클라우드기반 데이터 보안 전문가 양성 과정

모듈 프로젝트 3

2021년 3월 19일

이름: 박진환

■ 프로젝트 분배

1. 프로젝트 참가자
    A. 김승미
    B. 박진환
    C. 신선아
    D. 이희석
2. 총괄 책임 → 박진환
3. 전체 프로젝트 → 박진환, 이희석, 신선아, 김승미
    A. 주제 1 / 날씨 → 신선아, 김승미
        i. 데이터 수집 → 신선아, 김승미
            1. Scrapy 작성 → 신선아, 김승미
            2. Pipeline 구축 → 박진환, 신선아, 김승미
            3. RDS 구축 → 박진환
        ii. 기능 → 박진환, 신선아, 김승미
            1. 5개의 도시의 일-시간 별 온도 비교 그래프 출력 → 박진환, 신선아, 김승미
            2. 특정 도시의 일-시간 별 온도와 습도 그래프 출력 → 박진환, 신선아, 김승미
            3. 5개의 도시의 여러 정보를 JSON 형식으로 출력 → 박진환, 신선아, 김승미
    B. 주제 2 / 암호화폐(주식) → 박진환, 이희석
        i. 데이터 수집 → 박진환, 이희석
            1. Scrapy 작성 → 이희석
            2. Pipeline 구축 → 박진환, 이희석

3. RDS 구축 → 박진환

ii. 기능 → 박진환, 이희석

1. 특정 암호화폐의 일-시간 별 가격 변동 그래프 출력 → 박진환, 이희석
2. 3개의 암호화폐의 변동률 그래프 출력 → 박진환, 이희석
3. 3개의 암호화폐의 여러 정보를 JSON 형식으로 출력 → 박진환, 이희석

4. Python Django RestfulAPI 설계 및 구현 → 이희석, 신선아, 김승미

A. 서버 구축 및 API 설계 → 이희석, 신선아, 김승미

B. 기능 구현을 위한 코드 작성 → 박진환, 이희석

5. 마무리 검수 → 박진환, 이희석

■ 모듈 프로젝트 내용

<table>
<tr><td>주제</td><td>1. 날씨
2. 암호화폐(주식)</td></tr>
<tr><td>기능 – 날씨</td><td>1. 5개의 도시의 일-시간 별 온도 비교 그래프 출력
2. 특정 도시의 일-시간 별 온도와 습도 그래프 출력
3. 5개의 도시의 여러 정보를 JSON 형식으로 출력</td></tr>
<tr><td>기능 – 주식</td><td>1. 특정 암호화폐의 일-시간 별 가격 변동 그래프 출력
2. 3개의 암호화폐의 변동률 그래프 출력
3. 3개의 암호화폐의 여러 정보를 JSON 형식으로 출력</td></tr>
<tr><td>설계 – 날씨</td><td>1. Scrapy와 apscheduler를 활용하여 날씨 사이트에서 원하는 정보를 5분 간격으로 crawling</td></tr>
</table>

```python
import scrapy
from weatherbot.items import WeatherbotItem


class WeatherbotsSpider(scrapy.Spider):
    name = 'weatherbot'
    custom_settings = {'JOBDIR': 'crawl_weatherbots1'}
    allowed_domains = ['yahoo.com/news/weather/south-korea/seoul/seoul-1132599/','yahoo.com/news/weather/united-states/new-york/new-york-2459115','yahoo.co
    start_urls = ['https://www.yahoo.com/news/weather/south-korea/seoul/seoul-1132599/', 'https://www.yahoo.com/news/weather/united-states/new-york/new-yor

    def parse(self, response):
        temper = response.xpath('//*[@id="Lead-1-WeatherLocationAndTemperature"]/div/section[2]/div/div[3]/span[1]/text()').extract()
        humid = response.xpath('//*//*[@id="weather-detail"]/div/div[1]/div/div[2]/ul/li[2]/div[2]/text()').extract()
        high_temp = response.xpath('//*//*[@id="Lead-1-WeatherLocationAndTemperature"]/div/section[2]/div/div[2]/span[1]/text()').extract()
        low_temp = response.xpath('//*//*[@id="Lead-1-WeatherLocationAndTemperature"]/div/section[2]/div/div[2]/span[2]/text()').extract()
        title = response.xpath('//*[@id="Lead-1-WeatherLocationAndTemperature"]/div/section[1]/div[1]/div/h1/text()').extract()
        wind = response.xpath('//*[@id="weather-wind"]/div/div[1]/div[2]/div/p/span/text()').extract()
        weather = response.xpath('//*[@id="Lead-1-WeatherLocationAndTemperature"]/div/section[2]/div/div[1]/span[2]/text()').extract()

        for row in zip(temper,humid,high_temp,low_temp,title,wind,weather):
            item = WeatherbotItem()
            item['temper'] = row[0]
            item['humid'] = row[1]
            item['high_temp'] = row[2]
            item['low_temp'] = row[3]
            item['title'] = row[4]
            item['wind'] = row[5]
            item['weather'] = row[6]

            yield item
```

WeatherbotsSpider › parse() › for row in zip(temper,humid,hig...

```python
from scrapy.crawler import CrawlerProcess
from scrapy.utils.project import get_project_settings
from apscheduler.schedulers.twisted import TwistedScheduler
from weatherbot.spiders.weatherbot import WeatherbotsSpider

process = CrawlerProcess(get_project_settings())
scheduler = TwistedScheduler()
scheduler.add_job(process.crawl, 'cron', args=[WeatherbotsSpider], minute='0,5,10,15,20,25,30,35,40,45,50,55')
scheduler.start()
process.start(False)
```

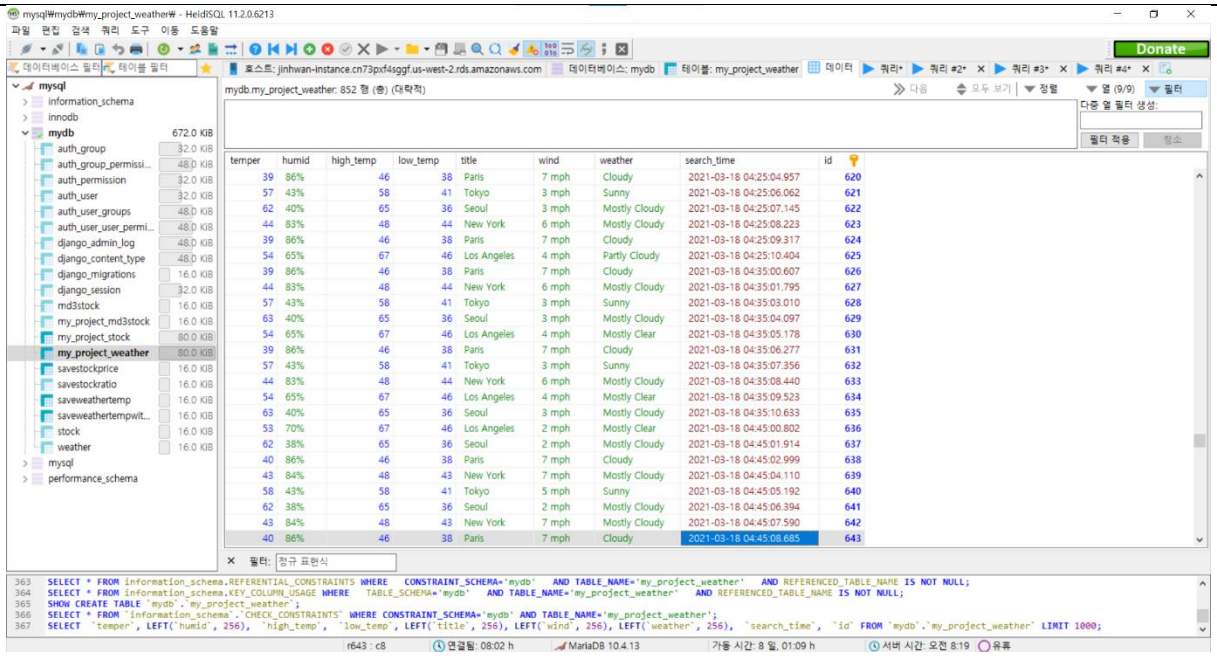
2. Scrapy의 Pipeline을 활용하여 Scrapy, Kafka, RDS를 연결

```python
from kafka import KafkaProducer
from json import dumps
import time


class WeatherbotPipeline:
    def __init__(self):
        self.producer = KafkaProducer(acks=0,
                                       compression_type='gzip',
                                       bootstrap_servers=['localhost:9092'],
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))

    def process_item(self, item, spider):
        item = dict(item)
        item["temper"] = int(item["temper"])
        item["high_temp"] = int(item["high_temp"])
        item["low_temp"] = int(item["low_temp"])
        data = {"schema": {"type": "struct", "fields": [{"type": "int32", "optional": "false", "field": "temper"},{"type": "string", "optional": "false", "f
        self.producer.send("my_project_weather", data)
        self.producer.flush()

