

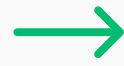


세바기 워크샵

나만의 챗봇 만들기(RASA 활용편)

2024.09.21

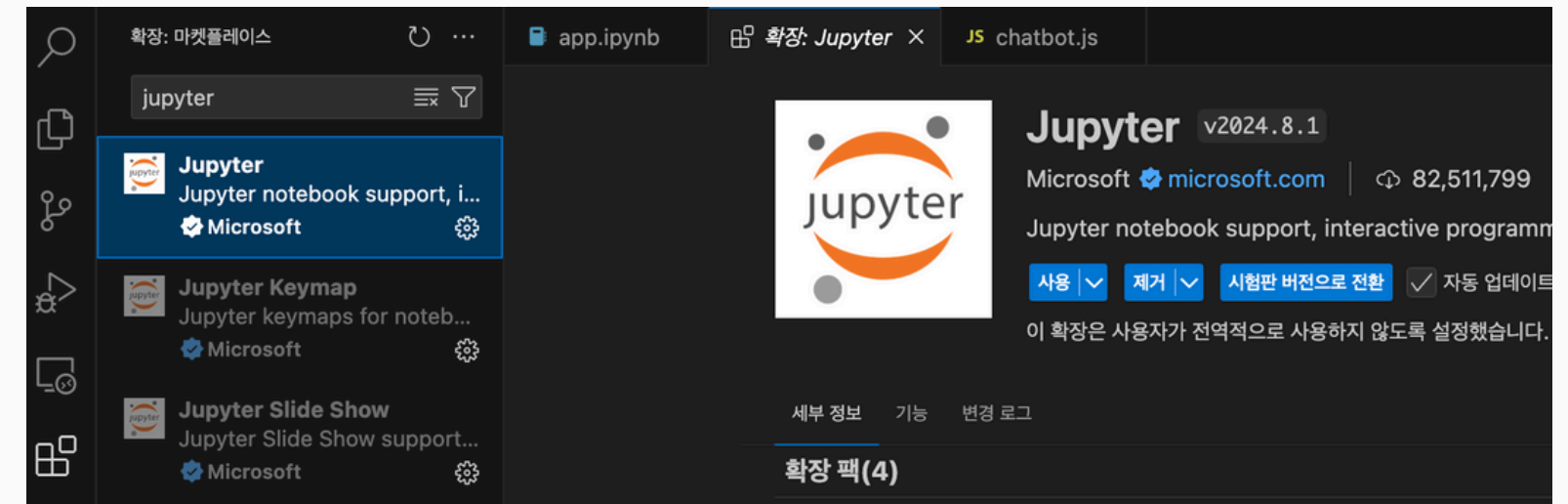
번호	플랫폼(언어)	주요특징	추가정보	제작회사	오픈소스
1	Microsoft Bot (다양한 언어)	다양한 플랫폼과 통합 가능, 오픈 소스 라이브러리 지원	Azure 서비스와 통합 가능	Microsoft	Yes(일부)
2	Tidio (JavaScript)	실시간 채팅과 챗봇 기능 통합	사용자 친화적인 인터페이스	Tidio	No
3	Rasa (Python)	강력한 NLU, 대화 관리, 로컬 호스팅 지원	오픈 소스, 커스터마이징 용이	Rasa Technologies	Yes
4	Botpress (Node.js)	비주얼 인터페이스, NLU 기능 제공	GUI 기반의 대화 흐름 설계 가능	Botpress Inc.	Yes
5	DeepPavlov (Python)	자연어 처리 및 대화형 AI 모델 지원	다양한 사전 훈련된 모델 제공	Neural Networks Group (MIPT)	Yes
6	ChatterBot (Python)	머신러닝 기반, 간단한 사용	간단한 설치 및 학습 방법 제공	Gunther Cox	Yes
7	Jasper (Python)	머신러닝 기반, 대화 흐름 관리	대화 이력을 바탕으로 학습 가능	Jasper	Yes
8	BotMan (PHP)	다양한 메시징 플랫폼과 쉽게 통합 가능	Laravel과의 통합 지원	Marcel Pociot	Yes



SECTION 01

환경설정

1. Vs code에 Jupyter 설치



2. Python, Anaconda 설치



- `conda create --name myenv python=3.8`
- `conda activate myenv`
- `conda install numpy`
- `conda deactivate`
- `conda env list`

3. Chat GPT 오류나 문법수정

4. Git Hub 디폴트 파일 다운

<https://github.com/Sebagi-community/Workshop/>



1. Rasa 설치 및 환경설정

종속성과 호환 문제가 있을수
있기때문에 **터미널** 실행 후

(<http://localhost:5005> 확인)

