신모듈 설정

REFERENCE : **handler.py**

```python
# handler.py
# 슬랙과 통신하는 모듈을 정의하는 코드

import os

# pip install python-dotenv
from dotenv import load_dotenv

# pip install slack_sdk
from slack_sdk import WebClient
from slack_sdk.errors import SlackApiError
```

메시지 보내기

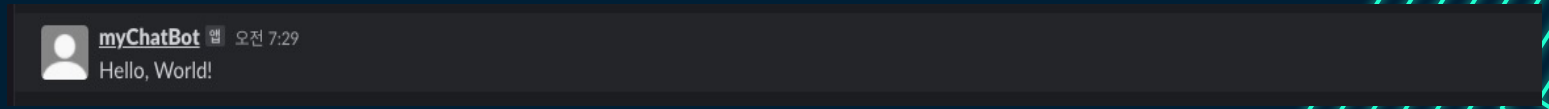
REFERENCE : 1_simple_pose.py

```python
# 1_simple_post.py
# 생성한 슬랙 봇을 통해 메시지를 보내는 예제

# 생성한 슬랙 handler 불러오기
from handler import slack_handler

# 메인 함수
if __name__ == "__main__":
    slack_handler.post_slack_message(message="Hello, World!", channel="#일반")
```

생성된 query


```
{'ok': True, 'channel': 'C026AN94Q2U', 'ts': '1624228999.001200', 'message': {'bot_id': 'B025T8AULNQ', 'type': 'message', 'text': 'Hello, World!', 'user': 'U025M2A1T6W', 'ts': '1624228999.001200', 'team': 'T025T7XTD8C', 'bot_profile': {'id': 'B025T8AULNQ', 'deleted': False, 'name': 'myChatBot', 'updated': 1624224709, 'app_id': 'A025T83H9E0', 'icons': {'image_36': 'https://a.slack-edge.com/80588/img/plugins/app/bot_36.png', 'image_48': 'https://a.slack-edge.com/80588/img/plugins/app/bot_48.png', 'image_72': 'https://a.slack-edge.com/80588/img/plugins/app/service_72.png'}, 'team_id': 'T025T7XTD8C'}}}
```


결과



플라스크 서버 설정

REFERENCE : 2_auto_response.py

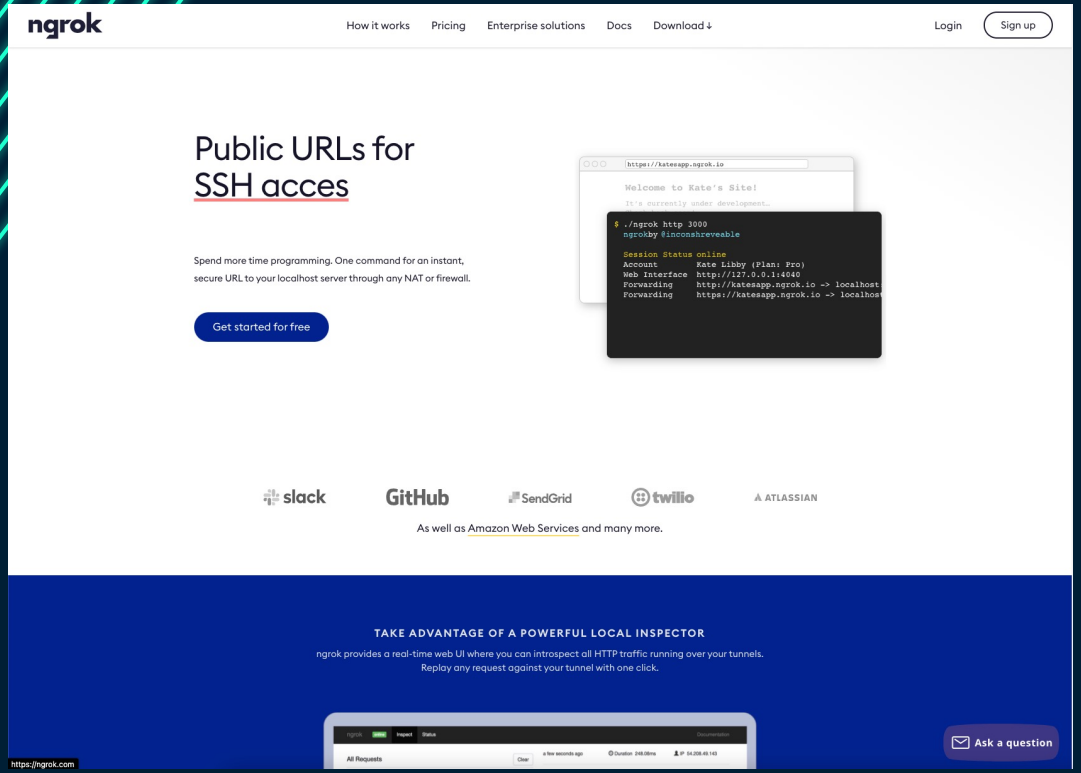
```python
import json

from flask import Flask, request, make_response

from handler import slack_handler

# 플라스크 인스턴스 생성
app = Flask(__name__)

```

플라스크 서버 설정

REFERENCE : 2_auto_response.py

```python
@app.route('/', methods=['POST'])
def default_listener():
    # 슬랙에서 보낸 request 데이터를 json으로 파싱한다.
    slack_event = json.loads(request.data)
    print(slack_event)

    # 인자 중 challenge가 있으면 해당 인자의 값을 반환한다.
    # slack api specification. 참고:https://api.slack.com/
    if "challenge" in slack_event:
        return make_response(slack_event["challenge"], 200, {"content_type": "application/json"})

    # slack에서 발생한 event를 통한 request에 대한 핸들링
    if "event" in slack_event:
        event_type = slack_event["event"]["type"]
        # bot_mention일 경우에 대한 핸들링
        if event_type == 'app_mention':
            try:
                # 멘션을 남긴 채널 읽어오기
                channel = slack_event['event']['channel']
                # 유저가 멘션과 함께 남긴 텍스트 읽어오기
                user_query = slack_event['event']['blocks'][0]['elements'][0]['elements'][1]['text']

                slack_handler.post_slack_message(message=user_query, channel=channel)

                # 정상적으로 완료했음에 대한 http response
                return make_response("response made :)", 200, )
            except IndexError:
                # 멘션은 했지만 텍스트는 남기지 않은 경우에 대한 에러.
                # do nothing
                pass
        # 그 외 event에 대한 핸들링: 404 error
        msg = f"[{event_type}] cannot find event handler"
        return make_response(msg, 404, {"X-Slack-No-Retry": 1 })
```

Ngrok 설정



1. https://ngrok.com/
2. 다운로드
3. Ngrok 실행 파일을 코드와 같은 폴더로 이동

Ngrok 실행

1. 작성한 플라스크 서버를 로컬서버에서 실행

```
➜  codes python3 autoResponse.py
 * Serving Flask app 'autoResponse' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deplo
yment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 845-649-508
```

2. Ngrok을 통해 로컬과 외부 인터넷 연결

```
➜  codes ./ngrok http 5000
```

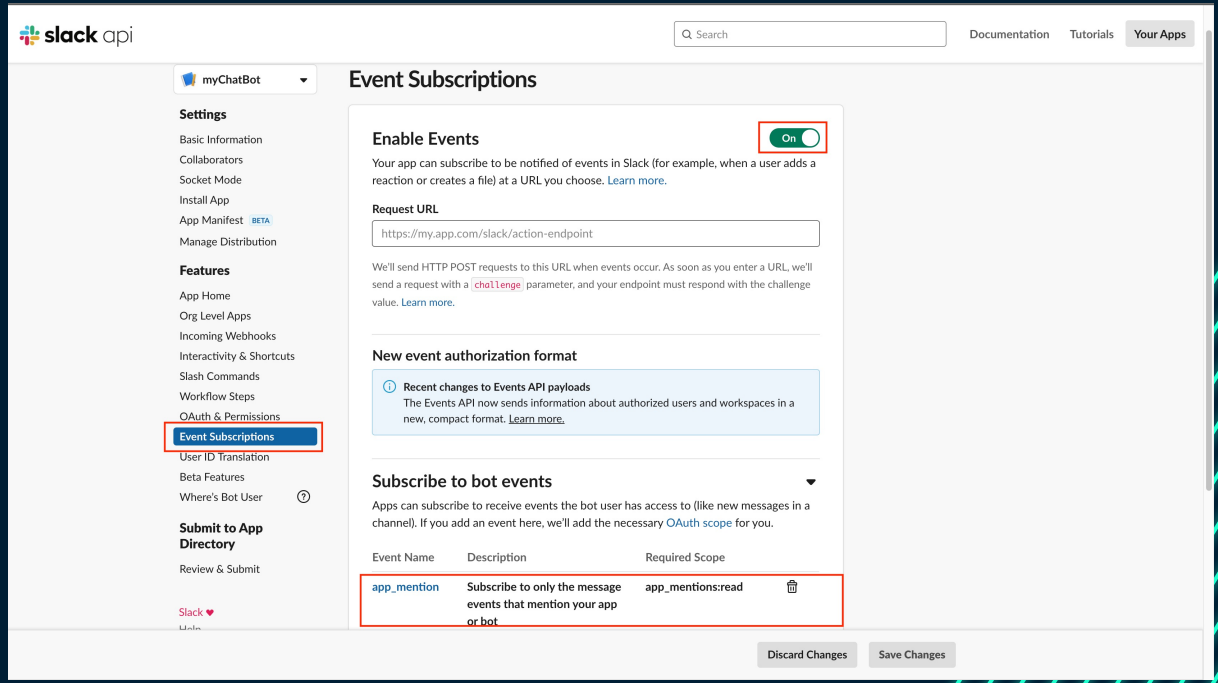
3. Forwarding URL 복사

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Account                       KORguy (Plan: Free)
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://ffd948d685a0.ngrok.io -> http://localhos
Forwarding                    https://ffd948d685a0.ngrok.io -> http://localho

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

SLACK API 추가 설정

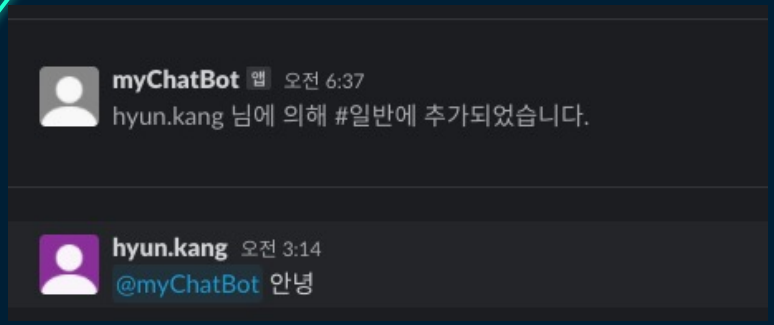
슬랙에서 봇을 멘션하면 플라스크 서버로 request 보내는 환경 설정

1. Your apps > event subscriptions
2. Enable events
3. Subscribe to bot events > app_mention
4. Request url : <forwarding_url_from_ngrok>