```

3. 스크래피를 통한 RDS 데이터 삽입을 확인(사진은 스크래피, 토픽 전달 확인, RDS 데이터 사진)

```
1 주식스크래피  2 날씨스크래피  3 잡다한거
  File "/usr/local/lib/python3.7/site-packages/itemadapter/adapter.py", line 189, in __init__
    raise TypeError(f"No adapter found for objects of type: {type(item)} ({item})")
TypeError: No adapter found for objects of type: <class 'NoneType'> (None)
2021-03-18 08:05:11 [scrapy.core.engine] INFO: Closing spider (finished)
2021-03-18 08:05:11 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 1319,
 'downloader/request_count': 5,
 'downloader/request_method_count/GET': 5,
 'downloader/response_bytes': 333896,
 'downloader/response_count': 5,
 'downloader/response_status_count/200': 5,
 'elapsed_time_seconds': 11.280626,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2021, 3, 18, 8, 5, 11, 656080),
 'item_scraped_count': 5,
 'log_count/DEBUG': 259,
 'log_count/ERROR': 10,
 'log_count/INFO': 26,
 'memusage/max': 128086016,
 'memusage/startup': 128086016,
 'response_received_count': 5,
 'scheduler/dequeued': 5,
 'scheduler/dequeued/disk': 5,
 'scheduler/enqueued': 5,
 'scheduler/enqueued/disk': 5,
 'start_time': datetime.datetime(2021, 3, 18, 8, 5, 0, 375454)}
2021-03-18 08:05:11 [scrapy.core.engine] INFO: Spider closed (finished)
```

```
1 주식스크래피  2 날씨스크래피  3 잡다한거
h_temp": 65, "low_temp": 36, "title": "Seoul", "wind": "2 mph", "weather": "Mostly Cloudy", "search_time": 1616043900686}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": "false", "field": "temper"}, {"type": "string", "optional": "false", "field": "humid"}, {"type": "
int32", "optional": "false", "field": "high_temp"}, {"type": "int32", "optional": "false", "field": "low_temp"}, {"type": "string", "optional": "false", "field": "title
}, {"type": "string", "optional": "false", "field": "wind"}, {"type": "string", "optional": "false", "field": "weather"}, {"type": "int64", "optional": "true", "name": "
org.apache.kafka.connect.data.Timestamp", "version": 1, "field": "search_time"}], "optional": "false", "name": "weather"}, "payload": {"temper": 58, "humid": "43%", "hig
h_temp": 58, "low_temp": 41, "title": "Tokyo", "wind": "5 mph", "weather": "Sunny", "search_time": 1616043901883}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": "false", "field": "temper"}, {"type": "string", "optional": "false", "field": "humid"}, {"type": "
int32", "optional": "false", "field": "high_temp"}, {"type": "int32", "optional": "false", "field": "low_temp"}, {"type": "string", "optional": "false", "field": "title
}, {"type": "string", "optional": "false", "field": "wind"}, {"type": "string", "optional": "false", "field": "weather"}, {"type": "int64", "optional": "true", "name": "
org.apache.kafka.connect.data.Timestamp", "version": 1, "field": "search_time"}], "optional": "false", "name": "weather"}, "payload": {"temper": 40, "humid": "86%", "hig
h_temp": 46, "low_temp": 38, "title": "Paris", "wind": "7 mph", "weather": "Cloudy", "search_time": 1616043902976}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": "false", "field": "temper"}, {"type": "string", "optional": "false", "field": "humid"}, {"type": "
int32", "optional": "false", "field": "high_temp"}, {"type": "int32", "optional": "false", "field": "low_temp"}, {"type": "string", "optional": "false", "field": "title
}, {"type": "string", "optional": "false", "field": "wind"}, {"type": "string", "optional": "false", "field": "weather"}, {"type": "int64", "optional": "true", "name": "
org.apache.kafka.connect.data.Timestamp", "version": 1, "field": "search_time"}], "optional": "false", "name": "weather"}, "payload": {"temper": 43, "humid": "84%", "hig
h_temp": 48, "low_temp": 43, "title": "New York", "wind": "7 mph", "weather": "Mostly Cloudy", "search_time": 1616043904061}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": "false", "field": "temper"}, {"type": "string", "optional": "false", "field": "humid"}, {"type": "
int32", "optional": "false", "field": "high_temp"}, {"type": "int32", "optional": "false", "field": "low_temp"}, {"type": "string", "optional": "false", "field": "title
}, {"type": "string", "optional": "false", "field": "wind"}, {"type": "string", "optional": "false", "field": "weather"}, {"type": "int64", "optional": "true", "name": "
org.apache.kafka.connect.data.Timestamp", "version": 1, "field": "search_time"}], "optional": "false", "name": "weather"}, "payload": {"temper": 53, "humid": "70%", "hig
h_temp": 67, "low_temp": 46, "title": "Los Angeles", "wind": "2 mph", "weather": "Partly Cloudy", "search_time": 1616043905151}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": "false", "field": "temper"}, {"type": "string", "optional": "false", "field": "humid"}, {"type": "
int32", "optional": "false", "field": "high_temp"}, {"type": "int32", "optional": "false", "field": "low_temp"}, {"type": "string", "optional": "false", "field": "title
}, {"type": "string", "optional": "false", "field": "wind"}, {"type": "string", "optional": "false", "field": "weather"}, {"type": "int64", "optional": "true", "name": "
org.apache.kafka.connect.data.Timestamp", "version": 1, "field": "search_time"}], "optional": "false", "name": "weather"}, "payload": {"temper": 53, "humid": "70%", "hig
h_temp": 67, "low_temp": 46, "title": "Los Angeles", "wind": "2 mph", "weather": "Partly Cloudy", "search_time": 1616043906255}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "optional": "false", "field": "temper"}, {"type": "string", "optional": "false", "field": "humid"}, {"type": "
int32", "optional": "false", "field": "high_temp"}, {"type": "int32", "optional": "false", "field": "low_temp"}, {"type": "string", "optional
```



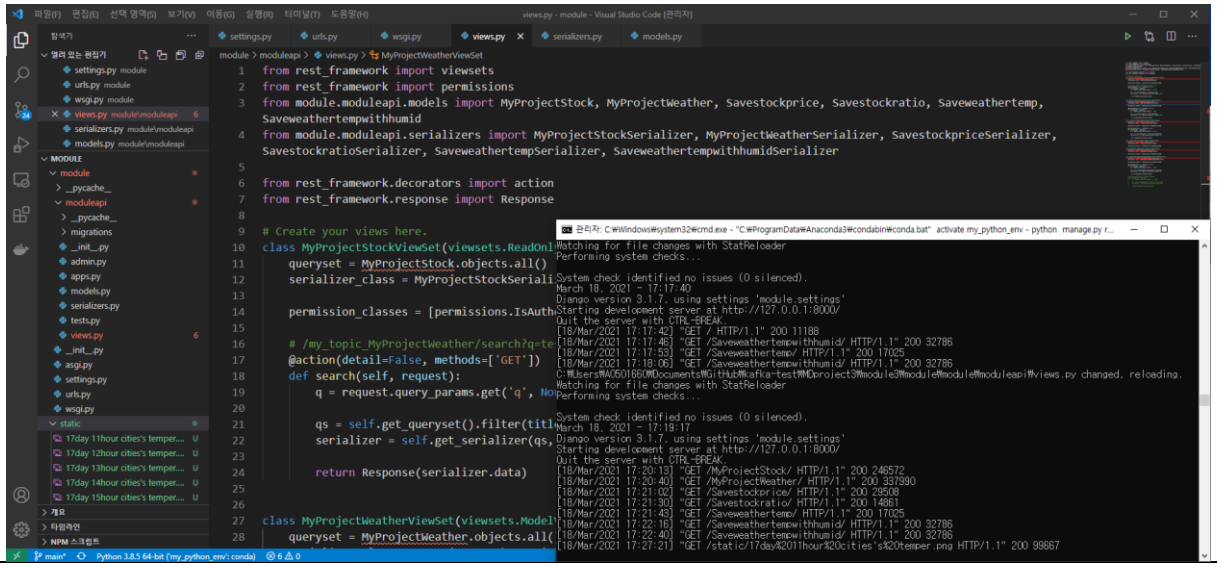
4. RDS에 저장된 데이터를 활용하여 시각화를 구현하고 파일로 저장(사진은 기능 1, 기능 2 순서)

