# 중간 보고서

## 오픈소스 기여

32173154 이승현

32172086 석홍준

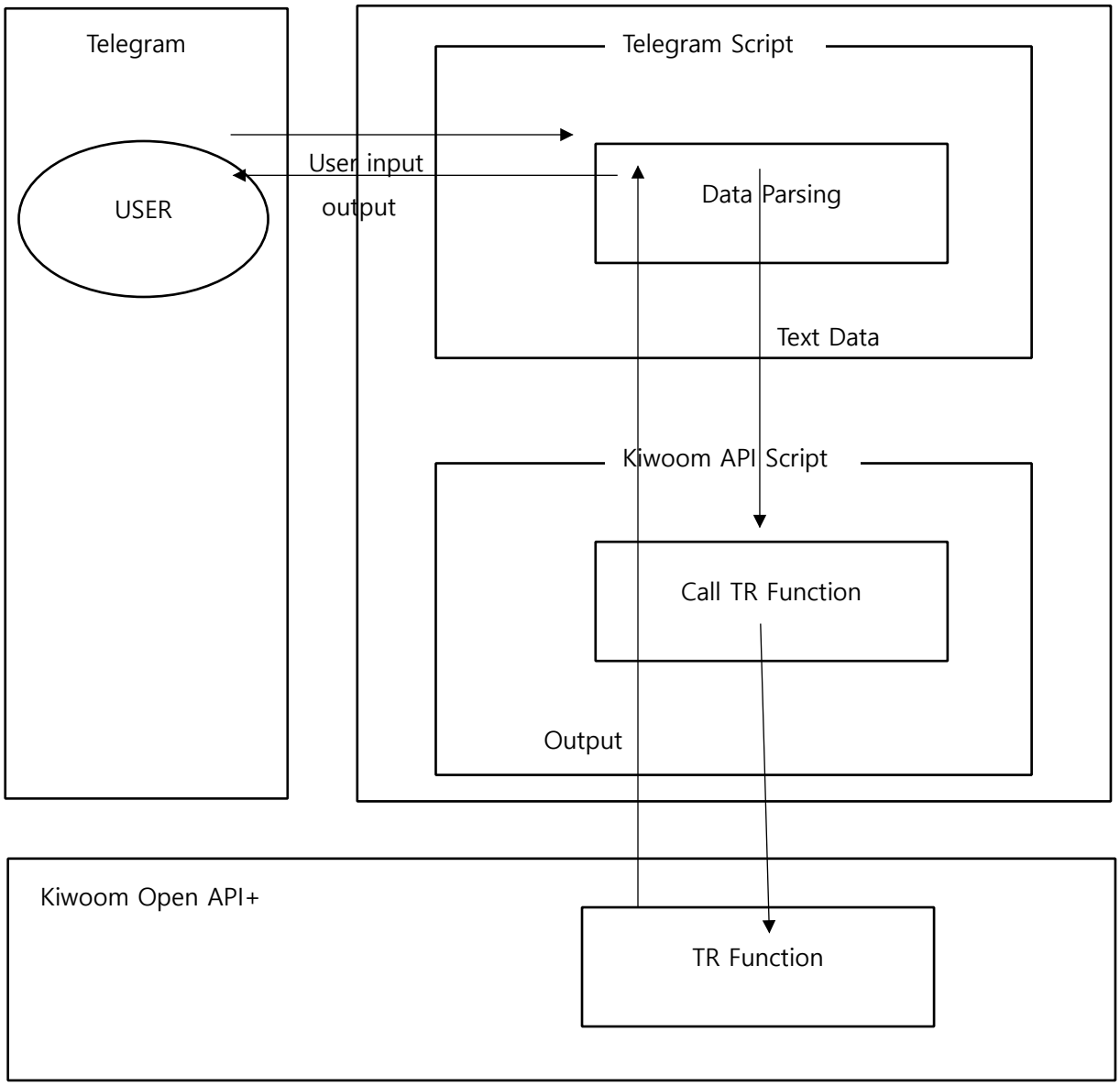
## 프로젝트 개요

키움 OPEN API+와 Telegram ChatBot을 사용한 주식 정보 제공 챗봇 제작

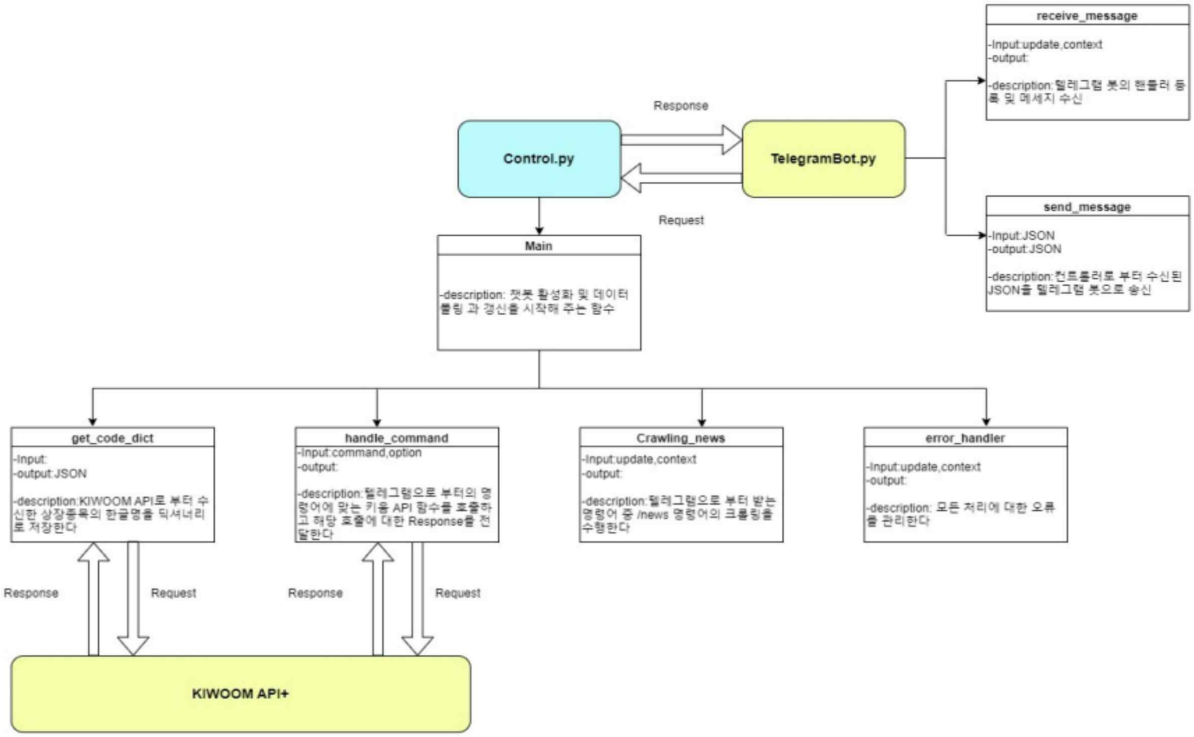
## 프로젝트 목표

주식을 사용하는 사용자들이 좀 더 쉽고, 편리하게 관련 정보를 얻을 수 있게 해주는 챗봇 제작

## 프로젝트 시스템 아키텍처



# 모듈 관계도


receive_message
-Input:update,context
-output:
-description:텔레그램 봇의 핸들러 등록 및 메세지 수신
Response
Control.py
TelegramBot.py
Request
send_message
-Input:JSON
-output:JSON
-description:컨트롤러로 부터 수신된 JSON을 텔레그램 봇으로 송신
Main
-description: 챗봇 활성화 및 데이터 폴링 과 경신을 시작해 주는 함수
get_code_dict
-Input:
-output:JSON
-description:KIWOOM API로 부터 수신한 상장종목의 한글명을 딕셔너리로 저장한다
handle_command
-Input:command,option
-output:
-description:텔레그램으로 부터의 명령어에 맞는 키움 API 함수를 호출하고 해당 호출에 대한 Response를 전달한다
Crawling_news
-Input:update,context
-output:
-description:텔레그램으로 부터 받는 명령어 중 /news 명령어의 크롤링을 수행한다
error_handler
-Input:update,context
-output:
-description: 모든 처리에 대한 오류를 관리한다
Response
Request
Response
Request
KiWOOM API+