URL : https://api.slack.com/apps

자동응답 테스트




```
{
  'token': 'fd54wao3UbSiugxNiVAWXq0W',
  'team_id': 'T025T7XTD8C',
  'api_app_id': 'A025T83H9E0',
  'event': {
    'client_msg_id': '9b9bbabe-c2df-4552-894a-2a35baf93730',
    'type': 'app_mention',
    'text': '<@U025M2A1T6W> 안녕',
    'user': 'U025E9D5KSS',
    'ts': '1624299241.000300',
    'team': 'T025T7XTD8C',
    'blocks': [
      {
        'type': 'rich_text',
        'block_id': 'D2m',
        'elements': [
          {
            'type': 'rich_text_section',
            'elements': [
              {
                'type': 'user',
                'user_id': 'U025M2A1T6W'
              },
              {
                'type': 'text',
                'text': ' 안녕'
              }
            ]
          }
        ]
      }
    ],
    'channel': 'C026AN94Q2U',
    'event_ts': '1624299241.000300'
  },
  'type': 'event_callback',
  'event_id': 'Ev02632VGUGZ',
  'event_time': 1624299241,
  'authorizations': [
    {
      'enterprise_id': None,
      'team_id': 'T025T7XTD8C',
      'user_id': 'U025M2A1T6W',
      'is_bot': True,
      'is_enterprise_install': False
    }
  ],
  'is_ext_shared_channel': False,
  'event_context': '2-app_mention-T025T7XTD8C-A025T83H9E0-C026AN94Q2U'
}
```

Handler 추가

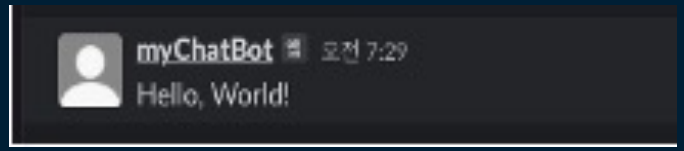
REFERENCE : updated_handler.py

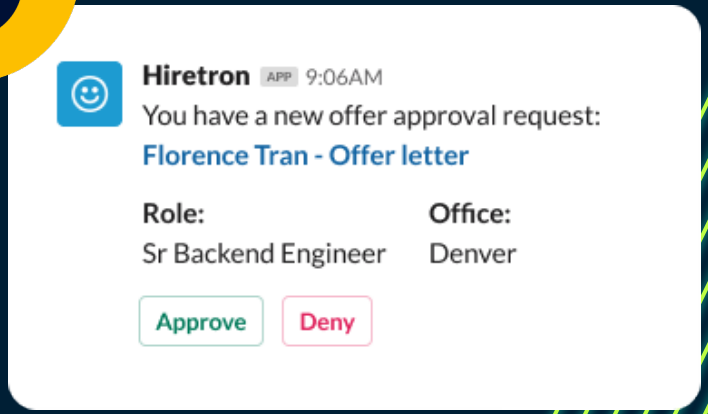
```python
# 레이아웃을 포함한 메시지를 보내는 메소드
def post_layout_message(self, channel: str, blocks: list):
    try:
        response = self.client.chat_postMessage(channel=channel, blocks=blocks)
        print(response)
    except SlackApiError as e:
        assert e.response["ok"] is False
        assert e.response["error"]
        print(f"ERROR: {e.response['error']}")
```

BLOCKS

Slack Block이란.

단순한 텍스트를 넘어 이미지나 링크, 투표 등을 포함한 레이아웃에 맞춰진 응답 또한 가능.





Block Types

Block type	Available in surfaces
Actions	Modals Messages Home tabs
Context	Modals Messages Home tabs
Divider	Modals Messages Home tabs
File	Messages
Header	Modals Messages Home tabs
Image	Modals Messages Home tabs
Input	Modals Messages Home tabs
Section	Modals Messages Home tabs

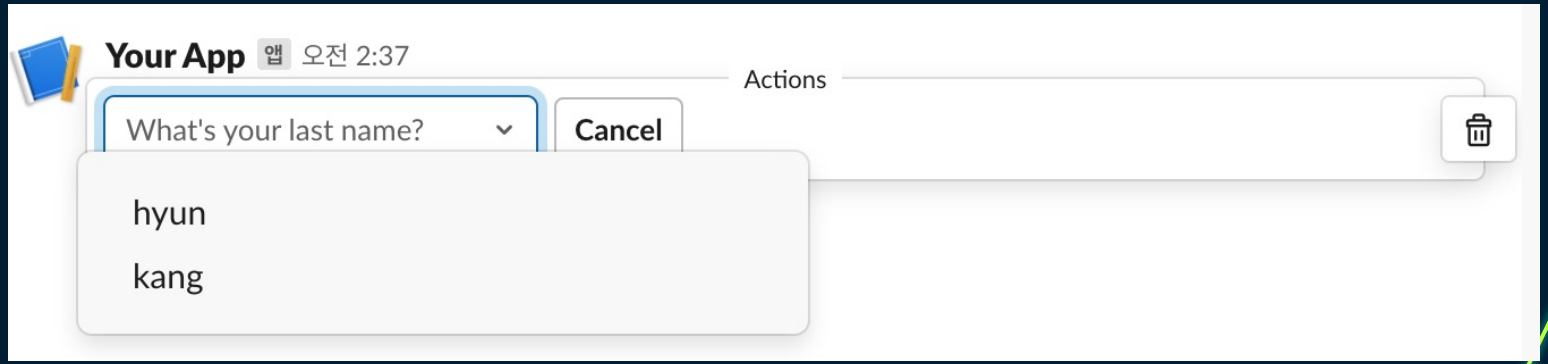
1. Action Type

사용자와 상호작용을 할 수 있는 버튼 등을 포함한 블록

Field	Type	Required?	Description
`type`	String	Yes	The type of block. For an actions block, `type` is always `actions`.
`elements`	Object[]	Yes	An array of interactive element objects - buttons, select menus, overflow menus, or date pickers. There is a maximum of 5 elements in each action block.
`block_id`	String	No	A string acting as a unique identifier for a block. If not specified, a `block_id` will be generated. You can use this `block_id` when you receive an interaction payload to identify the source of the action. Maximum length for this field is 255 characters. `block_id` should be unique for each message and each iteration of a message. If a message is updated, use a new `block_id`.

1. Action Type

사용자와 상호작용을 할 수 있는 버튼 등을 포함한 블록



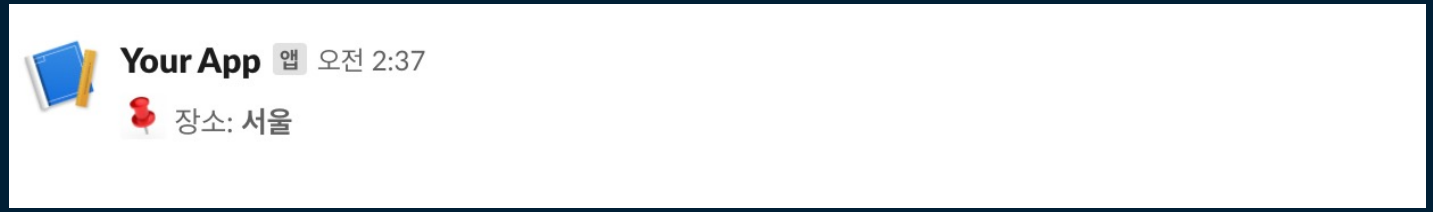
2. Context Type

텍스트와 이미지를 포함한 블록

Field	Type	Required?	Description
`type`	String	Yes	The type of block. For a context block, `type` is always `context`.
`elements`	Object[]	Yes	An array of image elements and text objects. Maximum number of items is 10.
`block_id`	String	No	A string acting as a unique identifier for a block. If not specified, one will be generated. Maximum length for this field is 255 characters. `block_id` should be unique for each message and each iteration of a message. If a message is updated, use a new `block_id`.

2. Context Type

텍스트와 이미지를 포함한 블록



3. Divider Type

블록들을 나누는 평행선 블록

Field	Type	Required?	Description
`type`	String	Yes	The type of block. For a divider block, `type` is always `divider`.
`block_id`	String	No	A string acting as a unique identifier for a block. If not specified, one will be generated. Maximum length for this field is 255 characters. `block_id` should be unique for each message and each iteration of a message. If a message is updated, use a new `block_id`.

3. Divider Type

블록들을 나누는 평행선 블록

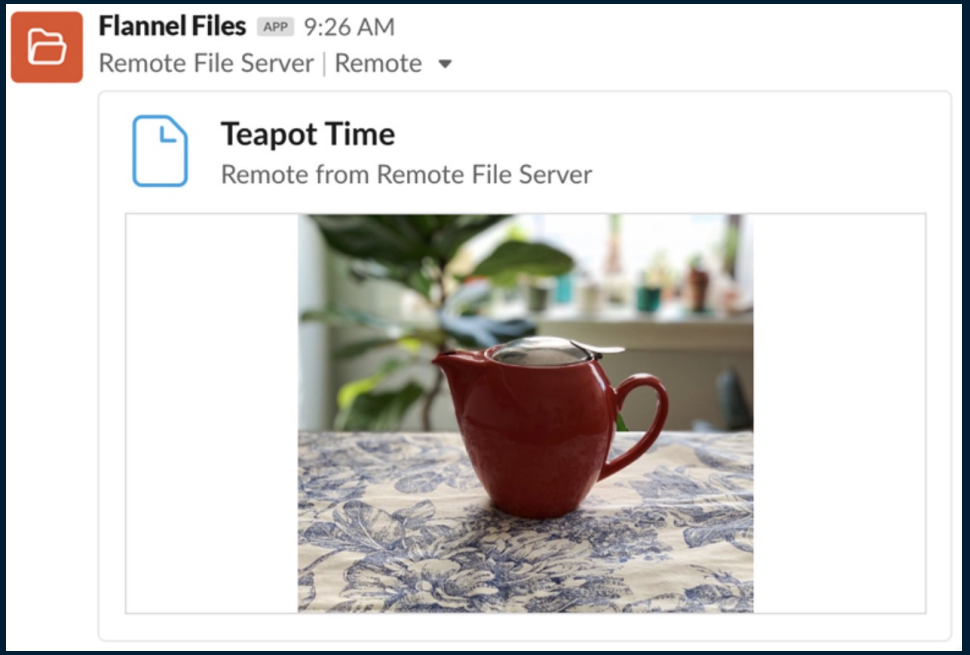
4. File Type

Remote File을 포함한 블록

Field	Type	Required?	Description
`type`	String	Yes	The type of block. For a file block, `type` is always `file`.
`external_id`	String	Yes	The external unique ID for this file.
`source`	String	Yes	At the moment, `source` will always be `remote` for a remote file.
`block_id`	String	No	A string acting as a unique identifier for a block. If not specified, one will be generated. Maximum length for this field is 255 characters. `block_id` should be unique for each message and each iteration of a message. If a message is updated, use a new `block_id`.

4. File Type

Remote File을 포함한 블록



5. Header Type

일반 텍스트보다 크고, 볼드체의 텍스트를 포함한 블록

Field	Type	Required?	Description
`type`	String	Yes	The type of block. For this block, type will always be `header`.
`text`	Object	Yes	The text for the block, in the form of a `plain_text` text object. Maximum length for the `text` in this field is 150 characters.
`block_id`	String	No	A string acting as a unique identifier for a block. If not specified, one will be generated. Maximum length for this field is 255 characters. `block_id` should be unique for each message and each iteration of a message. If a message is updated, use a new `block_id`.