Jupyter notebook과 Anaconda 설치완료 후,

설치_	<code>pip install rasa</code>
초기화_	<code>rasa init</code>
실행_	<code>rasa run --enable-api</code>

pip를 사용하여 Rasa를 설치하고,
가상 환경을 사용하는 것이 좋습니다.

```
# 가상환경 생성 및 활성화_  
python -m venv rasa_env  
source rasa_env/bin/activate
```

```
# 윈도우의 경우_ `source` 대신 `.\rasa_env\Scripts\activate`
```

```
# Rasa 설치_  
pip install rasa
```

2. 음성인식 및 기능추가

음성 입력을 **Web Speech API**를
통해 구현할 수 있습니다

```
let recognition = new (  
  window.SpeechRecognition || window.webkitSpeechRecognition)();  
  
$('#voiceInputButton').click(function() { recognition.start(); });  
  
recognition.onresult = function(event) {  
  let voiceMessage = event.results[0][0].transcript;  
  $('#userInput').val(voiceMessage);  
  $('#chatForm').submit(); // 음성 인식 결과를 Rasa 서버로 전송  
};
```

3. 웹훅 설정

Rasa에서 **웹훅을 설정하여 API**
요청을 처리하는 방식을 설정

endpoints.yml 파일 예시

```
http://localhost:5005/webhooks/rest/webhook
```

로컬 웹 서버 실행

```
python -m http.server 8000
```



4. API 요청 처리

음성 데이터를 Rasa로 전송 후
처리된 결과를 **클라이언트로 반환**

```
// Rasa 웹훅 엔드포인트를 호출하는 코드 (JavaScript)
```

```
const rasaWebhookUrl =  
'http://localhost:5005/webhooks/rest/webhook';
```

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

```
@app.route('/webhook', methods=['POST'])
```

```
def webhook():
```

```
data = request.get_json() user_message = data.get('message')
```

```
# Rasa 또는 다른 서버로 메시지를 전달하여 처리
```

```
# 여기서는 예시로 간단히 응답 생성
```

```
response_message = f"Rasa received your message: {user_message}"
```

```
return jsonify([{'text': response_message}])
```

```
if __name__ == '__main__': app.run(port=5005)
```



SECTION 02

구조 및 대화데이터 생성

파일/디렉토리	설명
actions.py	사용자 정의 액션을 정의하는 파일. API 호출 등의 로직을 포함.
config.yml	Rasa 모델의 파이프라인과 정책을 설정하는 파일.
credentials.yml	Rasa 서버와 외부 서비스 간의 인증 정보를 포함하는 파일.
domain.yml	인텐트, 엔티티, 슬롯, 액션 및 응답을 정의하는 파일.
endpoints.yml	Rasa 서버의 사용자 정의 액션 서버와의 엔드포인트를 설정하는 파일.
models/	학습된 Rasa 모델이 저장되는 디렉토리.
data/	NLU 및 스토리 데이터를 포함하는 디렉토리.
├── nlu.yml	사용자 입력 예시와 인텐트를 정의하는 파일.
└── stories.yml	대화의 흐름을 정의하는 파일.
tests/	대화 흐름 테스트를 위한 스토리 데이터를 저장하는 디렉토리.
└── test_stories.yml	테스트용 스토리 데이터.

GITHUB 에서 제공된 파일들	
static/	정적 파일을 저장하는 디렉토리.
├── images/	이미지 파일을 저장하는 디렉토리.
├── chatbot.js	챗봇의 동작을 제어하는 자바스크립트 파일.
├── jquery.min.js	jQuery 라이브러리 파일.
└── style.css	웹 페이지의 스타일을 정의하는 CSS 파일.
template/	웹 페이지 템플릿을 저장하는 디렉토리.
└── index.html	챗봇 사용자 인터페이스를 위한 HTML 파일.
app.py	Flask 애플리케이션 파일로, 챗봇 인터페이스를 제공.
actions/	(선택 사항) 사용자 정의 액션을 위한 추가적인 Python 모듈을 저장하는 디렉토리.
├── __init__.py	이 디렉토리가 Python 패키지임을 인식시키는 파일.
└── actions.py	추가적인 사용자 정의 액션 로직이 들어가는 파일.