# 사용 모듈함수

### GetCommRealData

| 함수 원형 | BSTR GetCommRealData(LPCTSTR strCode, long nFid) |
|---|---|
| 설명 | 실시간 시세 데이터 반환 |
| 입력값 | strCode – 종목 코드<br>nFid – 실시간 아이템 |
| 반환값 | 수신 데이터 |
| 비고 | 현재가 출력 – openApi.GetCommRealData("039490", 10)<br>strCode는 OnReceiveRealData 첫번째 매개변수를 사용 |

### OnReceiveRealData

| 함수 원형 | void OnReceiveRealData(LPCTSTR sJongmokCode, LPCTSTR sRealType, LPCTSTR sRealData) |
|---|---|
| 설명 | 실시간데이터를 받은 시점을 반환 |
| 입력값 | sJongmokCode – 종목코드 sRealType – 리얼타입 sRealData – 실시간 데이터전문 |
| 반환값 | X |
| 비고 | X |

### GetConditionNameList

| 함수 원형 | BSTR GetConditionNameList() |
|---|---|
| 설명 | 조건검색 조건명 리스트를 받아옴 |
| 입력값 | |
| 반환값 | 조건명 리스트(인덱스^조건명) |
| 비고 | 조건명 리스트를 구분(';' 사용)하여 받아온다.<br>Ex) 인덱스1^조건명;인덱스2^조건명;인덱스3^조건명... |

### SendCondition

| 함수 원형 | BOOL SendCondition(LPCTSTR strScrNo, LPCTSTR strConditionName, int nIndex, int nSearch) |
|---|---|
| 설명 | 조건검색 종목조회TR송신 |
| 입력값 | LPCTSTR strScrNo : 화면번호<br>LPCTSTR strConditionName : 조건명<br>int nIndex : 조건명인덱스 |

| | |
|---|---|
| | int nSearch : 조회구분(0:일반조회, 1:실시간조회, 2:연속조회) |
| 반환값 | 성공 1, 실패 0 |
| 비고 | 단순 조건식에 맞는 종목을 조회하기 위해서는 조회구분을 0으로 하고, 실시간 조건검색을 하기 위해서는 조회구분을 1로 한다. OnReceiveTrCondition으로 결과값이 온다. 연속조회가 필요한 경우에는 응답받는 곳에서 연속조회 여부에 따라 연속조회를 송신하면된다. |

### OnReceiveTrCondition

| | |
|---|---|
| 함수 원형 | void OnReceiveTrCondition(LPCTSTR sScrNo, LPCTSTR strCodeList, LPCTSTR strConditionName, int nIndex, int nNext) |
| 설명 | 조건검색 조회응답으로 종목리스트를 구분자(";")로 붙어서 받는 시점 |
| 입력값 | LPCTSTR sScrNo : 종목코드<br>LPCTSTR strCodeList : 종목리스트(";"로 구분)<br>LPCTSTR strConditionName : 조건명<br>int nIndex : 조건명 인덱스<br>int nNext : 연속조회(2:연속조회, 0:연속조회없음) |
| 반환값 | X |
| 비고 | X |

### SendCondition

| | |
|---|---|
| 함수 원형 | BOOL SendCondition(LPCTSTR strScrNo, LPCTSTR strConditionName, int nIndex, int nSearch) |
| 설명 | 조건검색 종목조회TR송신 |
| 입력값 | LPCTSTR strScrNo : 화면번호<br>LPCTSTR strConditionName : 조건명<br>int nIndex : 조건명인덱스<br>int nSearch : 조회구분(0:일반조회, 1:실시간조회, 2:연속조회) |
| 반환값 | 성공 1, 실패 0 |
| 비고 | 단순 조건식에 맞는 종목을 조회하기 위해서는 조회구분을 0으로 하고, 실시간 조건검색을 하기 위해서는 조회구분을 1로 한다. OnReceiveTrCondition으로 결과값이 온다. 연속조회가 필요한 경우에는 응답받는 곳에서 연속조회 여부에 따라 연속조회를 송신하면 된다. |