5. Header Type

일반 텍스트보다 크고, 볼드체의 텍스트를 포함한 블록

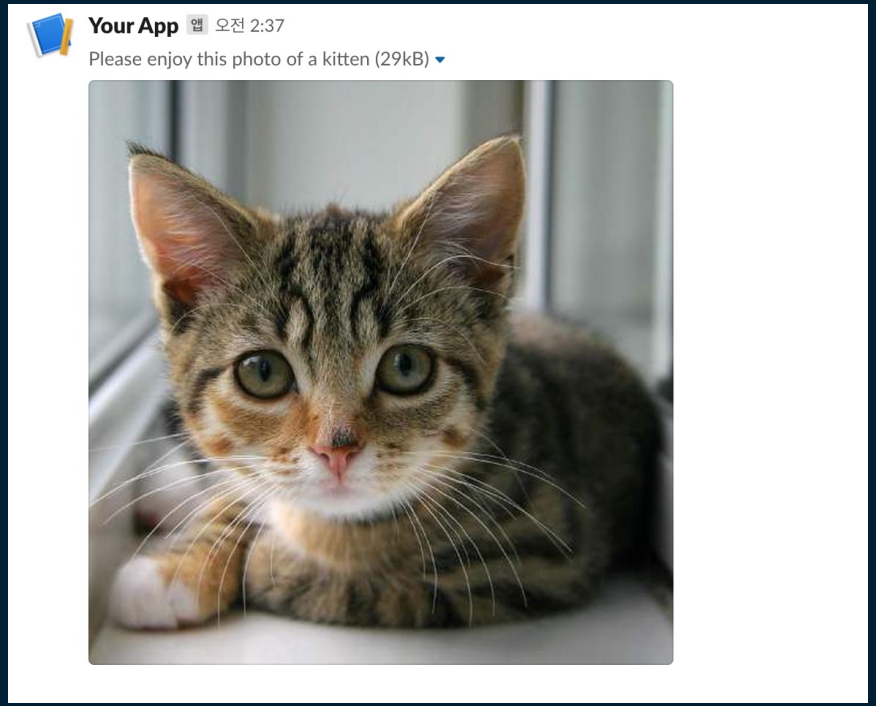
6. Image Type

이미지를 포함한 블록

Field	Type	Required?	Description
`type`	String	Yes	The type of block. For an image block, `type` is always `image`.
`image_url`	String	Yes	The URL of the image to be displayed. Maximum length for this field is 3000 characters.
`alt_text`	String	Yes	A plain-text summary of the image. This should not contain any markup. Maximum length for this field is 2000 characters.
`title`	Object	No	An optional title for the image in the form of a text object that can only be of `type: plain_text`. Maximum length for the `text` in this field is 2000 characters.
`block_id`	String	No	A string acting as a unique identifier for a block. If not specified, one will be generated. Maximum length for this field is 255 characters. `block_id` should be unique for each message and each iteration of a message. If a message is updated, use a new `block_id`.

6. Image Type

이미지를 포함한 블록



7. Input Type

사용자의 인풋을 받을 수 있는 블록

Field	Type	Required?	Description
`type`	String	Yes	The type of block. For an input block, `type` is always `input`.
`label`	Object	Yes	A label that appears above an input element in the form of a text object that must have `type` of `plain_text`. Maximum length for the `text` in this field is 2000 characters.
`element`	Object	Yes	An plain-text input element, a checkbox element, a radio button element, a select menu element, a multi-select menu element, or a datepicker.
`dispatch_action`	Boolean	No	A boolean that indicates whether or not the use of elements in this block should dispatch a `block_actions` payload. Defaults to `false`.
`block_id`	String	No	A string acting as a unique identifier for a block. If not specified, one will be generated. Maximum length for this field is 255 characters. `block_id` should be unique for each message or view and each iteration of a message or view. If a message or view is updated, use a new `block_id`.
`hint`	Object	No	An optional hint that appears below an input element in a lighter grey. It must be a a text object with a `type` of `plain_text`. Maximum length for the `text` in this field is 2000 characters.
`optional`	Boolean	No	A boolean that indicates whether the input element may be empty when a user submits the modal. Defaults to `false`.

7. Input Type

사용자의 인풋을 받을 수 있는 블록

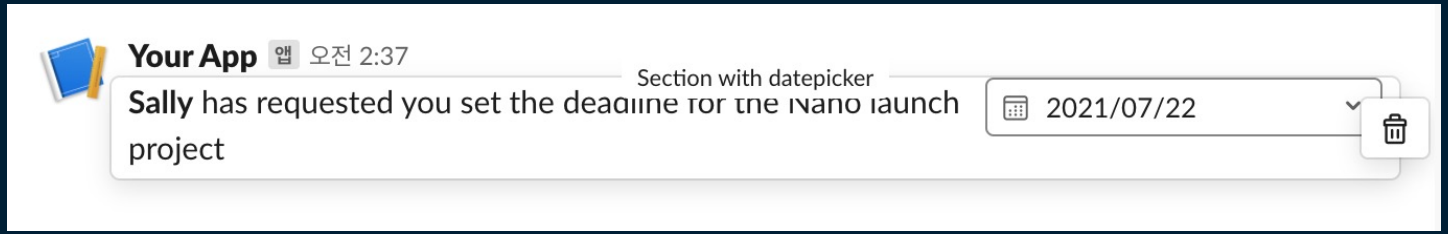
8. Section Type

가장 유연한 블록으로 이미지나 텍스트를 자유자재로 배치 가능한 블록

Field	Type	Required?	Description
`type`	String	Yes	The type of block. For a section block, type will always be `section`.
`text`	Object	Preferred	The text for the block, in the form of a text object. Maximum length for the `text` in this field is 3000 characters. This field is not *required* if a valid array of `fields` objects is provided instead.
`block_id`	String	No	A string acting as a unique identifier for a block. If not specified, one will be generated. You can use this `block_id` when you receive an interaction payload to identify the source of the action. Maximum length for this field is 255 characters. `block_id` should be unique for each message and each iteration of a message. If a message is updated, use a new `block_id`.
`fields`	Object[]	*Maybe*	Required if no `text` is provided. An array of text objects. Any text objects included with `fields` will be rendered in a compact format that allows for 2 columns of side-by-side text. Maximum number of items is 10. Maximum length for the `text` in each item is 2000 characters. Click here for an example.
`accessory`	Object	No	One of the available element objects.

8. Section Type

가장 유연한 블록으로 이미지나 텍스트를 자유자재로 배치 가능한 블록



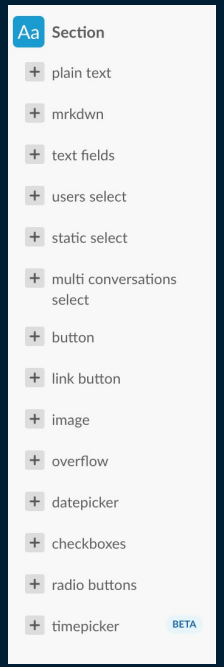
* Rich Text

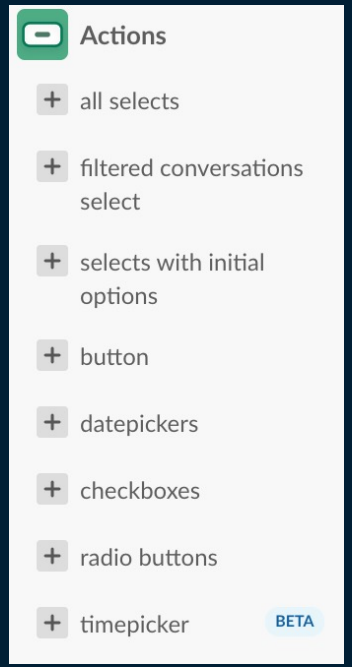
Text Type 을 마크다운으로 좀 더 자유로운 텍스트 또한 가능

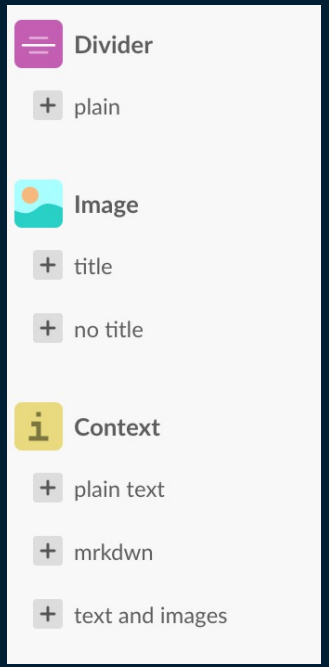
```
{
  "type": "section",
  "text": {
    "type": "mrkdwn",
    "text": "New Paid Time Off request from <example.com|Fred Enriquez>\n\n<https://example.com|View request>"
  }
}
```

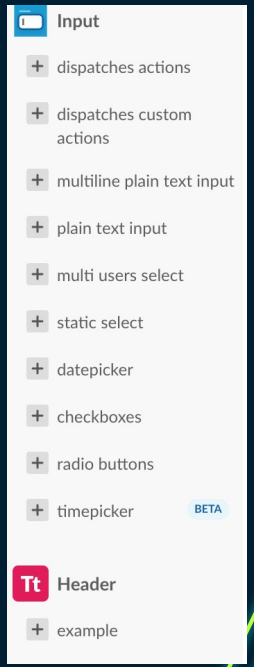
GUI Block Building Kit

단순 클릭으로 block 자동 포맷 생성









DEMO:
MOVIE BOT

영화 정보 챗봇

개요:

챗봇을 통해 현재 상영중인 영화에 대한 정보를 불러오고 이를 슬랙 인터페이스를 통해 유저에게 전달해주는 업무를 진행한다.

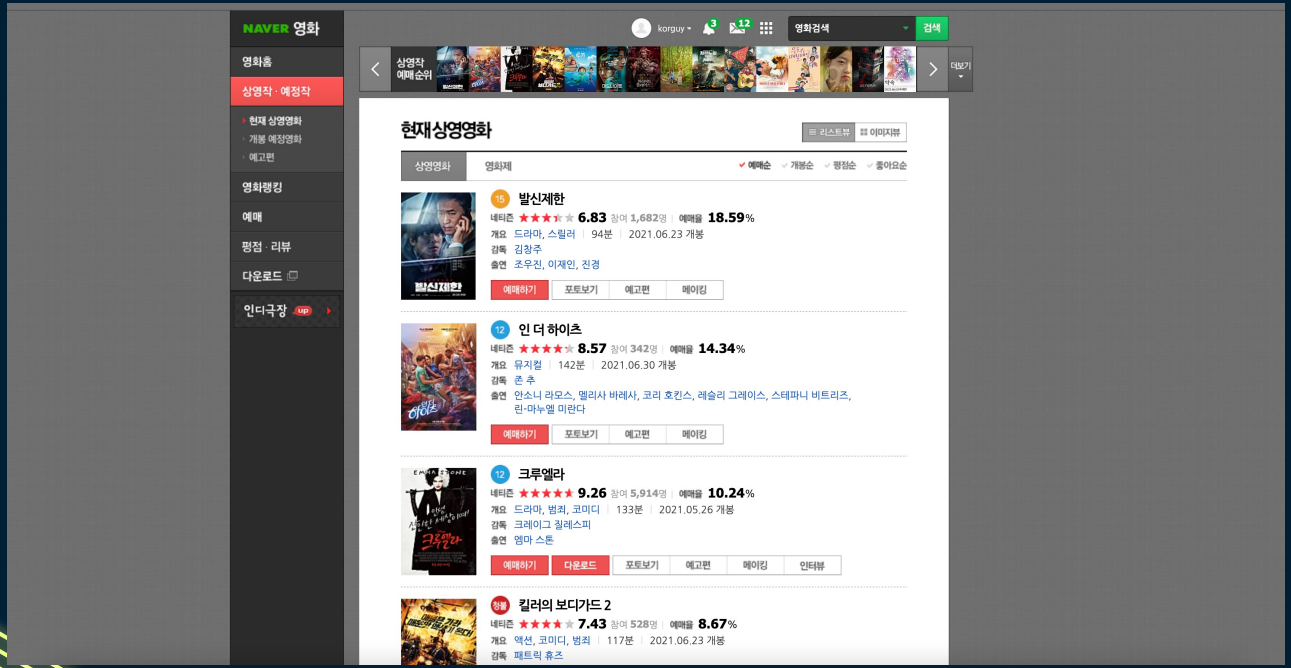
단계:

1. 네이버 영화로부터 현재 상영작에 대한 정보 불러오기
2. 슬랙 블록으로 변환하여 유저에게 보여주기
3. 사용자의 위치 정보 받기
4. 위치 정보를 기반으로 예약 페이지에 대한 링크 생성

크롤링

영화 상영정보 확인하기

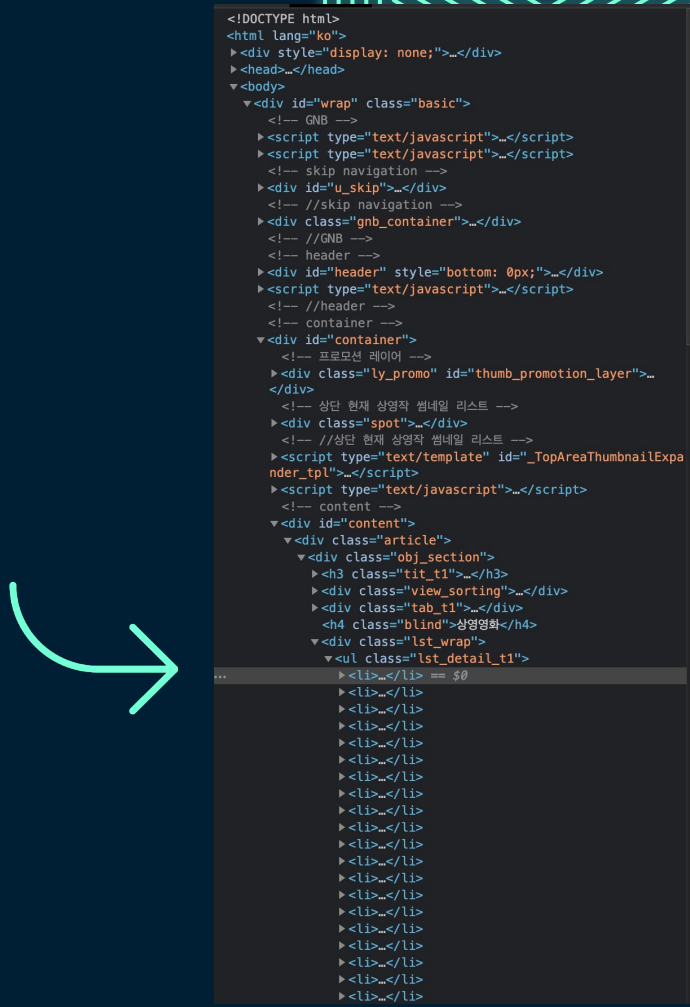
https://movie.naver.com/movie/running/current.nhn



크롤링

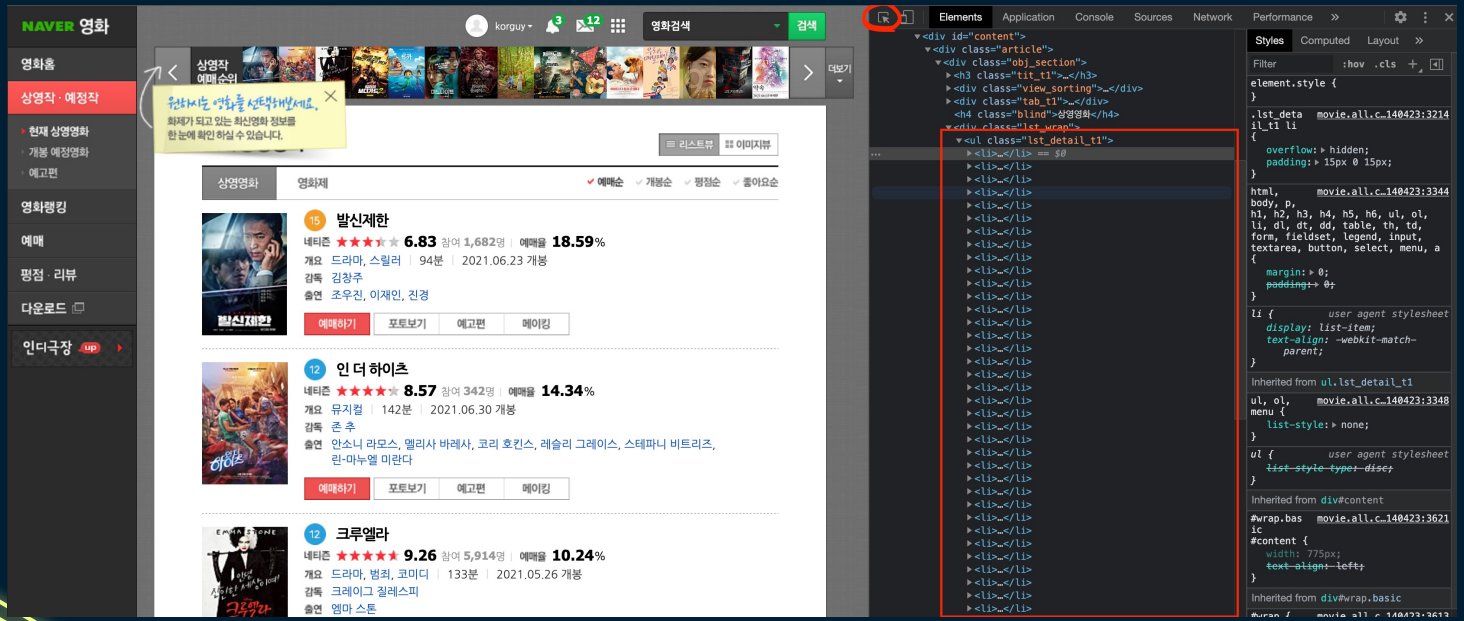
REFERENCE : 1_crawling.py

```python
# 1_crawling.py
# 네이버 실시간 영화 정보를 불러오는 코드를 작성한다.

# pip install beautifulsoup4
import requests
from bs4 import BeautifulSoup

URL = "https://movie.naver.com/movie/running/current.nhn#"

# 영화 정보를 불러와 포맷에 맞춰 return 해주는 함수
def get_current_movies():
    # 1. URL로부터 html을 파싱해오기
    req = requests.get(URL)
    html = req.text
    soup = BeautifulSoup(html, 'html.parser')

# 메인함수
if __name__ == "__main__":
    get_current_movies()
```



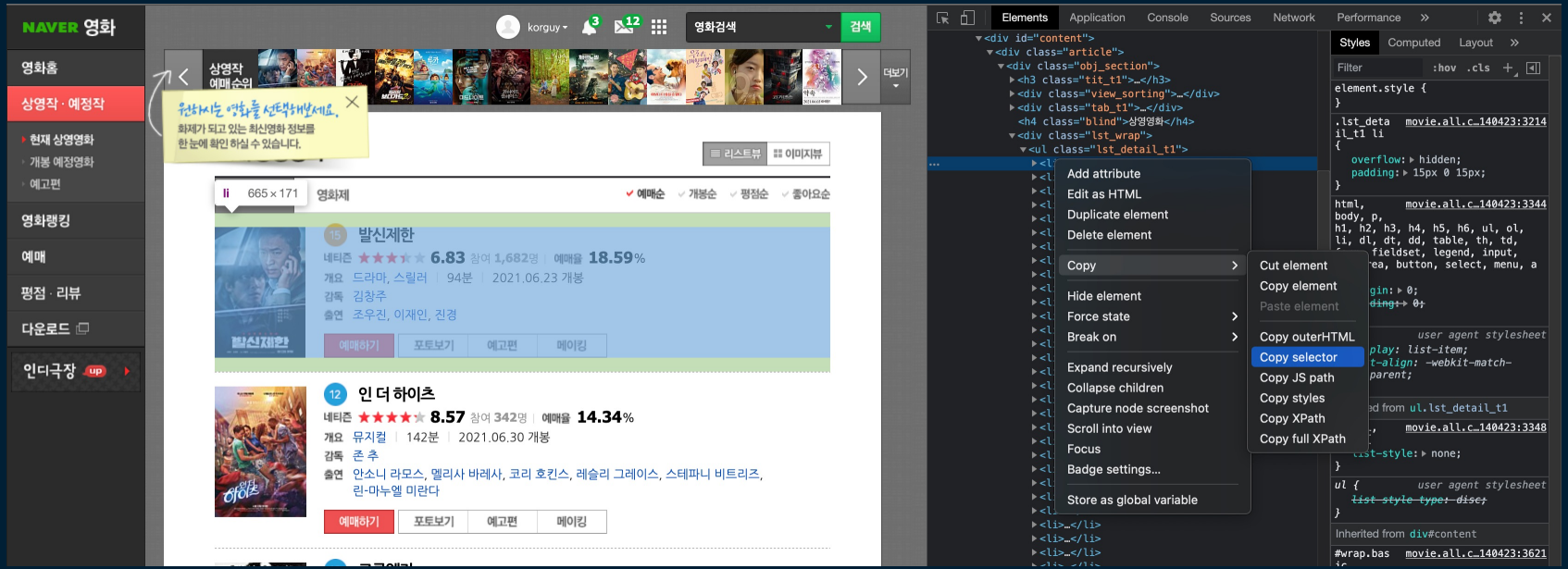
크롤링

1. F12 혹은 우클릭 > 페이지 소스 검사
2. 영화들 정보가 있는 부분 찾기



=> lst_detail_t1 이라는 ul 안에 li 형태로 영화들 정보가 담겨져 있음

크롤링



=> #content > div.article > div:nth-child(1) > div.lst_wrap > ul > li:nth-child(1)

크롤링

REFERENCE : 1_crawling.py

```
10  # 영화 정보를 불러와 포맷에 맞춰 return 해주는 함수
11  def get_current_movies():
12      # 1. URL로부터 html을 파싱해오기
13      req = requests.get(URL)
14      html = req.text
15      soup = BeautifulSoup(html, 'html.parser')
16      # 2. 영화 부분 추출하기
17      lis = soup.select('#content > div.article > div:nth-of-type(1) > div.lst_wrap > ul > li')
```

=> #content > div.article > div:nth-child(1) > div.lst_wrap > ul > li:nth-child(1)
**** nth-child를 nth-of-type으로 바꿔줘야 함**
**** 모든 li를 불러오기 위해 마지막 nth-child는 삭제**

크롤링

REFERENCE : 1_crawling.py

```
# 3. 필요한 정보만 추출
movies = []
for idx, li in enumerate(lis):
    temp = {}
    temp['title'] = li.select('dl > dt > a')[0].text
    temp['score'] = float(li.select('dl > dd.star > dl.info_star > dd > div > a > span.num')[0].text)
    detail = li.select('.link_txt') # 개요, 감독, 출연
    temp['genre'] = [x.text for x in detail[0].select('a')]
    temp['director'] = [x.text for x in detail[1].select('a')]
    # 애니메이션처럼 배우 정보가 없는 경우가 있음
    if len(detail) > 2:
        temp['actors'] = [x.text for x in detail[2].select('a')]
    movies.append(temp)
```

