테스팅 보고서

오픈소스 기여

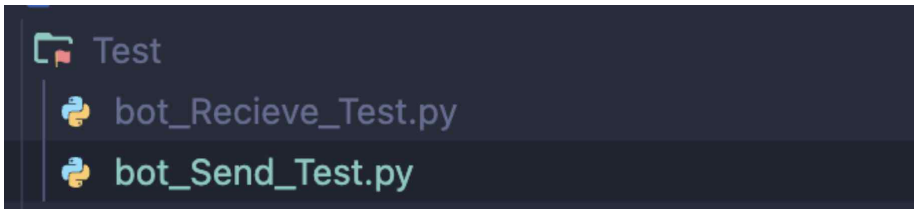
-단위 테스팅

-전체 테스팅 시나리오

32173154 이승현

32172086 석홍준

- 챗 봇 단위테스팅 설계



키우미 봇에서 챗봇 이 실행하는 동작인 메세지 수신과 송신으로 단위를 나누어 이를 Test폴더에 따로 두어 관리.

-> 메세지 수신의경우 메세지를 터미널에 출력 할 수 있으나 송신의 경우 직접 챗봇에 오는 답변을 확인하는 테스팅이 필요.

```python
import telegram
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters
import subprocess

# 봇 토큰을 입력하세요.
TOKEN = '6068822535:AAEdX6RZ9_7N85NtDsyYornSU5LeoH9tEdI'
# 텔레그램 봇 객체 생성
bot = telegram.Bot(token=TOKEN)

#Start 명령어에 대한 메세지전송
def start(update, context):
    context.bot.send_message(chat_id=update.message.chat_id, text='안녕하세요! 키우미를 시작합니다.')

#메세지를 받아 이를 그대로 재전송
def echo(update, context):
    text = update.message.text
    chat_id=update.message.chat_id
    print('Message received from {chat_id}: {text}')
    #subprocess를 통하여 bot_send_Test.py를 호출
    subprocess.run(['python', 'bot_Send_Test.py', TOKEN,chat_id, text])

def main():
    updater = Updater(TOKEN, use_context=True)
    dp = updater.dispatcher
    dp.add_handler(CommandHandler('start', start))
    dp.add_handler(MessageHandler(Filters.text, echo))
    updater.start_polling()
    updater.idle()

if __name__ == '__main__':
    main()
```

테스팅 순서

1.telegram 모듈을 통하여 bot,Updater,CommandHandler,MessageHandler,Dispatcher객체를 생성해줍니다. -> 생성에 실패시 모듈에서 오류를 전달해주기 때문에 이부분에 대한 테스트는 생략

2.Main함수에서 poling 메서드를 통하여 봇의 데이터를 계속 받아옵니다.

3.Start, echo 함수를 통하여 메세지 처리 함수 구현 /start 입력에 대한 처리 함수와 메세지 입력시 이를 터미널에 출력한 후 bot_Send_Test.py 로 매개변수를 전달하며 실행

```python
import telegram
import argparse

def send_message(token, chat_id, message):
    bot = telegram.Bot(token=token)
    bot.sendMessage(chat_id=chat_id, text=message)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Send Telegram Message'
    parser.add_argument('token', help='Telegram Bot Token')
    parser.add_argument('chat_id', help='Chat ID')
    parser.add_argument('message', help='Message to send')
    args = parser.parse_args()

    send_message(args.token, args.chat_id, args.message)
```

(bot_Receive_Test.py 로 부터 채팅 아이디와 메세지를 받아와 이를 그대로 유저에게 전송하는 테스트)

- 키움 api 단위테스팅 설계

```
def getCommList(strCode, Fid):
    commList = openApi.GetCommRealData(strCode, Fid)
    if commList == '':
        return None
    else:
        return commList

1 usage  new *
def test_getCommList():
    assert getCommList('039490', 10) != None
    assert getCommList('054210', 8) != None
    assert getCommList('021154', 10) != None

test_getCommList()
```

-BSTR GetCommRealData(LPCTSTR strCode, long nFid)

요청 시 정상적으로 종목리스트가 넘어와야 하고, 넘어오는 것을 확인했습니다.

-BSTR GetConditionNameList(), BOOL SendCondition(LPCTSTR strScrNo, LPCTSTR strConditionName, int nIndex, int nSearch)

요청 시 조건검색 조건명 리스트를 받아오고, 종목조회 TR송신이 완료되어야 합니다.

-전체 테스트 시나리오

1.Kiwoomi 봇을 통해 메세지를 받아옵니다.

2.메세지를 받은 후 메세지 내용과 함께 Controller.py파일을 호출합니다.

3.Controller.py는 메시지를 분석하여 명령어를 구분합니다.

4.명령어가 /info일 경우, Controller.py는 키움 오픈API를 이용하여 해당 주식 정보를 조회합니다.
조회한 정보를 메시지로 만들어 Kiwoomi 봇에게 전송합니다.

5.명령어가 /news일 경우, Controller.py는 셀레니움을 이용하여 뉴스를 크롤링합니다.
조회한 뉴스를 메시지로 만들어 Kiwoomi 봇에게 전송합니다.

6.명령어가 /search일 경우, Controller.py는 키움 오픈API를 이용하여 검색 결과를 조회합니다.
조회한 검색 결과를 메시지로 만들어 Kiwoomi 봇에게 전송합니다.

7.명령어가 /alarm일 경우, Controller.py는 키움 오픈API를 이용하여 알람을 설정합니다.
알람 설정 결과를 메시지로 만들어 Kiwoomi 봇에게 전송합니다.

8.Control.py에서 데이터가 잘 넘어 올 경우 이를 사용자에게 전송합니다.

```python
def receive_message_Test():
    # Kiwoomi 봇으로부터 메시지를 받아오는 코드
    return

def analyze_message_Test(message):
    # Controller.py를 호출하여 메시지를 분석하는 코드
    return

def get_stock_info_Test(stock_code):
    # 키움 오픈API를 이용하여 해당 주식 정보를 조회하는 코드
    return

def get_news_Test():
    # 셀레니움을 이용하여 뉴스를 크롤링하는 코드
    return

def search_data_Test(keyword):
    # 키움 오픈API를 이용하여 검색 결과를 조회하는 코드
    return

def set_alarm_Test(stock_code, price):
    # 키움 오픈API를 이용하여 알람을 설정하는 코드
    return

def send_message_Test(message, chat_id):
    # Kiwoomi 봇으로 메시지를 전송하는 코드
    return

def test_scenario_Test():
    # 위에서 작성한 함수들을 조합하여 전체 테스트 시나리오를 실행하는 코드
    return
```