1. actions.py

```
from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher

class ActionHelloWorld(Action):
    def name(self) -> str:
        return "action_hello_world"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: dict) -> list:
        dispatcher.utter_message(text="Hello! How can I help you today?")
        return []
```

2. config.yml

```
language: ko
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: DIETClassifier
epochs: 100
- name: EntitySynonymMapper
- name: ResponseSelector
epochs: 100
policies:
- name: MemoizationPolicy
- name: TEDPolicy
max_history: 5
assistant_id: 20240921-105101-fixed-sycamore
```



3. credentials.yml

```
# 예시: Slack과 같은 외부 서비스와의 연결 정보slack:verification_token:  
  
  "YOUR_SLACK_VERIFICATION_TOKEN"signing_secret:  
  "YOUR_SLACK_SIGNING_SECRET"
```



4. domain.yml

intents :

- greet # 인사하기
- goodbye # 작별인사
- affirm # 긍정
- deny # 부정

entities :

- location # 위치 정보

slots :

location:

type: text # 슬롯 타입 설정

influence_conversation: false

대화에 영향을 주지 않음

responses :

utter_greet:

- text: "안녕하세요! 어떻게 도와드릴까요?" # 인사 응답

utter_goodbye:

- text: "안녕히 가세요! 좋은 하루 되세요!" # 작별 응답

actions:

- action_hello_world # 사용자 정의 액션



5. endpoints.yml

```
action_endpoint:  
url: "http://localhost:5055/webhook" # 액션 서버의 URL 설정
```



6. nlu.yml

```
version: "2.0"
nlu:
  - intent: greet
    examples: |
      - 안녕하세요
      - 안녕
      - 여보세요
      - 좋은 아침이에요
  - intent: goodbye
    examples: |
      - 안녕히 가세요
      - 잘 가요
      - 나중에 봐요
```



```
version: "2.0"
stories:
- story: test greet and goodbye
  steps:
  - intent: greet
  - action: action_hello_world
  - intent: goodbye
  - action: utter_goodbye
```


version: "2.0"

stories:

- story: test greet and goodbye

steps:

- intent: greet
- action: action_hello_world
- intent: goodbye
- action: utter_goodbye

점검할 사항과 제안

<점검할 사항>

nlu.yml 및 stories.yml 파일 구성 : 이 두 파일의 구성과 데이터가 Rasa의 학습 규칙에 맞게 되어 있는지 확인하세요. 제대로 설정되지 않으면 NLU 모델이나 대화 흐름이 올바르게 동작하지 않을 수 있습니다.

endpoints.yml 설정 확인 : 이 파일에 Rasa 서버와 연결될 웹훅 엔드포인트가 제대로 설정되어 있는지 확인하세요. 예를 들어, 올바른 URL이나 포트가 입력되었는지, Rasa 서버가 해당 엔드포인트를 인식하는지 체크하세요.

actions.py의 커스텀 액션 : 커스텀 액션이 제대로 작동하도록 domain.yml과 액션 파일이 연결되었는지 확인해야 합니다. 추가한 액션이 domain.yml에 등록되어 있지 않으면 동작하지 않을 수 있습니다.

가상환경(venv) : 가상환경 내의 필요한 Rasa 패키지가 설치되어 있는지 확인하세요. pip freeze 명령으로 필요한 패키지가 누락되지 않았는지 검토하면 좋습니다.

<제안>

코드의 통합성 확인 : 자바스크립트(chatbot.js)와 Rasa 엔드포인트 설정이 제대로 연결되어 있는지 확인하세요. fetch나 axios 같은 라이브러리를 사용해 API 요청을 보낼 때 오류가 없는지 점검하세요.

domain.yml과 actions.py 연결성 : 커스텀 액션이 필요하다면, domain.yml 파일에서 올바르게 등록되고 있는지, 그리고 actions.py에서 기능이 잘 구현되었는지 확인해야 합니다.



nlu.yml 및 stories.yml 점검 코드

```
import yaml

def check_nlu_stories():
    files_to_check = ['data/nlu.yml', 'data/stories.yml']
    for file in files_to_check:
        try:
            with open(file, 'r', encoding='utf-8') as f:
                data = yaml.safe_load(f)
                # nlu 파일에 intent와 examples 확인
                if 'nlu' in data:
                    print(f"{file}: Found {len(data['nlu'])} intents.")
                    for intent in data['nlu']:
                        if 'intent' not in intent or 'examples' not in intent:
                            print(f"Warning: {file} has an invalid format in an intent.")
                # stories 파일에서 story와 steps 확인
                elif 'stories' in data:
                    print(f"{file}: Found {len(data['stories'])} stories.")
                    for story in data['stories']:
                        if 'steps' not in story:
                            print(f"Warning: {file} has an invalid format in a story.")
                else:
                    print(f"Warning: {file} is missing key sections.")
        except Exception as e:
            print(f"Error reading {file}: {e}")

check_nlu_stories()
```

endpoints.yml 설정 점검 코드

```
import yaml

def check_endpoints():
    try:
        with open('endpoints.yml', 'r', encoding='utf-8') as f:
            data = yaml.safe_load(f)
            if 'rest' in data and 'url' in data['rest']:
                print(f"Webhook URL: {data['rest']['url']}")
            else:
                print("Warning: Webhook URL is missing in endpoints.yml")
    except Exception as e:
        print(f"Error reading endpoints.yml: {e}")

check_endpoints()
```