크롤링

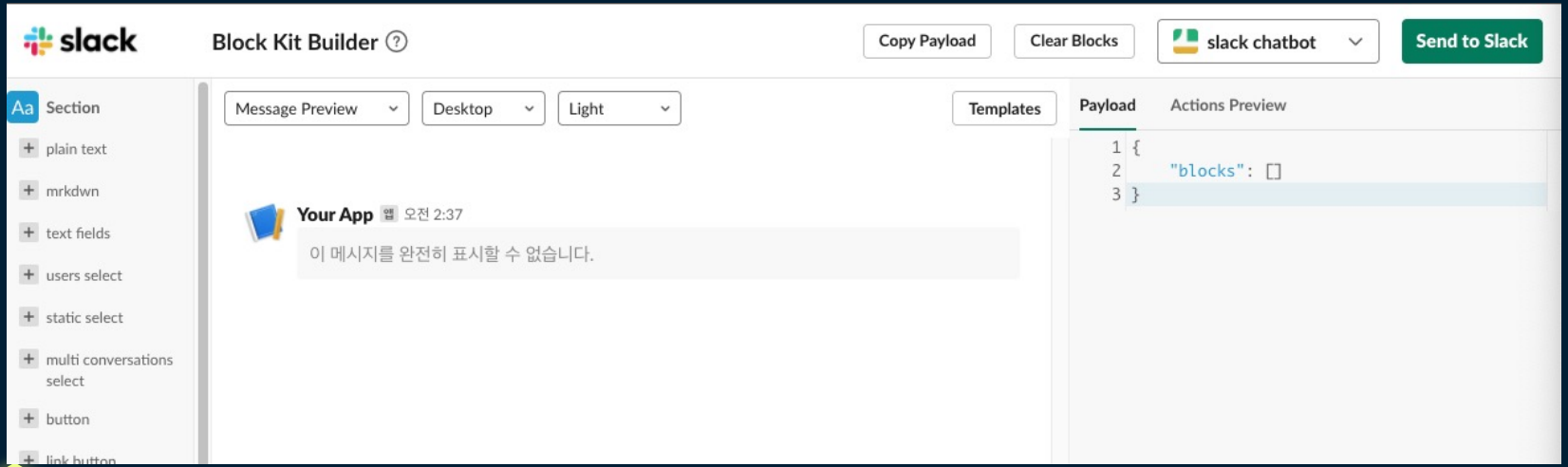
REFERENCE : 1_crawling.py

```python
# 1_crawling.py
# 네이버 실시간 영화 정보를 불러오는 코드를 작성한다.

# pip install beautifulsoup4
import requests
from bs4 import BeautifulSoup

URL = "https://movie.naver.com/movie/running/current.nhn#"

# 영화 정보를 불러와 딕셔너리 형태에 맞춰 리스트로 return 해주는 함수
def get_current_movies():
    # 1. URL로부터 html을 파싱해오기
    req = requests.get(URL)
    html = req.text
    soup = BeautifulSoup(html, 'html.parser')
    # 2. 영화 부분 추출하기
    lis = soup.select('#content > div.article > div:nth-of-type(1) > div.lst_wrap > ul > li')
    # 3. 필요한 정보만 추출
    movies = []
    for idx, li in enumerate(lis):
        temp = {}
        temp['title'] = li.select('dl > dt > a')[0].text
        temp['score'] = float(li.select('dl > dd.star > dl.info_star > dd > div > a > span.num')[0].text)
        detail = li.select('.link_txt') # 개요, 감독, 출연
        temp['genre'] = [x.text for x in detail[0].select('a')]
        temp['director'] = [x.text for x in detail[1].select('a')]
        # 애니메이션처럼 배우 정보가 없는 경우가 있음
        if len(detail) > 2:
            temp['actors'] = [x.text for x in detail[2].select('a')]
        movies.append(temp)
    return movies

# 메인함수
if __name__ == "__main__":
    movies = get_current_movies()
    print(movies[0])
```

```
{
  'title': '보이스',
  'score': 8.55,
  'genre': [
    '범죄',
    '액션'
  ],
  'director': [
    '김선',
    '김곡'
  ],
  'actors': [
    '변요한',
    '김무열',
    '김희원',
    '박명훈'
  ]
}
```

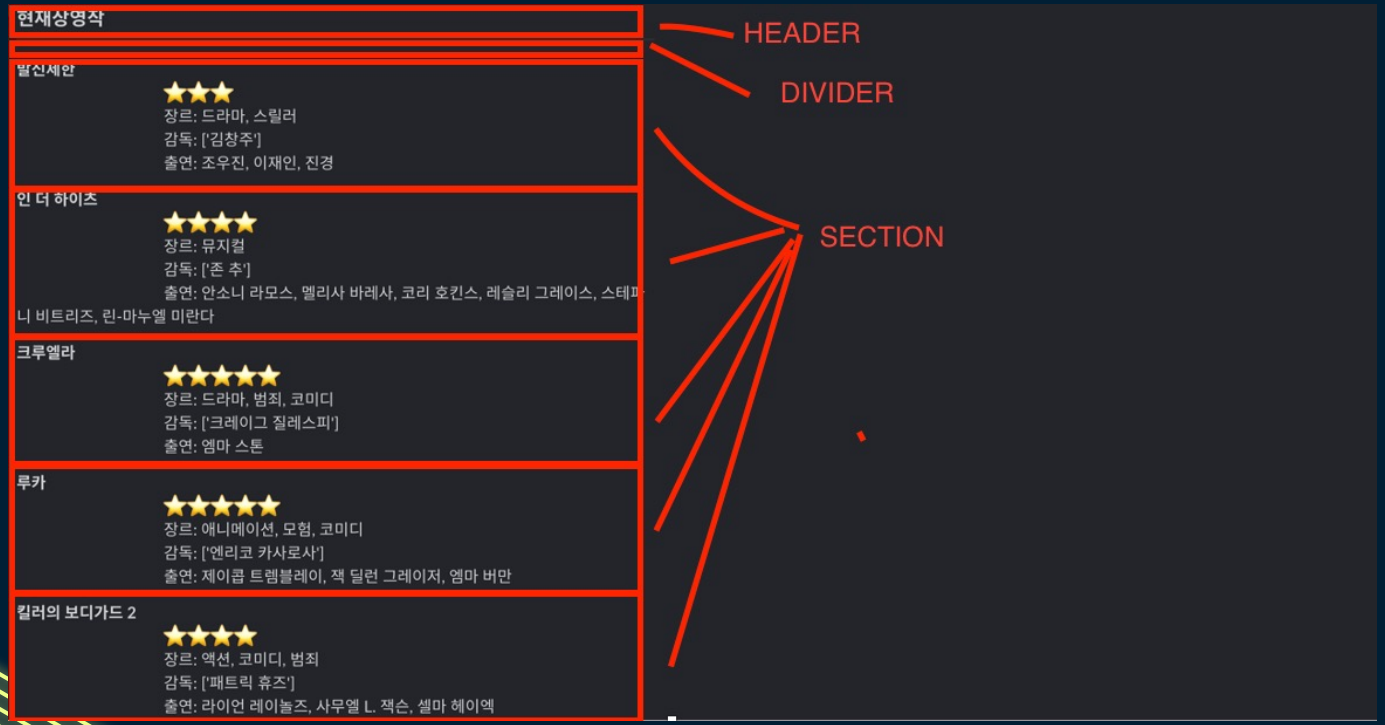
블록 구상

1. https://app.slack.com/block-kit-builder/
2. 빈 블록 생성



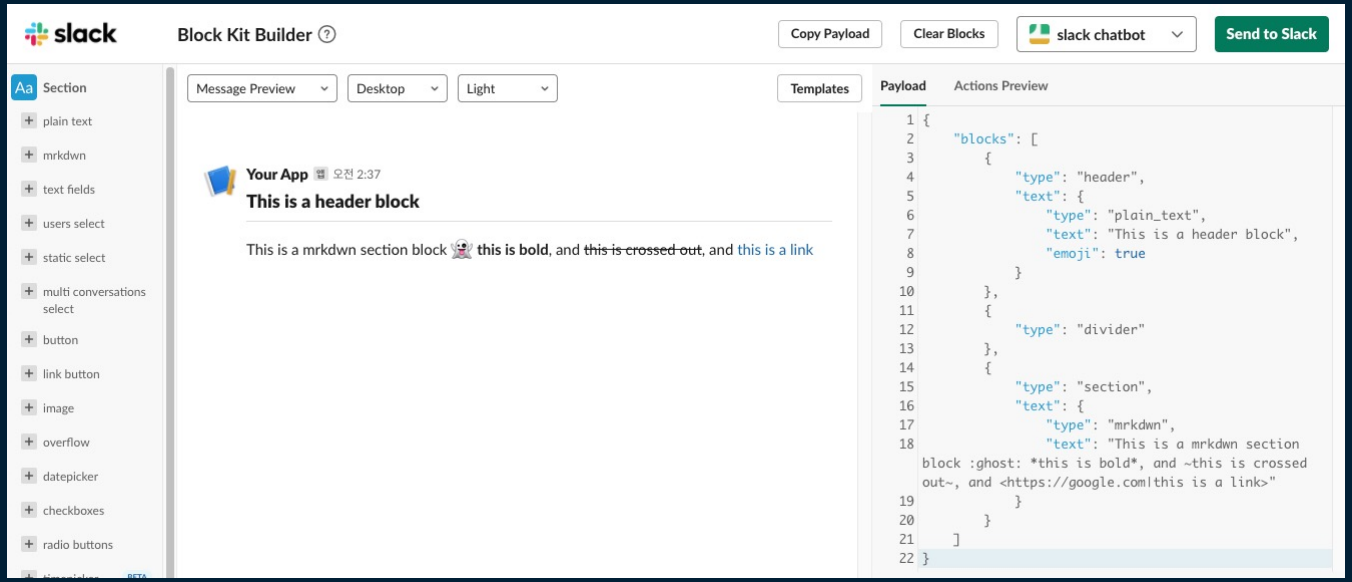
블록 구상

블록 레이아웃 구상



블록 생성

Block Kit Builder로 블록 포맷 확인



=> 좌측에서 header, divider, section (mrkdwn) 클릭한 결과

블록 생성

우측 블록 리스트 확인

```
{
    "blocks": [
        {
            "type": "header",
            "text": {
                "type": "plain_text",
                "text": "This is a header block",
                "emoji": true
            }
        },
        {
            "type": "divider"
        },
        {
            "type": "section",
            "text": {
                "type": "mrkdwn",
                "text": "This is a mrkdwn section block :ghost: *this is bold*, and ~this is crossed out~, and <https://google.com|this is a link>"
            }
        }
    ]
}
```

블록 생성

```
{
    "blocks": [
        {
            "type": "header",
            "text": {
                "type": "plain_text",
                "text": "This is a header block",
                "emoji": true
            }
        },
        {
            "type": "divider"
        },
        {
            "type": "section",
            "text": {
                "type": "mrkdwn",
                "text": "This is a mrkdwn section block :ghost: *this is bold*, and ~this is crossed out~, and <https://google.com|this is a link>"
            }
        }
    ]
}
```

Block 1:
- Type: header
- Text:
  - Type: plain_text
  - Text: string
  - Emoji: (optional)

Block 2:
- Type: divider

Block 3:
- Type: section
- Text:
  - Type: mrkdwn
  - Text: string

블록 생성

블록 담을 리스트 생성

```
# 블록 담을 리스트
block = []
```

블록 생성

헤더 블록 쌓기

1. 빈 dictionary 생성
2. Key 별로 value 지정 ex. key: type => value: header
3. Dictionary 내 dictionary가 있는 경우 마찬가지로 dictionary 생성 후 키 지정
4. 블록 리스트에 추가

Block 1:

- Type: header
- Text:
  - Type: plain_text
  - Text: string
  - Emoji: (optional)

```
# 헤더 블록
header = {}
header['type']  = 'header'
header['text'] = {}
header['text']['type'] = 'plain_text'
header['text']['text'] = '현재상영작'
block.append(header)
```

블록 생성

Divider 블록 쌓기

1. 빈 dictionary 생성
2. Key 별로 value 지정
3. 블록 리스트에 추가

Block 2:
- Type: divider

```
# divider 블록
divider = {}
divider['type'] = 'divider'
block.append(divider)
```