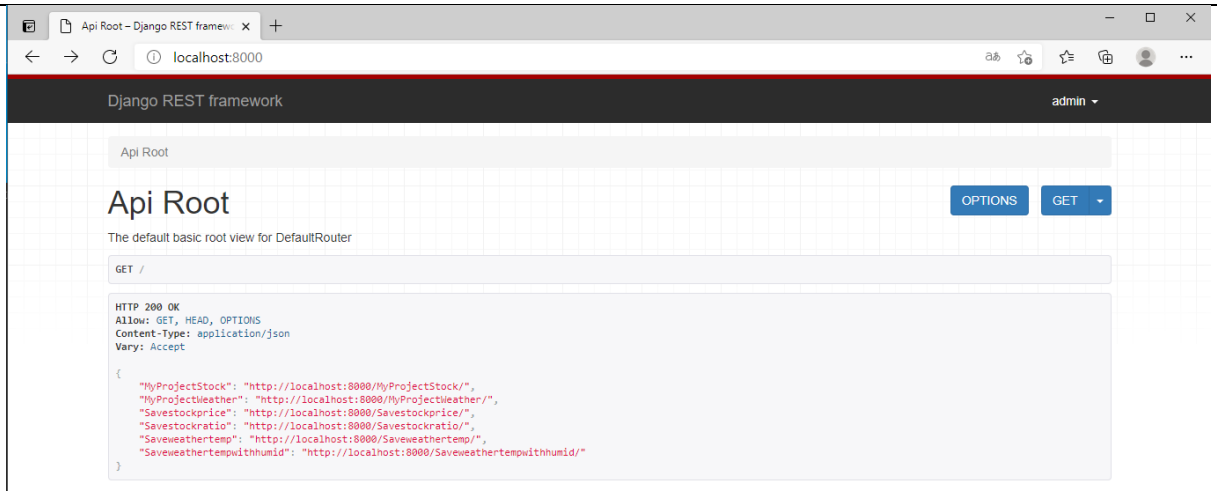
```
if len(Sdata) != 0:
    minute_test = set()
    for k in Sdata["time_minute"]:
        minute_test.add(k)
    minute_test = list(minute_test)
    minute_test.sort()

    fig = plt.figure()
    ax1 = fig.add_subplot(1, 1, 1)
    ax1.plot(Sdata["temper"], marker=r'o', color=u'blue', linestyle='-', label="Seoul")
    ax1.plot(Tdata["temper"], marker=r'o', color=u'red', linestyle='-', label="Tokyo")
    ax1.plot(Pdata["temper"], marker=r'o', color=u'green', linestyle='-', label="Paris")
    ax1.plot(Ndata["temper"], marker=r'o', color=u'yellow', linestyle='-', label="New York")
    ax1.plot(Ldata["temper"], marker=r'o', color=u'pink', linestyle='-', label="Los Angeles")
    plt.xticks(range(len(Sdata["temper"])), range(len(Sdata["temper"])), rotation=0,
               fontsize="large")
    ax1.xaxis.set_ticks_position('bottom')
    ax1.yaxis.set_ticks_position('left')
    ax1.set_title("{0}th {1}hour temper".format(j, i))
    plt.xlabel('data count')
    plt.ylabel('temper')
    plt.legend(loc='best')
    plt.savefig("{}day {}hour cities's temper.png".format(j, i), dpi=400, bbox_inches='tight')
    savepath = '/static/'+"{}day {}hour cities's temper.png".format(j, i)
    insertSql = 'INSERT INTO saveweathertemp(time_day, time_hour, savepath) VALUES(%s, %s, %s)'
    try:
        time.sleep(1)
        self.cursor.execute(insertSql, (j, i, savepath))
        print("데이터 추가 완료")
        self.connectdb.commit()
    except Exception as e:
        print(e)
```

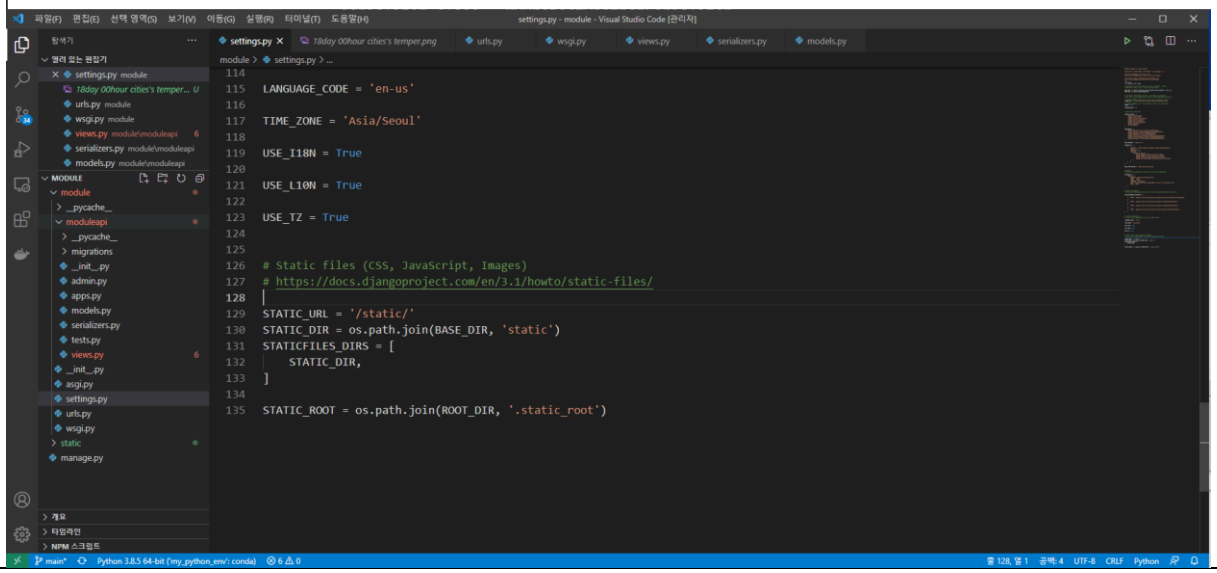
```
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('data count')
ax1.set_ylabel('temper', color=color)
ax1.plot(range(len(ndata["temper"])), ndata["temper"], marker=r'*', linestyle="-", color=color)
ax1.tick_params(axis='y', labelcolor=color)
plt.xticks(range(len(ndata["temper"])), range(len(ndata["temper"])), rotation=0,
           fontsize="large")
ax1.set_title("{} {}day {}hour temper with humid".format(i, j, k))

ax2 = ax1.twinx()
color = 'tab:blue'
ax2.set_ylabel('humid', color=color)
ax2.plot(range(len(ndata["temper"])), ndata["humid"], marker=r'o', linestyle='-', color=color)
ax2.set_ylim(bottom=0, top=100)
ax2.tick_params(axis='y', labelcolor=color)
fig.tight_layout()
plt.savefig("{}day {}hour cities's temwhum.png".format(j, k), dpi=400, bbox_inches='tight')
savepath = '/static/'+"{}day {}hour cities's temwhum.png".format(j, k)
insertSql = 'INSERT INTO saveweathertempwithhumid(title, time_day, time_hour, savepath) VALUES(%s, %s, %s, %s)'
try:
    time.sleep(1)
    self.cursor.execute(insertSql, (i, j, k, savepath))
    print("데이터 추가 완료")
    self.connectdb.commit()
except Exception as e:
    print(e)
plt.show()
```

Mdproject3 › by_minute_temper() › for j in days › for i in hours › if len(Sdata) != 0

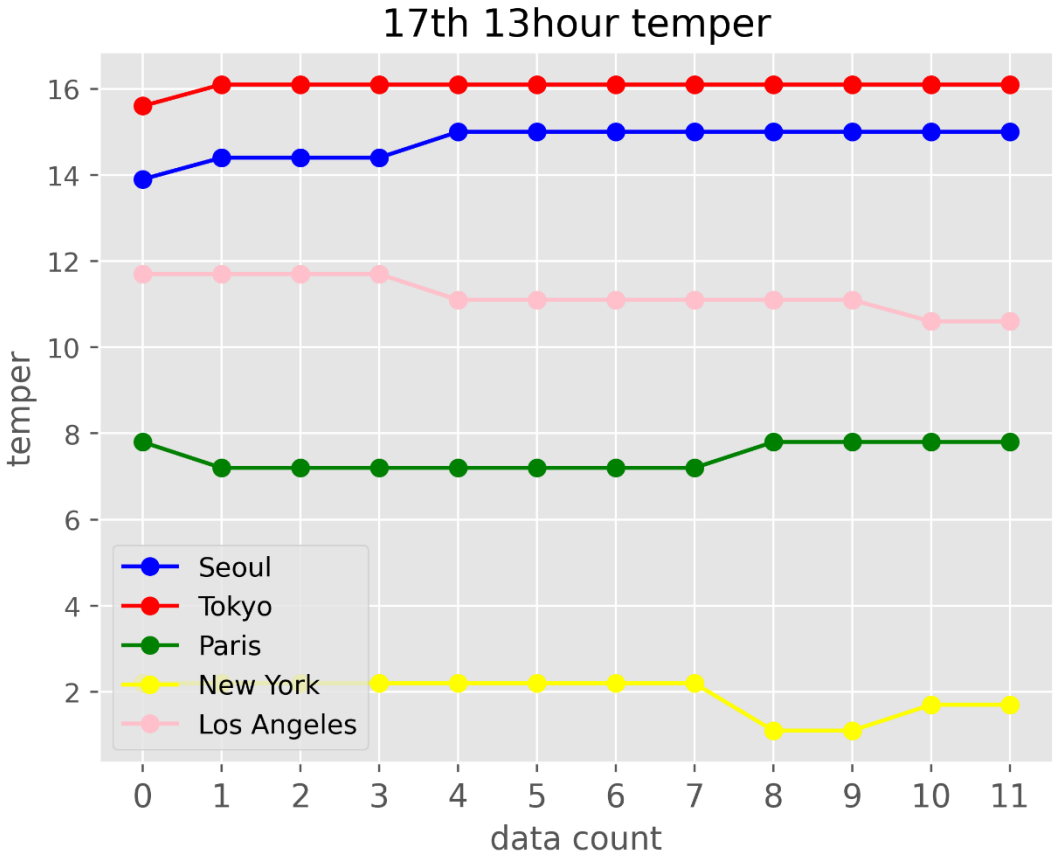
5. Python Django를 활용하여 RestAPI 구축하고 기능과의 연동 준비 완료(사진은 서버 구동, 메인 페이지, 사진을 저장한 Static 관련 코드)