커스텀 액션 점검 코드 (**actions.py**)

```
import yaml

def check_actions_in_domain():
    try:
        # actions.py에서 정의된 커스텀 액션 함수 목록 수집
        with open('actions.py', 'r', encoding='utf-8') as f:
            actions = [line.split(' ')[1] for line in f if line.startswith('class')]

        # domain.yml에서 actions 확인
        with open('domain.yml', 'r', encoding='utf-8') as f:
            domain_data = yaml.safe_load(f)
            if 'actions' in domain_data:
                domain_actions = domain_data['actions']
                # actions.py에 정의된 액션이 domain.yml에 있는지 확인
                for action in actions:
                    if action not in domain_actions:
                        print(f"Warning: {action} is not registered in domain.yml")
                    else:
                        print(f"{action} is correctly registered.")
            else:
                print("Warning: No actions found in domain.yml")
    except Exception as e:
        print(f"Error checking actions in domain.yml: {e}")

check_actions_in_domain()
```

가상환경 점검 코드

```
import os
import subprocess

def check_virtual_env():
    # 가상환경 패키지 목록 가져오기
    installed_packages = subprocess.check_output([os.path.join('.venv', 'bin', 'pip'),
    'freeze']).decode('utf-8')
    installed_packages = set([pkg.split('==')[0] for pkg in
    installed_packages.splitlines()])

    # requirements.txt 파일에서 요구되는 패키지 목록 가져오기
    try:
        with open('requirements.txt', 'r') as f:
            required_packages = set([pkg.split('==')[0] for pkg in f.readlines()])
    except FileNotFoundError:
        print("Error: requirements.txt file not found.")
        return

    # 빠진 패키지 확인
    missing_packages = required_packages - installed_packages
    if missing_packages:
        print(f"Warning: The following packages are missing from the virtual
        environment: {missing_packages}")
    else:
        print("All required packages are installed.")

check_virtual_env()
```

SECTION 03

배포 및 테스트

1. 가상환경 설정 및 필요한 패키지 설치

주의: requirements.txt 파일이 없을 경우, pip freeze > requirements.txt 명령으로 현재 패키지 목록을 저장할 수 있습니다.

```
# 가상환경 활성화 (Linux/Mac)
source .venv/bin/activate
```

```
# 가상환경 활성화 (Windows)
.venv\Scripts\activate
```

```
# Rasa 및 필요한 패키지 설치
pip install -r requirements.txt
```

2. Rasa 서버 실행

rasa 명령을 사용해 **NLU 모델을 학습**시키고 서버를 실행합니다.

```
# NLU 및 대화 모델 학습
rasa train
```

```
# Rasa 서버 실행
rasa run --enable-api --cors "*" --debug
```

3. 액션 서버 실행 (커스텀 액션이 있을 경우)

actions.py 파일에 커스텀 액션이 있는 경우, 액션 서버도 따로 실행해야 합니다.

```
# 액션 서버 실행  
rasa run actions
```

```
# 간단한 웹 서버 실행 (HTML 파일을 확인하기 위해)  
python -m http.server 8000
```

4. 테스트 파일 실행

tests/ 디렉토리에 있는 스토리 테스트 파일(test_stories.yml)을 사용해 자동화된 테스트를 실행

```
# Rasa 스토리 테스트 실행  
rasa test
```

```
# 도커 이미지 빌드  
docker build -t rasa-chatbot .
```

```
# 도커 컨테이너 실행  
docker run -p 5005:5005 rasa-chatbot
```



5. Heroku CLI 설치 및 로그인

actions.py 파일에 커스텀 액션이 있는 경우, 액션 서버도 따로 실행해야 합니다.

```
heroku login
```

6. 테스트 자동화

테스트를 자동화하고 문제를 빠르게 찾기 위해, Rasa의 테스트 명령을 주기적으로 실행하거나 CI/CD 파이프라인에 통합

- 테스트 스토리: tests/test_stories.yml에 작성된 스토리를 기반으로 자동화된 테스트를 실행합니다.
- CI/CD 통합: GitHub Actions나 Jenkins를 통해 Rasa 프로젝트를 배포 전에 자동으로 테스트할 수 있도록 설정



THANK YOU

P. 010.8826.3457

E. kds1028@nate.com

H. <https://buly.kr/7QKWLLN>