블록 생성

Section 블록 쌓기

1. 영화 object별로 빈 dictionary 생성
2. Key-value pair 지정
3. Markdown 포맷에 맞게 영화 object 값들 지정
4. 블록 리스트에 추가

Block 3:

- Type: section
- Text:
  - Type: mrkdwn
  - Text: string

블록 생성

Section 블록 쌓기

영화 Object

```
{
  'title': '보이스',
  'score': 8.55,
  'genre': [
    '범죄',
    '액션'
  ],
  'director': [
    '김선',
    '김곡'
  ],
  'actors': [
    '변요한',
    '김무열',
    '김희원',
    '박명훈'
  ]
}
```

좌: markdown string
우: 아웃풋

**StackEdit에서 Markdown 확인 가능*

블록 생성

Section 블록 쌓기

```
**{movie['title']}**
{":star:" * (movie['score']/2)}
장르 : {', '.join(movie['genre'])}
감독 : {', '.join(movie['director'])}
출연 : {', '.join(movie['actors'])}
```

```
{movie['title']}
{"⭐" * (movie['score']/2)}
장르 : {', '.join(movie['genre'])}
감독 : {', '.join(movie['director'])}
출연 : {', '.join(movie['actors'])}
```

변수로 치환

블록 생성

Section 블록 쌓기

```
# 영화 블록
for movie in movies:
    temp = {}
    temp['type'] = 'section'
    temp['text'] = {}
    temp['text']['type'] = 'mrkdwn'
    stars = ":star:" * round(movie['score']/2)
    temp['text']['text'] = f"*{movie['title']}*\n \
                            {stars}\n \
                            장르: {', '.join(movie['genre'])}\n \
                            감독: {movie['director']}\n \
                            출연: {', '.join(movie['actors'])}\n"
    block.append(temp)
```

블록 생성

REFERENCE : 2_display_movies.py

```python
def build_blocks(movies: list):
    # 블록 담을 리스트
    block = []
    # 헤더 블록
    header = {}
    header['type']  = 'header'
    header['text'] = {}
    header['text']['type'] = 'plain_text'
    header['text']['text'] = '현재상영작'
    block.append(header)
    # divider 블록
    divider = {}
    divider['type'] = 'divider'
    block.append(divider)
    # 영화 블록
    for movie in movies:
        temp = {}
        temp['type'] = 'section'
        temp['text'] = {}
        temp['text']['type'] = 'mrkdwn'
        stars = ":star:" * round(movie['score']/2)
        temp['text']['text'] = f"*{movie['title']}*\n \
                                {stars}\n \
                                장르: {', '.join(movie['genre'])}\n \
                                감독: {movie['director']}\n \
                                출연: {', '.join(movie['actors'])}\n"
        block.append(temp)
    return block
```

서버 수정

REFERENCE : 2_display_movies.py

```python
# 2_display_movies.py
# 크롤링한 영화 정보를 포맷에 맞춰 슬랙에 보내주는 역할

import json

# pip install Flask
from flask import Flask, request, make_response
from handler import slack_handler

crawling = __import__('1_crawling')
```

서버 수정

REFERENCE : 2_display_movies.py

```
25      # slack에서 발생한 event를 통한 request에 대한 핸들링
26      if "event" in slack_event:
27          event_type = slack_event["event"]["type"]
28          # bot_mention일 경우에 대한 핸들링
29          if event_type == 'app_mention':
30              try:
31                  # 멘션을 남긴 채널 읽어오기
32                  channel = slack_event['event']['channel']
33                  # 유저가 멘션과 함께 남긴 텍스트 읽어오기
34                  user_query = slack_event['event']['blocks'][0]['elements'][0]['elements'][1]['text']
35
36                  if user_query.strip() == '/영화정보':
37                      movies = crawling.get_current_movies()
38                      block = build_blocks(movies)
39                      slack_handler.post_layout_message(blocks=block, channel=channel)
40
41                  # 정상적으로 완료했음에 대한 http response
42                  return make_response("response made :)", 200, )
43              except IndexError:
44                  # 멘션은 했지만 텍스트는 남기지 않은 경우에 대한 에러.
45                  # do nothing
46                  pass
```

서버 수정

REFERENCE : 2_display_movies.py

```python
def build_blocks(movies: list):
    # 블록 담을 리스트
    block = []
    # 헤더 블록
    header = {}
    header['type']  = 'header'
    header['text'] = {}
    header['text']['type'] = 'plain_text'
    header['text']['text'] = '현재상영작'
    block.append(header)
    # divider 블록
    divider = {}
    divider['type'] = 'divider'
    block.append(divider)
    # 영화 블록
    for movie in movies:
        temp = {}
        temp['type'] = 'section'
        temp['text'] = {}
        temp['text']['type'] = 'mrkdwn'
        stars = ":star:" * round(movie['score']/2)
        temp['text']['text'] = f"*{movie['title']}*\n \
                                {stars}\n \
                                장르: {', '.join(movie['genre'])}\n \
                                감독: {movie['director']}\n \
                                출연: {', '.join(movie['actors'])}\n"
        block.append(temp)
    return block
```

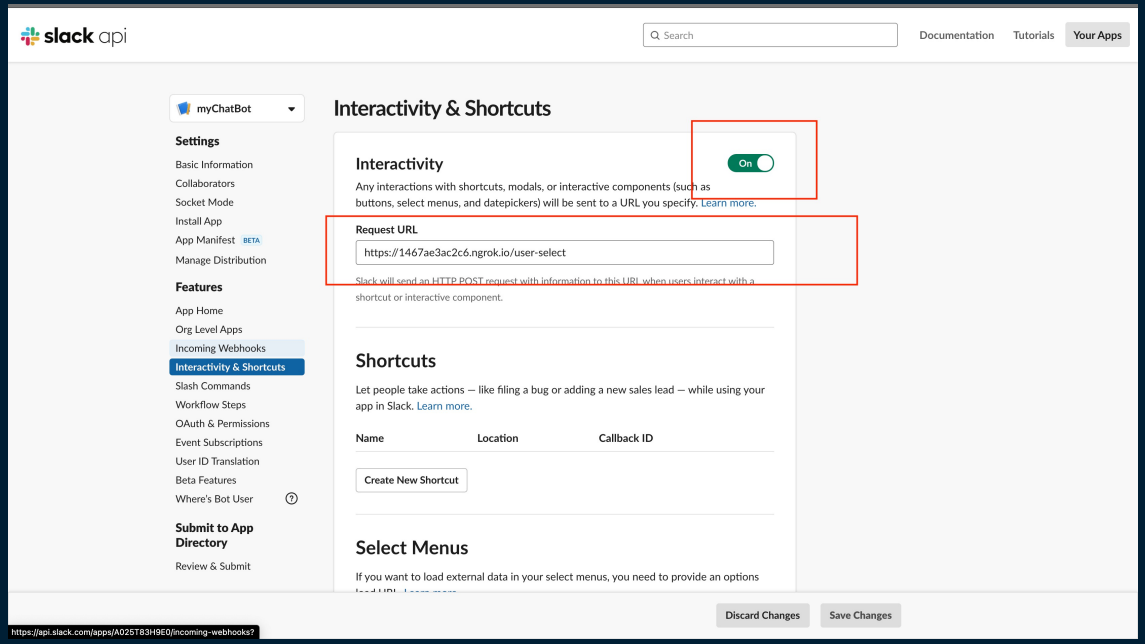
서버 수정

결과:


@myChatBot /영화정보
오늘
myChatBot 앱 오전 4:38
현재상영작
보이스
8.55
장르: 범죄, 액션
감독: 김선, 김곡
출연: 변요한, 김무열, 김희원, 박명훈
극장판 포켓몬스터: 정글의 아이, 코코
8.94
장르: 애니메이션, 모험
감독: 야지마 테츠오
출연: 마츠모토 리카, 오오타니 이쿠에, 하야시바라 메구미, 미키 신이치로
샹치와 텐 링즈의 전설
6.6
장르: 액션, 모험, 판타지
감독: 데스틴 크리튼
출연: 시무 리우, 양조위, 아콰피나
기적
9.23
장르: 드라마
감독: 이장훈
출연: 박정민, 이성민, 윤아, 이수경

SLACK API 수정

User Interaction 추가



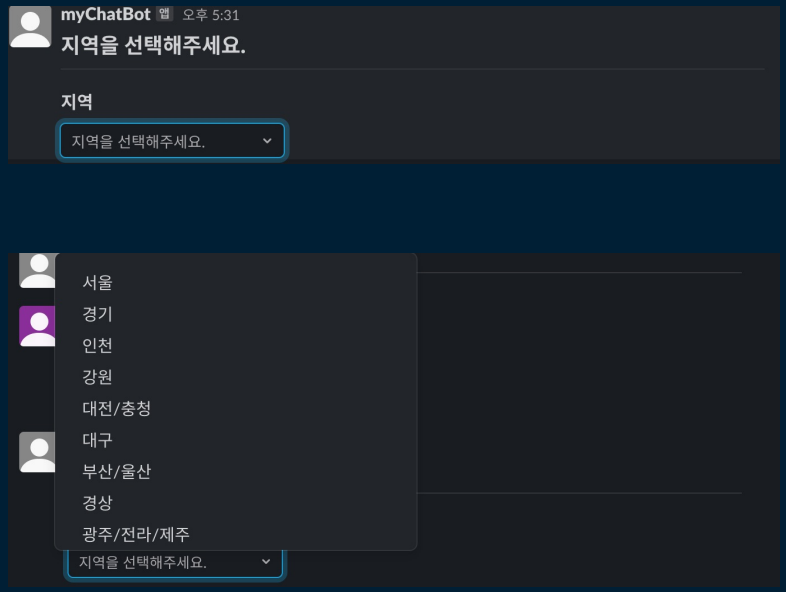
⇒ Request URL: <Forwarding URL> + ”/user-select”

⇒ ex. https://ec22-118-33-100-66.ngrok.io/user-select

Interaction 추가

REFERENCE : info.json

```
{
    "서울" : {
        "code": "01",
        "theaters" : {
            "CGV강남" : "0056",
            "CGV강변" : "0001",
            "CGV건대입구" : "0229",
            "CGV구로" : "0010",
            "CGV대학로" : "0063",
            "CGV동대문" : "0252",
            "CGV등촌" : "0230",
            "CGV명동" : "0009",
            "CGV명동역 씨네라이브러리" : "00105",
            "CGV목동" : "0011",
            "CGV미아" : "0057",
            "CGV불광" : "0030"
        }
    },
    "경기" :{
        "code" : "02"
    },
    "인천" :{
        "code" : "03"
    },
    "강원" :{
        "code" : "04"
    },
    "대전/충청" :{
        "code" : "05"
    },
    "대구" :{
        "code" : "06"
    },
    "부산/울산" :{
        "code" : "07"
    },
    "경상" :{
        "code" : "08"
    },
    "광주/전라/제주" :{
        "code" : "09"
    }
}
```