### SendConditionStop

| | |
|---|---|
| 함수 원형 | Void SendConditionStop(LPCTSTR strScrNo, LPCTSTR strConditionName, int nIndex) |
| 설명 | 조건검색 실시간 중지TR을 송신 |
| 입력값 | LPCTSTR strScrNo : 화면번호<br>LPCTSTR strConditionName : 조건명<br>int nIndex : 조건명인덱스 |
| 반환값 | X |
| 비고 | 해당 조건명의 실시간 조건검색을 중지하거나, 다른 조건명으로 바꿀 때 이전 조건명으로 실시간 조건검색을 반드시 중지해야한다. 화면 종료시에도 실시간 조건검색을 한 조건명으로 전부 중지해줘야 한다. |

## 테스팅

- 단위테스팅

키움 API 별 테스팅

<코스피 종목 주식 코드 가져오기>

```
from pykiwoom.kiwoom import *

kiwoom = Kiwoom()
kiwoom.CommConnect(block=True)

kospi = kiwoom.GetCodeListByMarket('0')

print(kospi)
```

<주식기본정보요청>

```
from pykiwoom.kiwoom import *

kiwoom = Kiwoom()
kiwoom.CommConnect(block=True)

df = kiwoom.block_request("opt10001",
                          종목코드="005930",
                          output="주식기본정보",
                          next=0)

print(df)
```

<실시간 주식 데이터 요청>

```
from pykiwoom.kiwoom import *

kiwoom = Kiwoom()
kiwoom.CommConnect(block=True)

df = kiwoom.GetCommRealData("005930",10)

print(df)
```

<조건검색 TR>

```python
from pykiwoom.kiwoom import *

kiwoom = Kiwoom()
kiwoom.CommConnect(block=True)

1 usage  new *
def condition_search(cond_name, cond_index):
    kiwoom.SendCondition("0101", cond_name, cond_index, 0)
    while kiwoom.received_cond == False:
        app.processEvents()
    codes = kiwoom.GetConditionNameList()
    return codes

cond_name = "종목조건식"
cond_index = 0
codes = condition_search(cond_name, cond_index)
print(codes)
```

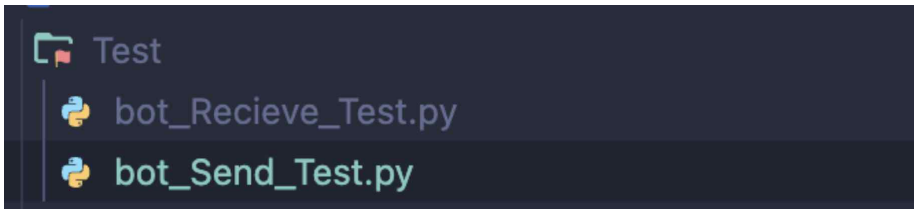
<조건검색 리스트 받아오기>

```python
from pykiwoom.kiwoom import *

kiwoom = Kiwoom()
kiwoom.CommConnect(block=True)

new *
def get_condition_codes():
    codes = []
    condition_list = kiwoom.GetConditionNameList()
    for condition in condition_list:
        if condition != "":
            condition_index = int(condition.split("^")[0])
            condition_name = condition.split("^")[1]
            code_list = kiwoom.SendCondition("0101", condition_name, condition_index, 1)
            for code in code_list:
                codes.append(code)
    return codes
```

**<챗봇 단위테스팅>**




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

```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()
```

<챗봇 통신 단위테스트>

```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)
```

-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 Kiyomi 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
```

## 개발 진행 내용

봇을 통한 메세지 수신 및 명령어에 따른 Controller 함수 호출

```
# 메시지를 받아들이고 명령어 처리 함수를 연결하는 Updater 객체 생성
updater = Updater(TOKEN, use_context=True)
dispatcher = updater.dispatcher
dispatcher.add_handler(CommandHandler('start', start))
dispatcher.add_handler(CommandHandler('help', help))
dispatcher.add_handler(CommandHandler('info', info))
dispatcher.add_handler(CommandHandler('news', news))
dispatcher.add_handler(CommandHandler('alarm', alarm))
dispatcher.add_handler(CommandHandler('search', search))