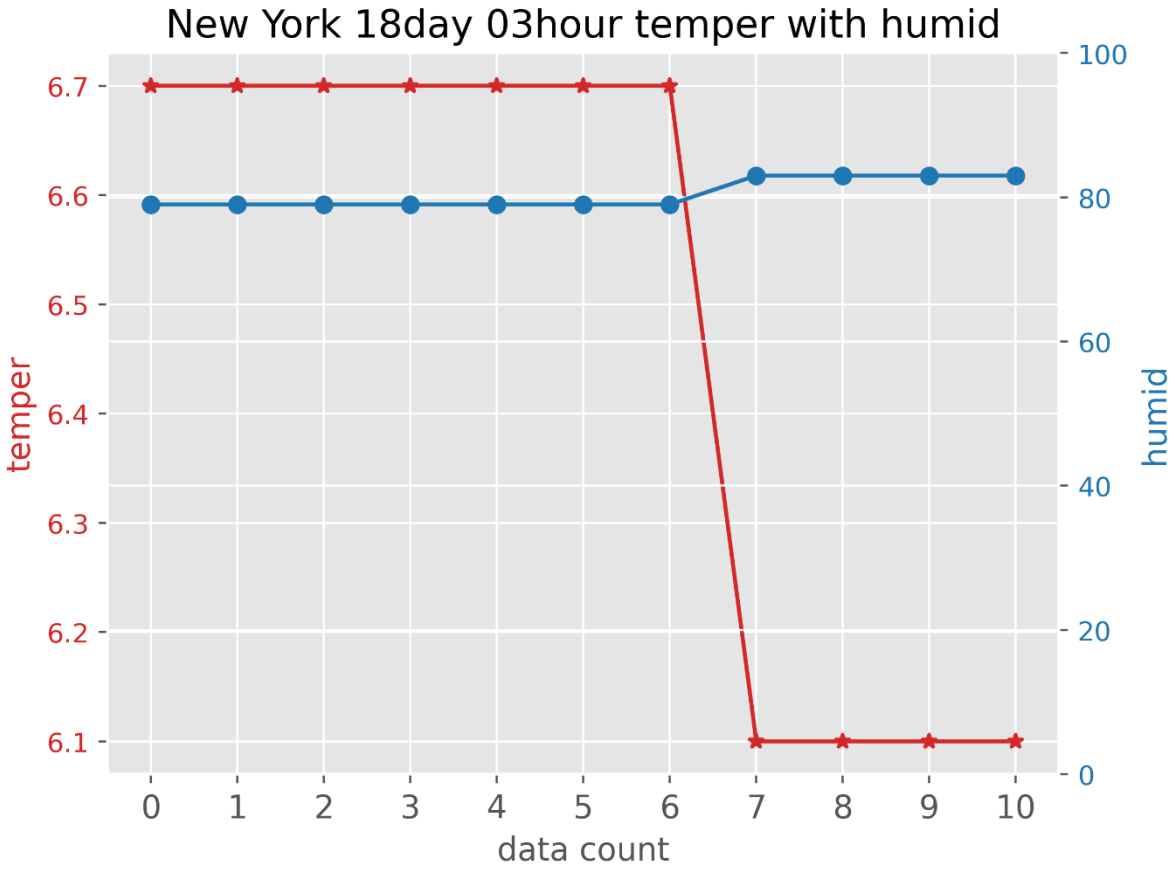






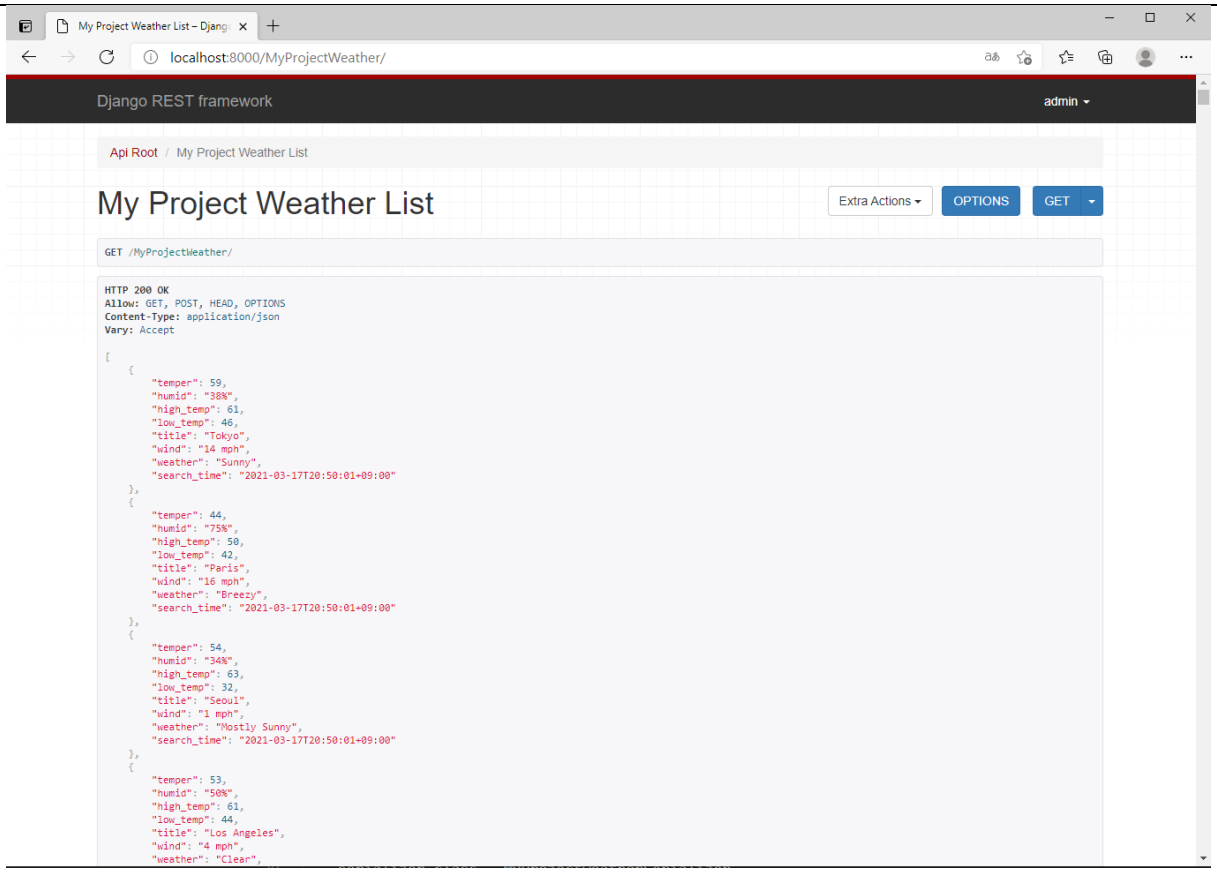
구현_날씨	1. Python의 matplotlib 모듈과 RDS의 데이터를 활용하여 그래프로 시각화 구현 (사진은 기능 1, 기능 2 순서)