Interaction 추가

REFERENCE : 3_user_interaction.py

```
30      # slack에서 발생한 event를 통한 request에 대한 핸들링
31      if "event" in slack_event:
32          event_type = slack_event["event"]["type"]
33          # bot_mention일 경우에 대한 핸들링
34          if event_type == 'app_mention':
35              try:
36                  # 멘션을 남긴 채널 읽어오기
37                  channel = slack_event['event']['channel']
38                  # 유저가 멘션과 함께 남긴 텍스트 읽어오기
39                  user_query = slack_event['event']['blocks'][0]['elements'][0]['elements
40
41                  if user_query.strip() == '/영화정보':
42                      movies = crawling.get_current_movies()
43                      block = build_blocks(movies)
44                      slack_handler.post_layout_message(blocks=block, channel=channel)
45
46                  if user_query.strip() == '/상영정보':
47                      block = build_select_block()
48                      slack_handler.post_layout_message(blocks=block, channel=channel)
49
50                  # 정상적으로 완료했음에 대한 http response
51                  return make_response("response made :)", 200, )
52              except IndexError:
53                  # 멘션은 했지만 텍스트는 남기지 않은 경우에 대한 에러.
54                  # do nothing
55                  pass
56          # 그 외 event에 대한 핸들링: 404 error
57          msg = f"[{event_type}] cannot find event handler"
58          return make_response(msg, 404, {"X-Slack-No-Retry": 1 })
59
60
61      # 그 외 request 핸들링: 404 error
62      return make_response("No Slack request events", 404, {"X-Slack-No-Retry": 1 })
```

Interaction 추가

REFERENCE : 3_user_interaction.py

```python
def build_select_block():
    # 블록 담을 리스트
    block = []
    # 헤더 블록
    header = {}
    header['type']  = 'header'
    header['text'] = {}
    header['text']['type'] = 'plain_text'
    header['text']['text'] = '지역을 선택해주세요.'
    block.append(header)
    # divider 블록
    divider = {}
    divider['type'] = 'divider'
    block.append(divider)
    # 지역 블록
    options = json.load(open('info.json', 'rb'), encoding='CP949')
    input_block = {}
    input_block['type'] = "input"
    input_block['element'] = {}
    input_block['element']['type'] = 'static_select'
    input_block['element']['placeholder'] = {}
    input_block['element']['placeholder']['type'] = "plain_text"
    input_block['element']['placeholder']['text'] = '지역을 선택해주세요.'
    input_block['element']['options'] = []
    for area in options:
        temp = {}
        temp['text'] = {}
        temp['text']['type'] = "plain_text"
        temp['text']['text'] = area
        temp['value'] = options[area]['code']
        input_block['element']['options'].append(temp)
    input_block['element']['action_id'] = 'static_select-action'
    input_block['label'] = {}
    input_block['label']['type'] = 'plain_text'
    input_block['label']['text'] = "지역"
    block.append(input_block)
    print(block)
    return block
```

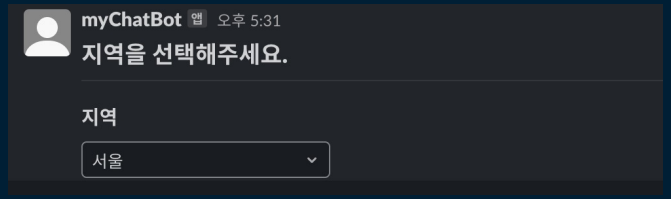
Interaction 추가

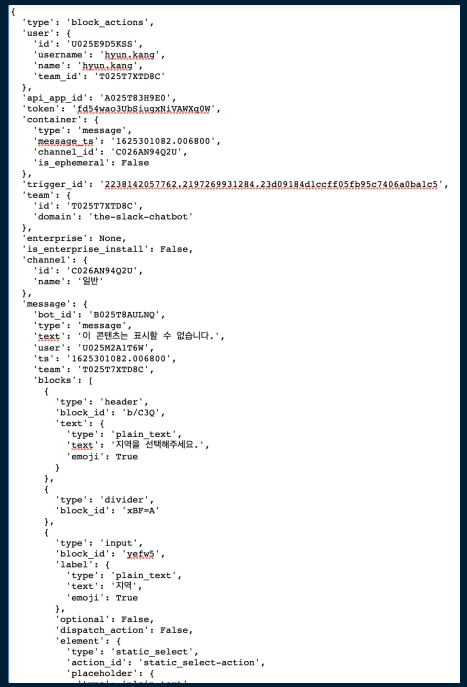
REFERENCE : 3_user_interaction.py

```
18  @app.route('/user-select', methods=['POST', 'GET'])
19  def handle_interaction():
20      slack_event = json.loads(request.form['payload'])
21      return make_response("response made :)", 200, )
```

Interaction 추가

Interaction이 생겼을 때 설정한 url로 날아오는 request 확인





Interaction 추가

Interaction Handling 추가

REFERENCE : 3_user_interaction.py

```
14  areaCode = ''
15  theaterCode = ''
16
```

```
18  @app.route('/user-select', methods=['POST', 'GET'])
19  def handle_interaction():
20      slack_event = json.loads(request.form['payload'])
21      # 만약 지역 선택 블록에서 interaction request가 온 경우
22      if slack_event['actions'][0]['placeholder']['text'] == '지역을 선택해주세요.':
23          area = slack_event['actions'][0]['selected_option']['text']['text']
24          areaCode = slack_event['actions'][0]['selected_option']['value']
25          block = build_theater_block(area)
26          slack_handler.post_layout_message(blocks=block, channel=slack_event['channel']['id'])
27
28      return make_response("response made :)", 200, )
```

Interaction 추가

Interaction Handling 추가

REFERENCE : 3_user_interaction.py

```python
def build_theater_block(area):
    # 블록 담을 리스트
    block = []
    # 헤더 블록
    header = {}
    header['type']  = 'header'
    header['text'] = {}
    header['text']['type'] = 'plain_text'
    header['text']['text'] = '극장을 선택해주세요.'
    block.append(header)
    # divider 블록
    divider = {}
    divider['type'] = 'divider'
    block.append(divider)
    # 영화관 블록
    options = json.load(open('info.json', 'rb'), encoding='CP949')
    input_block = {}
    input_block['type'] = "input"
    input_block['element'] = {}
    input_block['element']['type'] = 'static_select'
    input_block['element']['placeholder'] = {}
    input_block['element']['placeholder']['type'] = "plain_text"
    input_block['element']['placeholder']['text'] = '극장을 선택해주세요.'
    input_block['element']['options'] = []
    print(area)
    for theater, value in options[area]['theaters'].items():
        temp = {}
        temp['text'] = {}
        temp['text']['type'] = "plain_text"
        temp['text']['text'] = theater
        temp['value'] = value
        input_block['element']['options'].append(temp)
    input_block['element']['action_id'] = 'static_select-action'
    input_block['label'] = {}
    input_block['label']['type'] = 'plain_text'
    input_block['label']['text'] = "극장"
    block.append(input_block)
    return block
```

Interaction 추가

Interaction Handling 추가

결과:

Interaction 추가

Interaction Handling 추가

REFERENCE : 3_user_interaction.py

```
18  @app.route('/user-select', methods=['POST', 'GET'])
19  def handle_interaction():
20      slack_event = json.loads(request.form['payload'])
21      print(slack_event)
22      # 만약 지역 선택 블록에서 interaction request가 온 경우
23      if slack_event['actions'][0]['placeholder']['text'] == '지역을 선택해주세요.':
24          area = slack_event['actions'][0]['selected_option']['text']['text']
25          areaCode = slack_event['actions'][0]['selected_option']['value']
26          block = build_theater_block(area)
27          slack_handler.post_layout_message(blocks=block, channel=slack_event['channel']['id'])
28
29      # 만약 극장 선택 블록에서 interaction request가 온 경우
30      if slack_event['actions'][0]['placeholder']['text'] == '극장을 선택해주세요.':
31          theaterCode = slack_event['actions'][0]['selected_option']['value']
32          print(theaterCode)
33
34      return make_response("response made :)", 200, )
```

Interaction 추가

Interaction Handling 추가

REFERENCE : 3_user_interaction.py

```
31
32      # 만약 극장 선택 블록에서 interaction request가 온 경우
33      if slack_event['actions'][0]['placeholder']['text'] == '극장을 선택해주세요.':
34          date = datetime.datetime.now().strftime("%Y%m%d")
35          global theaterCode
36          theaterCode = slack_event['actions'][0]['selected_option']['value']
37          slack_handler.post_slack_message(channel=slack_event['channel']['id'], message=f'http://www.cgv.co.kr/reserve/\
38              show-times/?areacode={areaCode}&theaterCode={theaterCode}&date={date}')
```

Interaction 추가

myChatBot 앱 오후 11:31

지역을 선택해주세요.

지역

서울

극장을 선택해주세요.

극장

CGV등촌

myChatBot 앱 오후 11:48

http://www.cgv.co.kr/reserve/show-times/?areacode=&theaterCode=0230&date=20210703 (편집됨)

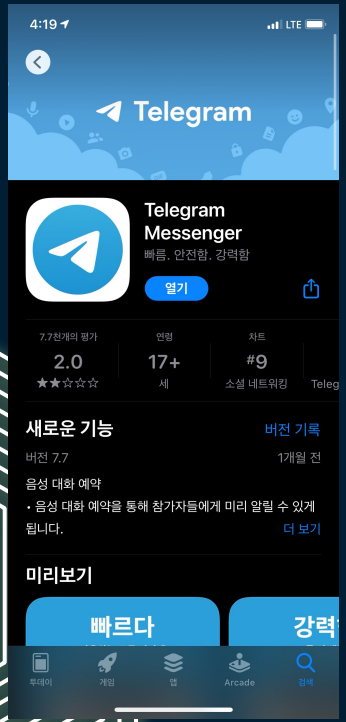
cgv.co.kr

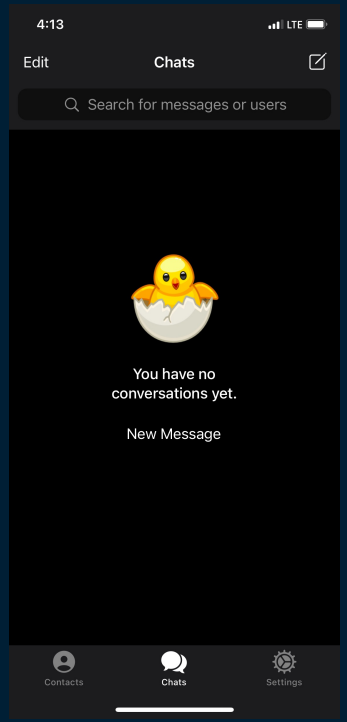
영화 그 이상의 감동. CGV

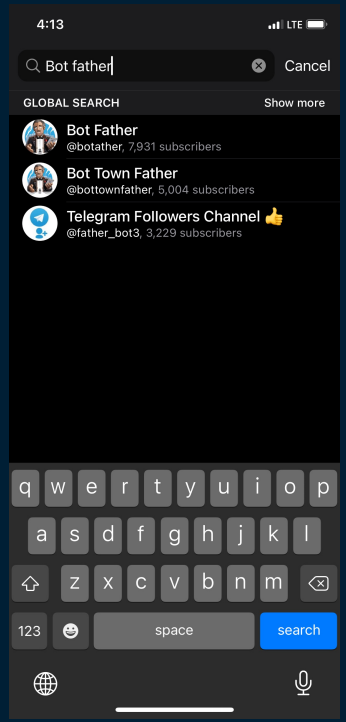
CGV는 선진화된 관람문화와 최고의 서비스로 고객에게 잊을 수 없는 감동을 선사합니다.
CGV홈페이지를 통해 영화 예매뿐만 아니라 그 이상의 서비스와 감동을 전달하고, 다양한 즐거움과 특별한 경험을 제공하고자 합니다. (100kB)