# 텔레그램 봇 시작
updater.start_polling()
```

/start- 봇과의 채팅을 시작하는 명령어

/help-봇의 사용법에 대한 설명

/info-주식명에 해당하는 정보를 가져오는 명령어

/news-주식명에 해당하는 뉴스를 크롤링하는 명령어

/alarm-주식명이 정해진 가격에 도달하였을 때 알람을 해주는 기능

/search- 주식명의 일봉,월봉,년봉 등 특정 정보를 가져오는 명령어

```python
# /start 명령어 처리 함수
def start(update, context):
    chat_id = update.effective_chat.id
    context.bot.send_message(chat_id=chat_id, text='안녕하세요! 무엇을 도와드릴까요?')
#/help 명령어 처리 함수
def help(update, context):
    chat_id = update.effective_chat.id
    context.bot.send_message(chat_id=chat_id, text='키우미 봇은 간단한 주식 도움 봇이에요 \n \
                             /info [주식명]= 주식의 현재 시세 및 관련 정보를 불러옵니다.\n \
                             /search [주식명][옵션]= 주식의 일봉,월봉,연봉 등 특정 정보를 불러옵니다.\n \
                             /alarm [주식명][가격]=주식이 특정 가격에 도달하였을 경우 봇을 통해 알려줍니다.\n \
                             /news [주식명]=해당 주식에 관련된 기사를 모으고 헤드라인 기사를 띄워줍니다. ')

# /info 명령어 처리 함수
def info(update, context):
    chat_id = update.effective_chat.id
    messages.append( context.args[0])
    # 메시지를 배열에 저장
    # Controller.py의 함수 호출
    Controller.bring_info(messages)

# /news 명령어 처리 함수
def news(update, context):
    chat_id = update.effective_chat.id
    # 메시지를 배열에 저장
    messages.append(context.args[0])
    # Controller.py의 함수 호출
    Controller.bring_News(messages)

# /alarm 명령어 처리 함수
def alarm(update, context):
```

각 명령어에 맞는 Controller 함수 호출 처리

뉴스 크롤링

```python
import requests
from bs4 import BeautifulSoup

url = 'https://finance.naver.com/'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

news_articles = []

for article in soup.find_all('article'):
    headline = article.find('h3', class_='story-title').text.strip()
    summary = article.find('p', class_='summary').text.strip()
    news_articles.append({'headline': headline, 'summary': summary})

print(news_articles)
```

주식 기본정보 (/info,/search) 구현

```python
class Kiwoom(QAxWidget):
    new *
    def __init__(self):
        super().__init__()
        self.setControl("KHOPENAPI.KHOpenAPICtrl.1")
        self.CommConnect()

        self.OnEventConnect.connect(self.event_connect)

    1 usage  new *
    def CommConnect(self):
        self.dynamicCall("CommConnect()")

    new *
    def GetCodeListByMarket(self, market):
        code_list = self.dynamicCall("GetCodeListByMarket(QString)", market)
        code_list = code_list.split(';')
        return code_list[:-1]

    1 usage  new *
    def GetMasterCodeName(self, code):
        code_name = self.dynamicCall("GetMasterCodeName(QString)", code)
        return code_name

    new *
    def GetCommRealData(self, code, fid):
        real_data = self.dynamicCall("GetCommRealData(QString, int)", code, fid)
        return real_data

    1 usage  new *
    def event_connect(self, err_code):
        if err_code == 0:
            print("로그인 성공")

        else:
            print("로그인 실패")
```

```python
app = QApplication(sys.argv)
kiwoom = Kiwoom()

code = "005930"
code_name = kiwoom.GetMasterCodeName(code)
print("종목 이름:", code_name)

app.exec_()
```

## 개발 예정 내용

Bot_Sender: 얻어온 정보의 데이터의 형식을 메세지형태로 전환하여 전송하는 스크립트.

Kiwom_alarm:지정가에 도달시 알람 스크립트.

Controller: 키움 api와 봇의 매개체 스크립트.