17th 13hour temper
temper
data count
Seoul
Tokyo
Paris
New York
Los Angeles

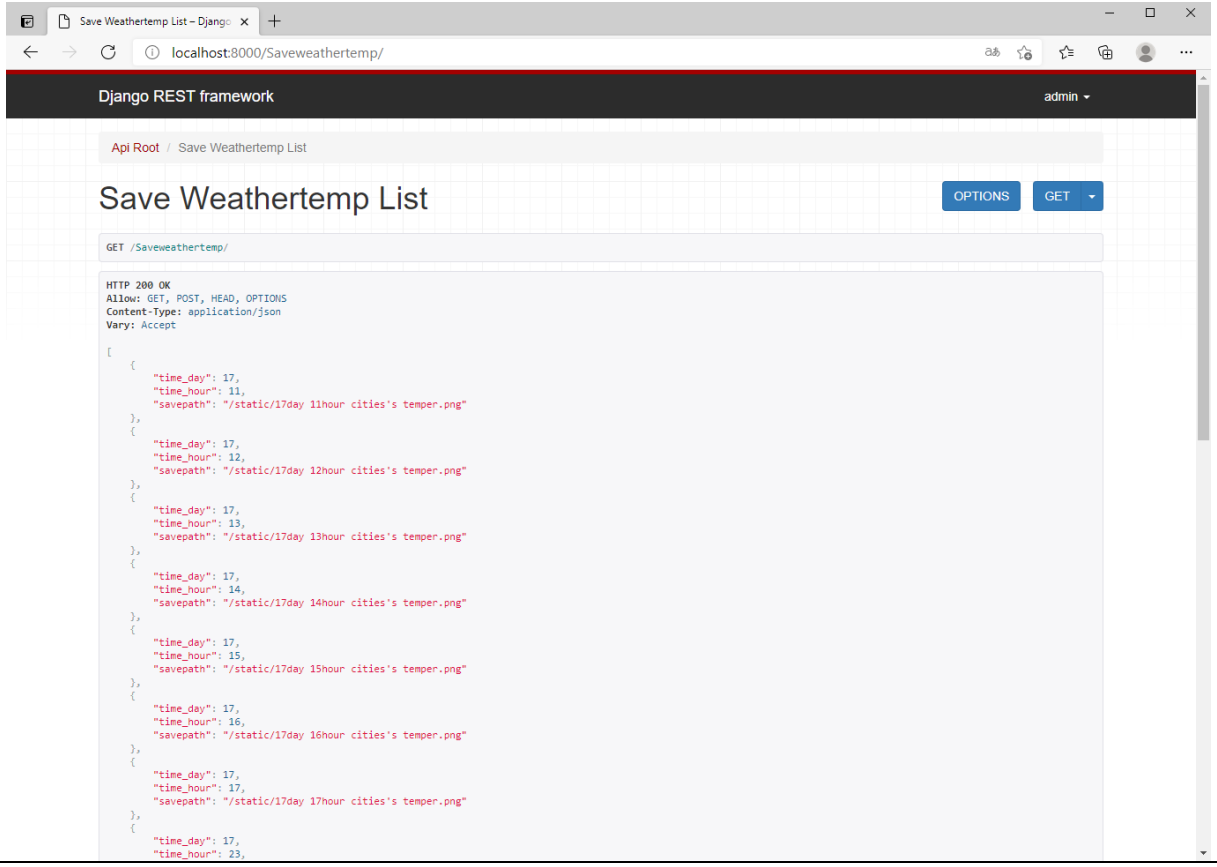

New York 18day 03hour temper with humid
temper
humid
data count

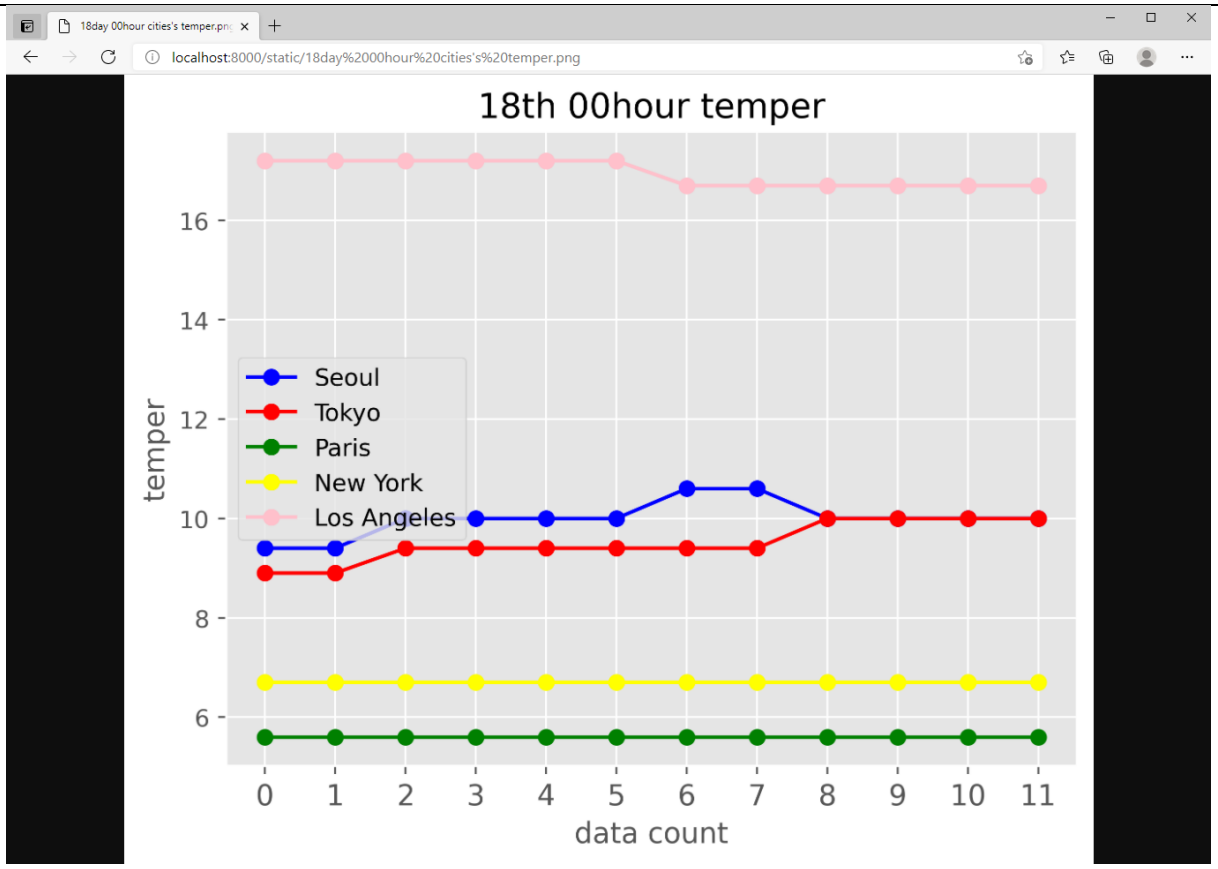
2. Python Django를 활용하여 RestfulAPI에 데이터 JSON파일과 경로 입력 시, 그래프 출력 기능 구현
3. 그래프 출력을 위한 사진이 모여 있는 Django의 Static 폴더

4. 기능 3에 해당하는 날씨 데이터 JSON 출력

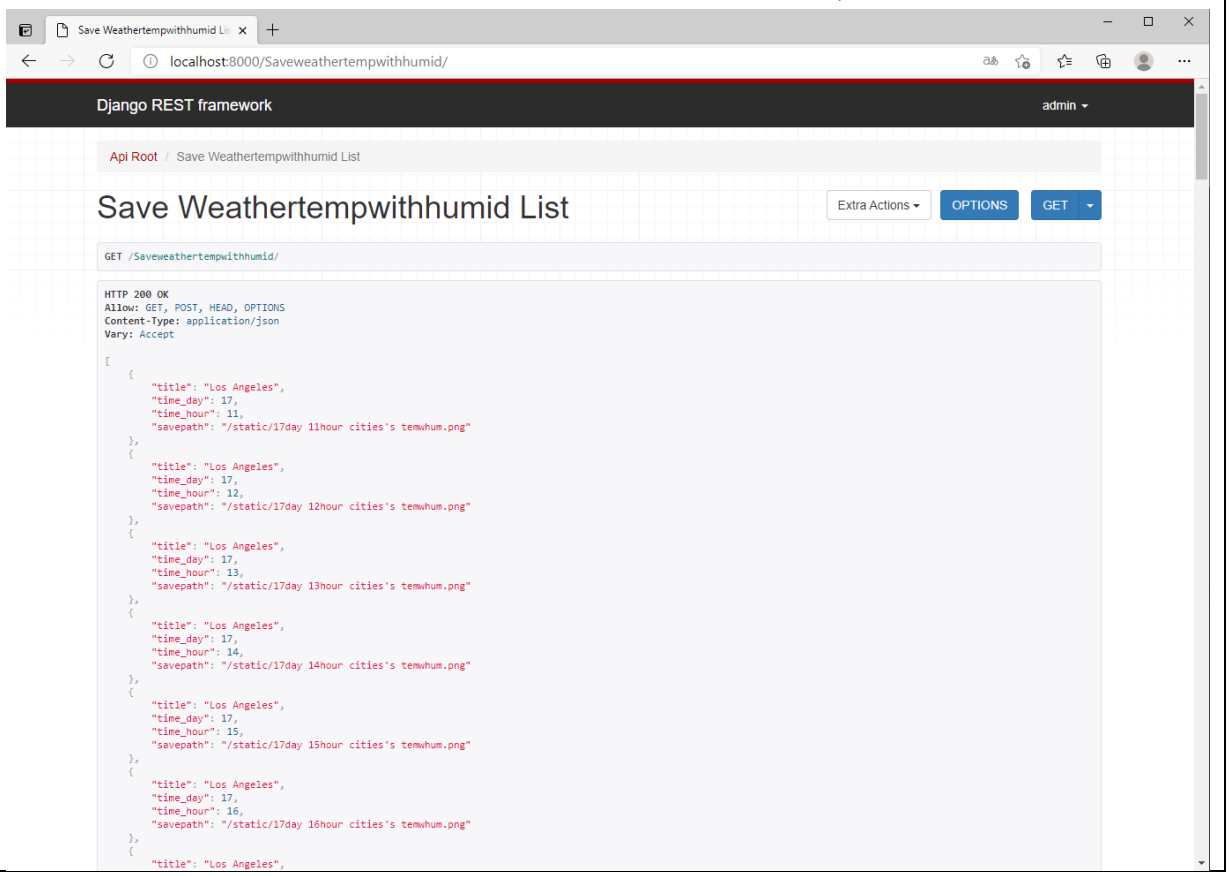


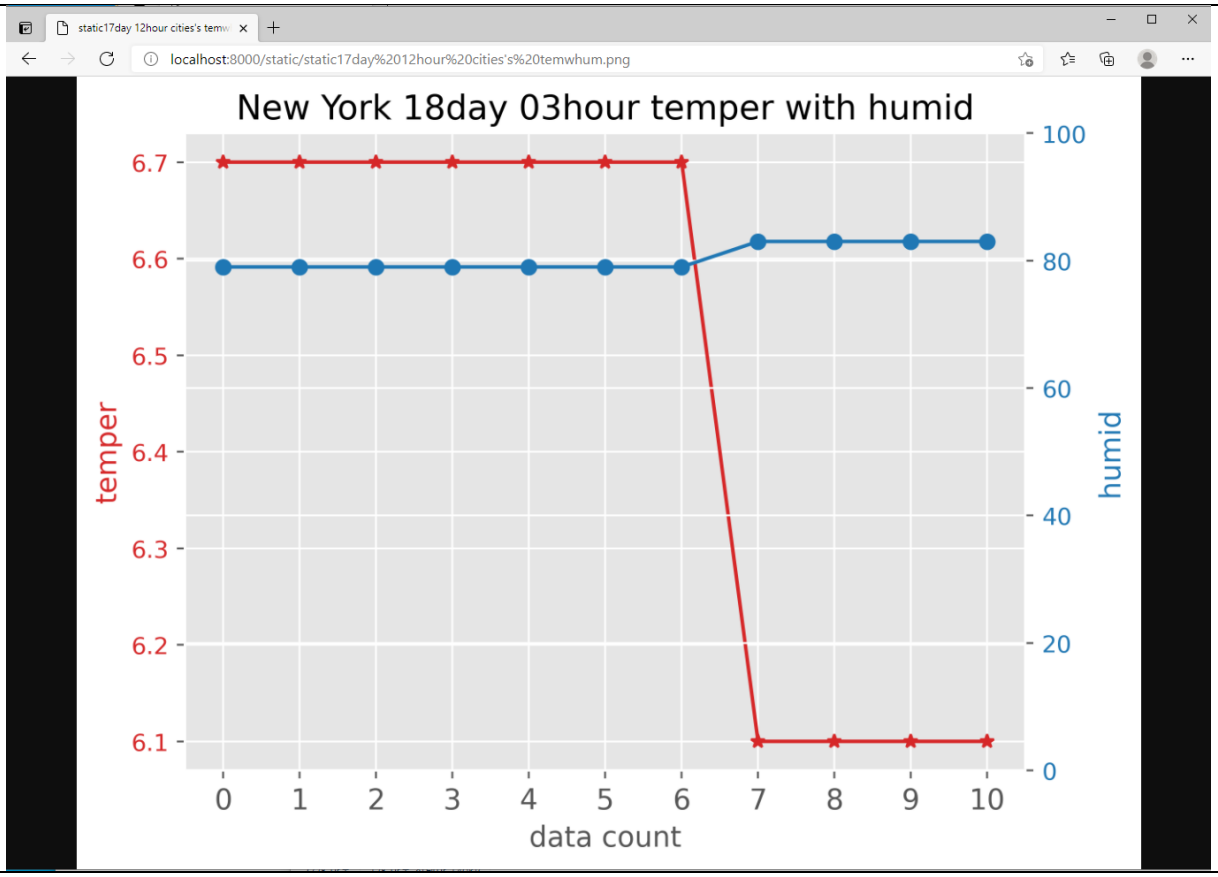
5. 기능 1에 해당하는 데이터 JSON과 웹페이지에 주소 입력 시, 그래프 출력