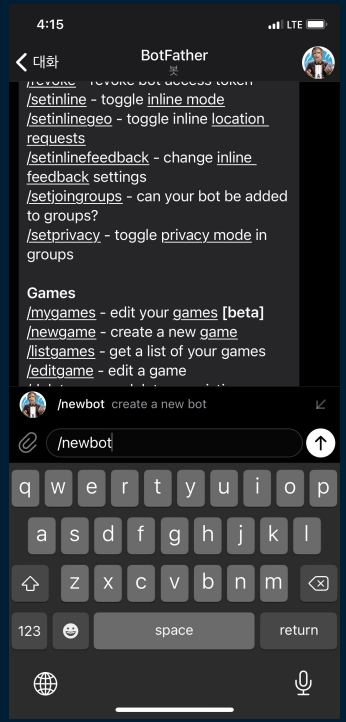
번외 : Telegram Bot

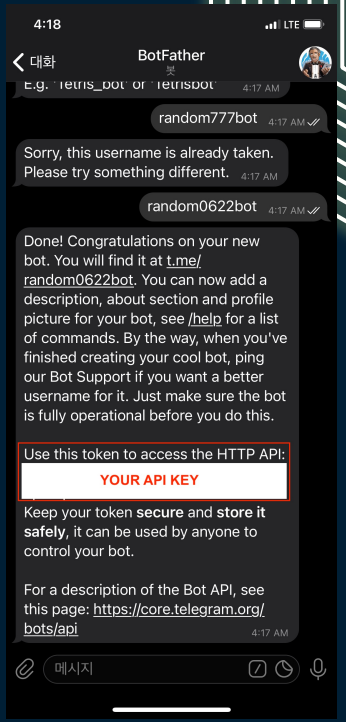
SETUP


Telegram
Telegram Messenger
빠름. 안전함. 강력함
열기
2.0
17+
#9
새로운 기능
버전 기록
버전 7.7
1개월 전
음성 대화 예약
• 음성 대화 예약을 통해 참가자들에게 미리 알릴 수 있게 됩니다.
더 보기
미리보기
빠르다
강력
Arcade


Edit
Chats
Search for messages or users
You have no conversations yet.
New Message
Contacts
Chats
Settings


Bot father
Cancel
GLOBAL SEARCH
Show more
Bot Father
@botather, 7,931 subscribers
Bot Town Father
@bottownfather, 5,004 subscribers
Telegram Followers Channel
@father_bot3, 3,229 subscribers
space
search


대화
BotFather
/setinline - toggle inline mode
/setinlinegeo - toggle inline location requests
/setinlinefeedback - change inline feedback settings
/setjoingroups - can your bot be added to groups?
/setprivacy - toggle privacy mode in groups
Games
/mygames - edit your games [beta]
/newgame - create a new game
/listgames - get a list of your games
/editgame - edit a game
/newbot create a new bot
/newbot
space
return


대화
BotFather
random777bot
Sorry, this username is already taken. Please try something different.
random0622bot
Done! Congratulations on your new bot. You will find it at t.me/random0622bot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.
Use this token to access the HTTP API:
YOUR API KEY
Keep your token secure and store it safely, it can be used by anyone to control your bot.
For a description of the Bot API, see this page: https://core.telegram.org/bots/api
메시지

자동 응답하기

REFERENCE : credentials.py

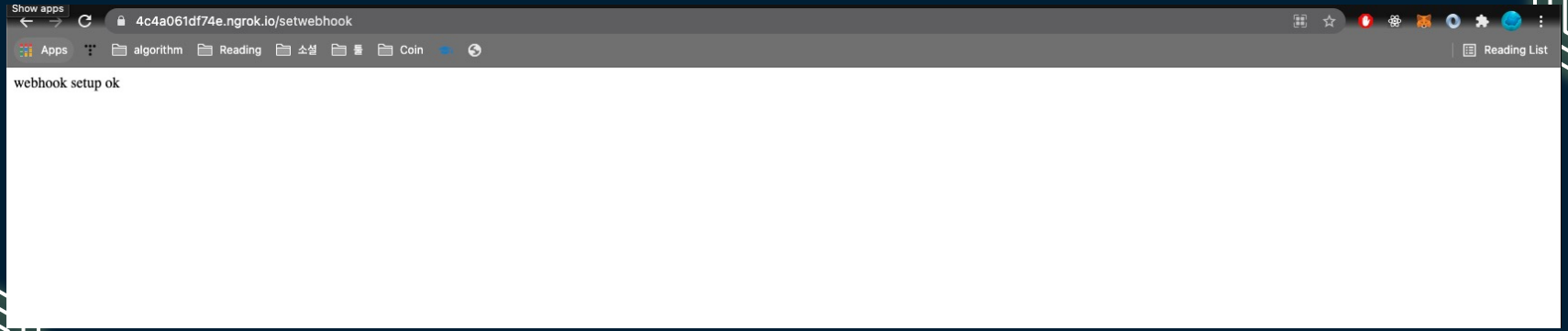
```
bot_token = "1804380674:AAGJrOaqeXCWvDHee-hpKUqTrINexSDWHRE"
URL = "https://4c4a061df74e.ngrok.io"
```

REFERENCE : telegramBot.py

```
from flask import Flask, request

# pip install python-telegram-bot
# pip install telegram
import telegram
from credentials import bot_token, URL

TOKEN = bot_token
bot = telegram.Bot(token=TOKEN)

app = Flask(__name__)

@app.route('/setwebhook', methods=['GET', 'POST'])
def set_webhook():
    # 텔레그램 봇과 연결 세팅
    s = bot.setWebhook(f'{URL}/{TOKEN}')
    if s:
        return "webhook setup ok"
    else:
        return "webhook setup failed"

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

자동 응답하기

결과:



=> <본인 url>/setwebhook 들어가면 webhook setup ok 라고 나와야 함.

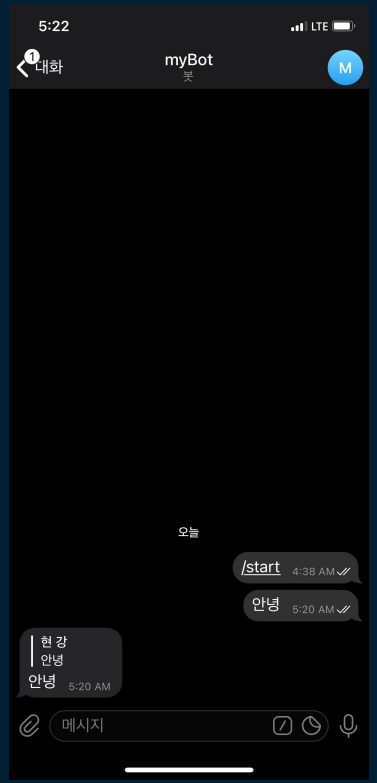
자동 응답하기

REFERENCE : telegramBot.py

```python
# 실제로 텔레그램에서 request를 보내는 URL
# 텔레그램의 경우 <URL>/<TOKEN> 으로 http request 전송
@app.route(f'/{TOKEN}', methods=['POST'])
def autoResponse():
    # request를 json 형식으로 받은 다음 Telegram object로 변환
    update = telegram.Update.de_json(request.get_json(force=True), bot)

    chat_id = update.message.chat.id # request를 발생시킨 채팅방의 id
    msg_id = update.message.message_id # request를 발생시킨 메시지의 id

    # 유저가 보낸 메시지 읽기 (텔레그램은 utf-8를 사용하기에 서버에서 사용하는 unicode와 호환을 위해 utf-8으로 인코딩을 한 후 디코드)
    text = update.message.text.encode('utf-8').decode()

    try:
        # 유저가 보낸 메시지에 대한 답장으로 유저 메시지 반복하기
        bot.sendMessage(chat_id=chat_id, text=text, reply_to_message_id=msg_id)
    except Exception:
        # 에러가 발생했을 때
        bot.sendMessage(chat_id=chat_id, text="문제가 발생하였습니다 :(", reply_to_message_id=msg_id)

    return "ok"
```

자동 응답하기

결과:



자동 응답하기

REFERENCE : telegramBot.py

```python
@app.route(f'/{TOKEN}', methods=['POST'])
def autoResponse():
    # request를 json 형식으로 받은 다음 Telegram object로 변환
    update = telegram.Update.de_json(request.get_json(force=True), bot)

    chat_id = update.message.chat.id # request를 발생시킨 채팅방의 id
    msg_id = update.message.message_id # request를 발생시킨 메시지의 id

    # 유저가 보낸 메시지 읽기 (텔레그램은 utf-8를 사용하기에 서버에서 사용하는 unicode와 호환을 위해 utf-8으로 인코딩을 한 후 디코드)
    text = update.message.text.encode('utf-8').decode()

    ##########################################################################################
    ##############                                                          ##############
    ##############                    NLP와 같은 여러가지 계산 추가                    ##############
    ##############                                                          ##############
    ##########################################################################################

    try:
        # 유저가 보낸 메시지에 대한 답장으로 유저 메시지 반복하기
        bot.sendMessage(chat_id=chat_id, text=text, reply_to_message_id=msg_id)
    except Exception:
        # 에러가 발생했을 때
        bot.sendMessage(chat_id=chat_id, text="문제가 발생하였습니다 :(", reply_to_message_id=msg_id)
```

채팅에 추가하기

REFERENCE : telegramBot.py

```python
# 실제로 텔레그램에서 request를 보내는 URL
# 텔레그램의 경우 <URL>/<TOKEN> 으로 http request 전송
@app.route(f'/{TOKEN}', methods=['POST'])
def autoResponse():
    # request를 json 형식으로 파싱
    resp = request.get_json(force=True)

    if 'message' in resp.keys():
        msgtext = resp["message"]["text"]
        sendername = resp["message"]["from"]["first_name"]
        chat_id = resp["message"]["chat"]["id"]
    elif 'channel_post' in resp.keys():
        msgtext = resp["channel_post"]["text"]
        sendername = resp["channel_post"]["chat"]["username"]
        chat_id = resp["channel_post"]["chat"]["id"]
    try:
        # 유저가 보낸 메시지에 대한 답장으로 유저 메시지 반복하기
        bot.sendMessage(chat_id=chat_id, text=msgtext)
    except Exception:
        # 에러가 발생했을 때
        bot.sendMessage(chat_id=chat_id, text="문제가 발생하였습니다 :(")

    return "ok"
```

이미지 보내기

REFERENCE : telegramImageBot.py

```python
from flask import Flask, request

# pip install python-telegram-bot
# pip install telegram
import telegram
from credentials import bot_token, URL

TOKEN = bot_token
bot = telegram.Bot(token=TOKEN)

app = Flask(__name__)

# 텔레그램 봇과 url 연동
@app.route('/setwebhook', methods=['GET', 'POST'])
def set_webhook():
    s = bot.setWebhook(f'{URL}/{TOKEN}')
    if s:
        return "webhook setup ok"
    else:
        return "webhook setup failed"

# 실제로 텔레그램에서 request를 보내는 URL
# 텔레그램의 경우 <URL>/<TOKEN> 으로 http request 전송
@app.route(f'/{TOKEN}', methods=['POST'])
def autoResponse():
    # request를 json 형식으로 받은 다음 Telegram object로 변환
    update = telegram.Update.de_json(request.get_json(force=True), bot)

    chat_id = update.message.chat.id # request를 발생시킨 채팅방의 id
    msg_id = update.message.message_id # request를 발생시킨 메시지의 id

    # 유저가 보낸 메시지 읽기 (텔레그램은 utf-8를 사용하기에 서버에서 사용하는 unicode와 호환을 위해 utf-8으로 인코딩을 한 후 디코드)
    text = update.message.text.encode('utf-8').decode()

    try:
        # 이미지 보내기
        bot.send_photo(chat_id=chat_id, photo='https://telegram.org/img/t_logo.png')
    except Exception:
        # 에러가 발생했을 때
        bot.sendMessage(chat_id=chat_id, text="문제가 발생하였습니다 :(", reply_to_message_id=msg_id)

    return "ok"

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