6. 기능 2에 해당하는 JSON과 웹페이지에 주소 입력 시, 그래프 출력





설계 - 암호화폐(주식)

1. Scrapy와 apscheduler를 활용하여 주식 사이트에서 원하는 정보를 5분 간격으로 crawling

```python
import scrapy
from stockscraper.items import StockscraperItem

class StockbotsSpider(scrapy.Spider):
    name = 'stockbots'
    custom_settings = {'JOBDIR': 'crawl_my_stock_bots1'}
    allowed_domains = ['finance.yahoo.com/quote/BTC-USD?p=BTC-USD&.tsrc=fin-srch',
                       'finance.yahoo.com/quote/ETH-USD?p=ETH-USD&.tsrc=fin-srch',
                       'finance.yahoo.com/quote/LTC-USD?p=LTC-USD&.tsrc=fin-srch']
    start_urls = ['https://finance.yahoo.com/quote/BTC-USD?p=BTC-USD&.tsrc=fin-srch',
                  'https://finance.yahoo.com/quote/ETH-USD?p=ETH-USD&.tsrc=fin-srch',
                  'https://finance.yahoo.com/quote/LTC-USD?p=LTC-USD&.tsrc=fin-srch']

    def parse(self, response):
        quantity = response.xpath('//*[@id="quote-summary"]/div[2]/table/tbody/tr[4]/td[2]/span/text()').extract()
        Price = response.xpath('//*[@id="quote-header-info"]/div[3]/div[1]/div/span[1]/text()').extract()
        days_range = response.xpath('//*[@id="quote-summary"]/div[1]/table/tbody/tr[3]/td[2]/text()').extract()
        title = response.xpath('//*[@id="quote-header-info"]/div[2]/div[1]/div[1]/h1/text()').extract()
        ratio = response.xpath('//*[@id="quote-header-info"]/div[3]/div[1]/div/span[2]/text()').extract()
        open_price = response.xpath('//*[@id="quote-summary"]/div[1]/table/tbody/tr[2]/td[2]/span/text()').extract()

        for row in zip(quantity, Price, days_range,  title, ratio, open_price):
            item = StockscraperItem()
            item['quantity'] = row[0]
            item['Price'] = row[1]
            item['days_range'] = row[2]
            item['title'] = row[3]
            item['ratio'] = row[4]
            item['open_price'] = row[5]

            yield item
```

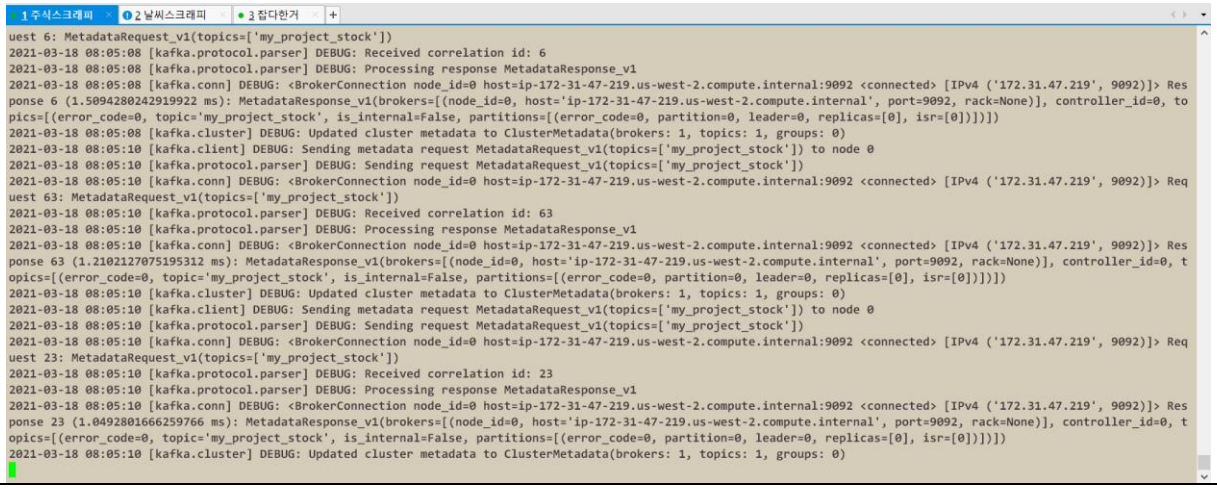
StockbotsSpider

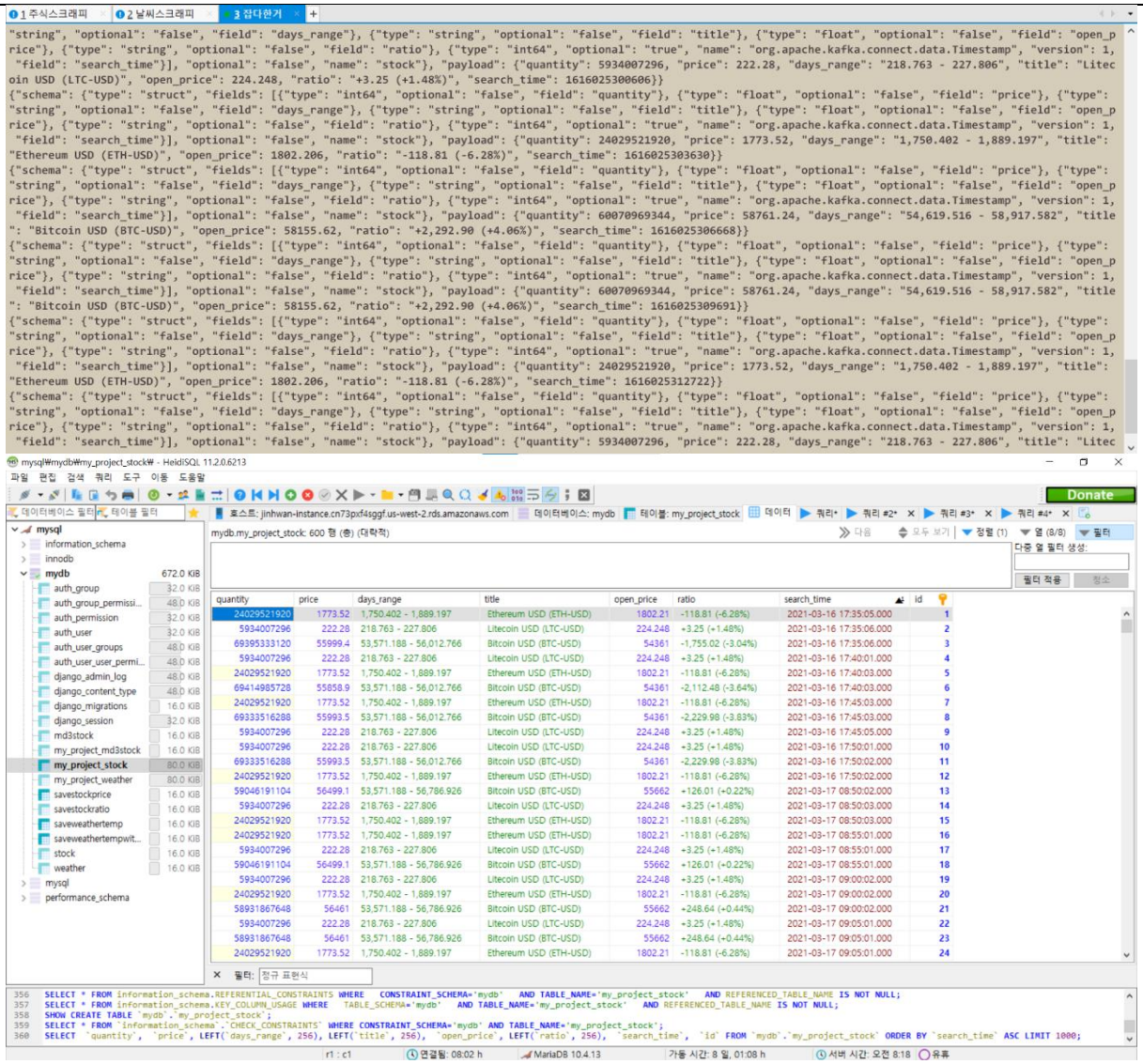
```python
from scrapy.crawler import CrawlerProcess
from scrapy.utils.project import get_project_settings
from apscheduler.schedulers.twisted import TwistedScheduler
from stockscraper.spiders.stockbots import StockbotsSpider

process = CrawlerProcess(get_project_settings())
scheduler = TwistedScheduler()
scheduler.add_job(process.crawl, 'cron', args=[StockbotsSpider], minute='0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55')
scheduler.start()
process.start(False)
```

2. Scrapy의 Pipeline을 활용하여 Scrapy, Kafka, RDS를 연결

```python
from kafka import KafkaProducer
from json import dumps
import time


class StockscraperPipeline(object):

    def __init__(self):
        self.producer = KafkaProducer(acks=0,
                                       compression_type='gzip',
                                       bootstrap_servers=['localhost:9092'],
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))

    def process_item(self, item, spider):
        item = dict(item)
        item["quantity"] = int(item["quantity"].replace(",", ""))
        item["Price"] = float(item["Price"].replace(",", ""))
        item["open_price"] = float(item["open_price"].replace(",", ""))
        data = {"schema": {"type": "struct", "fields": [{"type": "int64", "optional": "false", "field": "quantity"},{"type": "float",
        self.producer.send("my_project_stock", data)
        self.producer.flush()
```

3. 스크래피를 통한 RDS 데이터 삽입을 확인(사진은 스크래피, 토픽 전달 확인, RDS 데이터 저장 사진)

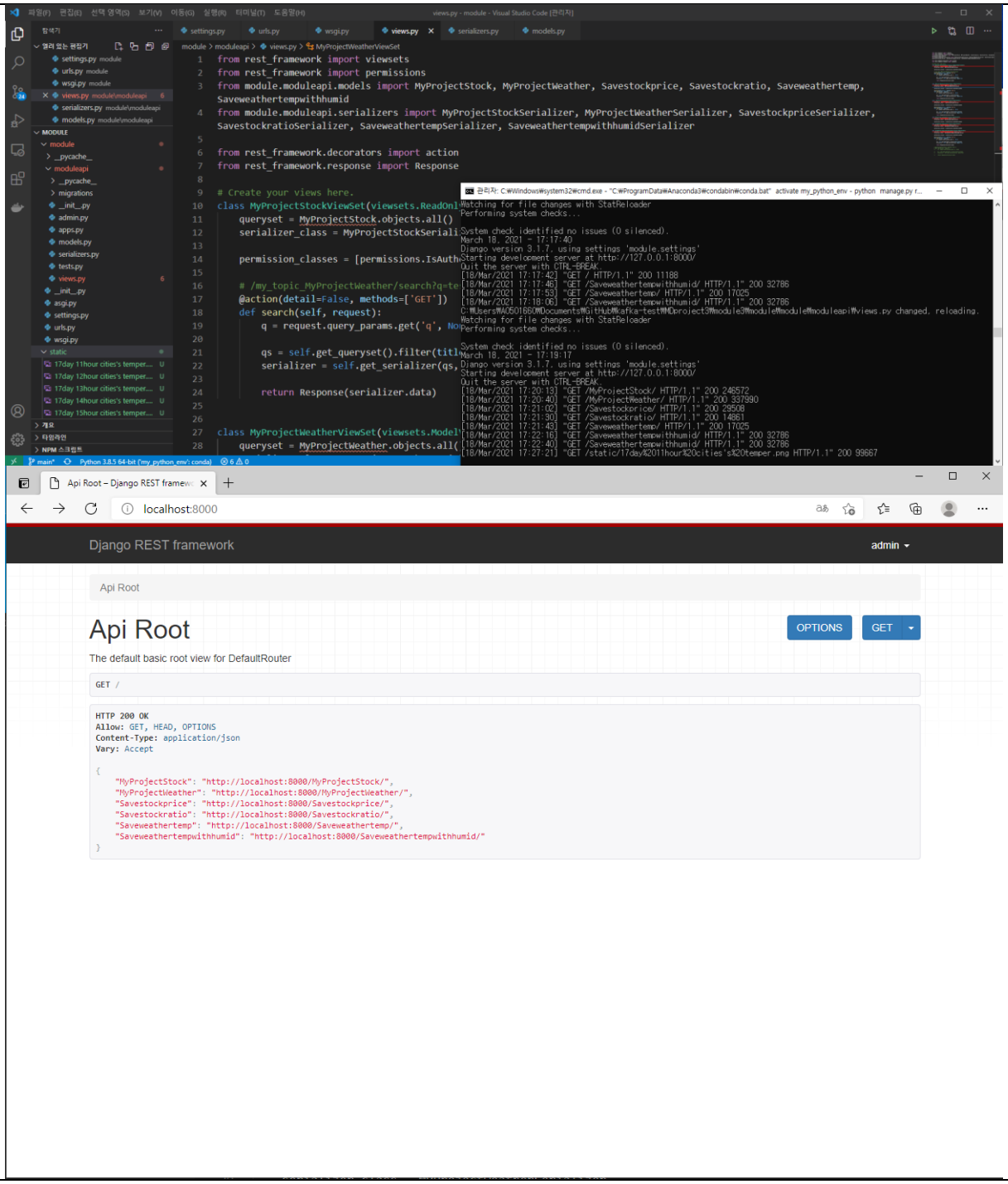


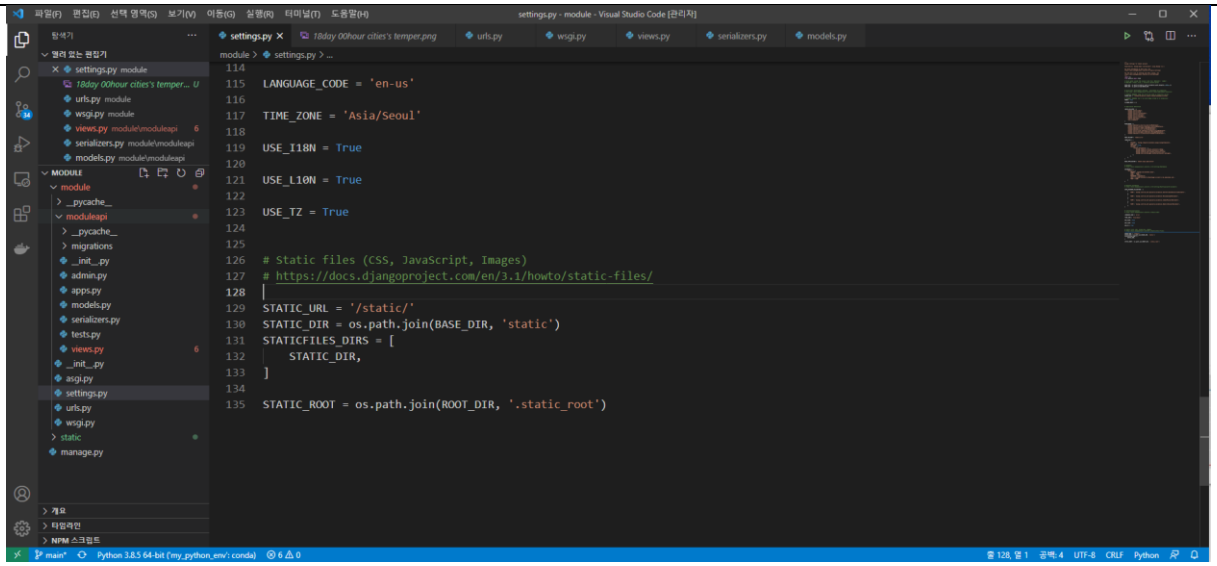


4. RDS에 저장된 데이터를 활용하여 시각화를 구현하고 파일로 저장(사진은 기능 1, 기능 2 순서)

```python
    stocks = list(stocks)

    for k in stocks:
        for j in days:
            for i in hours:
                ndata = df[(df["time_hour"] == i) & (df["time_day"] == j) & (df["title"] == k)]
                ndata = ndata.reset_index()
                if len(ndata) != 0:
                    fig = plt.figure()
                    ax1 = fig.add_subplot(1, 1, 1)
                    ax1.plot(ndata["price"], marker=r'o', color=u'blue', linestyle='-', label=k)
                    plt.xticks(range(len(ndata["price"])), range(len(ndata["price"])), rotation=0, fontsize="large")
                    ax1.xaxis.set_ticks_position('bottom')
                    ax1.yaxis.set_ticks_position('left')
                    ax1.set_title("{0}th {1}hour {2} Price".format(j, i, k))
                    plt.xlabel('data check')
                    plt.ylabel('price')
                    plt.legend(loc='best')
                    plt.savefig(k+'price'+j+'day'+i+'hour price.png', dpi=400, bbox_inches='tight')
                    savepath = '/static/'+k+'price'+j+'day'+i+'hour price.png'
                    insertSql = 'INSERT INTO savestockprice(stock_title, time_day, time_hour, savepath) VALUES(%s, %s, %s, %s)'
                    try:
                        time.sleep(1)
                        self.cursor.execute(insertSql, (k, j, i, savepath))
                        print("데이터 추가 완료")
                        self.connectdb.commit()
                    except Exception as e:
                        print(e)
                    plt.show()

        elif "Litecoin" in i:
            Ldf["ratio"] = df["ratio"][df["title"] == i]
        else:
            Edf["ratio"] = df["ratio"][df["title"] == i]

    Bdf["ratio"] = Bdf["ratio"].map(lambda x: self.ratio_parse(x))
    Ldf["ratio"] = Ldf["ratio"].map(lambda x: self.ratio_parse(x))
    Edf["ratio"] = Edf["ratio"].map(lambda x: self.ratio_parse(x))

    fig = plt.figure()
    ax1 = fig.add_subplot(1, 1, 1)
    ax1.plot(Bdf["ratio"], marker=r'o', color=u'blue', linestyle='-', label='Bitcoin')
    ax1.plot(Ldf["ratio"], marker=r'+', color=u'red', linestyle='--', label='Litecoin')
    ax1.plot(Edf["ratio"], marker=r'*', color=u'green', linestyle='-.', label='Ethereum')
    ax1.xaxis.set_ticks_position('bottom')
    ax1.yaxis.set_ticks_position('left')
    ax1.set_title("Rate by cryptocurrency")
    plt.xlabel('data check')
    plt.ylabel('ratio')
    plt.legend(loc='best')
    plt.savefig('crypto_ratio_plot.png', dpi=400, bbox_inches='tight')
    savepath = '/static/crypto_ratio_plot.png'
    insertSql = """INSERT INTO savestockratio(datacount, savepath) VALUES(%s, %s)"""
    try:
        time.sleep(1)
        self.cursor.execute(insertSql, (str(len(df)), savepath))
        print("데이터 추가")
        self.connectdb.commit()
    except Exception as e:
        print(e)
    plt.show()
```

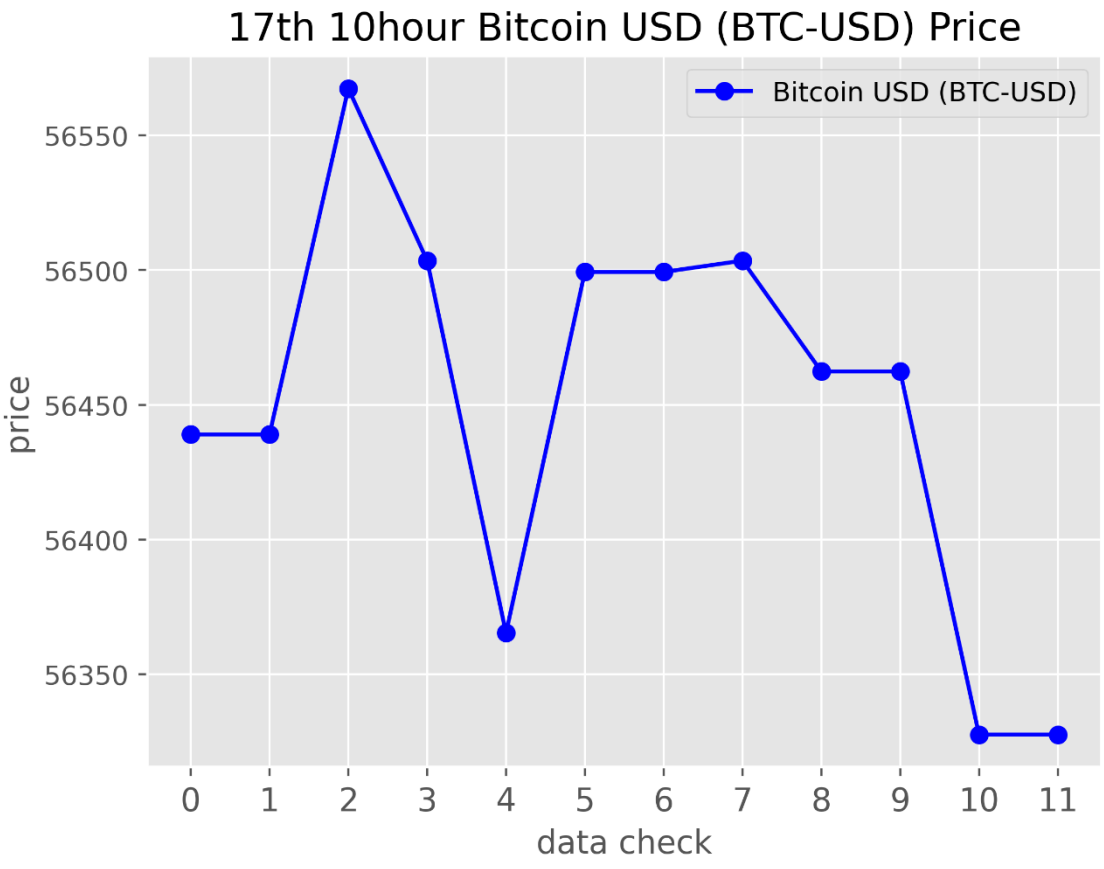
5. Python Django를 활용하여 RestAPI 구축하고 기능과의 연동 준비 완료(사진은 서버 구동, 메인 페이지, 사진을 저장한 Static 관련 코드)

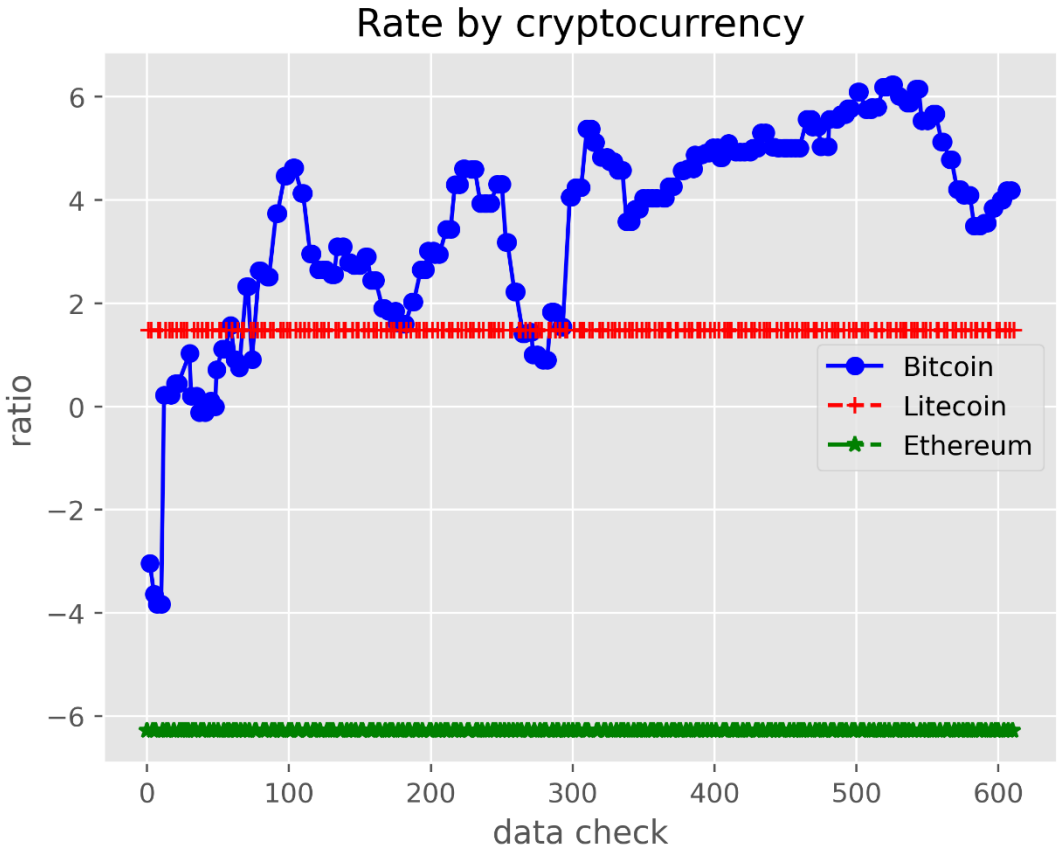

views.py - module - Visual Studio Code (관리자)
settings.py
urls.py
wsgi.py
views.py
serializers.py
models.py
module > moduleapi > views.py > MyProjectWeatherViewSet
from rest_framework import viewsets
from rest_framework import permissions
from module.moduleapi.models import MyProjectStock, MyProjectWeather, Savestockprice, Savestockratio, Saveweathertemp, Saveweathertempwithhumid
from module.moduleapi.serializers import MyProjectStockSerializer, MyProjectWeatherSerializer, SavestockpriceSerializer, SavestockratioSerializer, SaveweathertempSerializer, SaveweathertempwithhumidSerializer
from rest_framework.decorators import action
from rest_framework.response import Response
# Create your views here.
class MyProjectStockViewSet(viewsets.ReadOnlyModelViewSet):
queryset = MyProjectStock.objects.all()
serializer_class = MyProjectStockSerializer
permission_classes = [permissions.IsAuthenticated]
# /my_topic_MyProjectWeather/search?q=test
@action(detail=False, methods=['GET'])
def search(self, request):
q = request.query_params.get('q', None)
qs = self.get_queryset().filter(title__icontains=q)
serializer = self.get_serializer(qs, many=True)
return Response(serializer.data)
class MyProjectWeatherViewSet(viewsets.ModelViewSet):
queryset = MyProjectWeather.objects.all()
관리자: C:\Windows\system32\cmd.exe - "C:\ProgramData\Anaconda3\condabin\conda.bat" activate my_python_env - python manage.py r...
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
March 18, 2021 - 17:17:40
Django version 3.1.7, using settings 'module.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[18/Mar/2021 17:17:42] "GET / HTTP/1.1" 200 11188
[18/Mar/2021 17:17:46] "GET /Saveweathertempwithhumid/ HTTP/1.1" 200 32786
[18/Mar/2021 17:17:53] "GET /Saveweathertemp/ HTTP/1.1" 200 17025
[18/Mar/2021 17:18:06] "GET /Saveweathertempwithhumid/ HTTP/1.1" 200 32786
C:\Users\WA050160\Documents\GitHub\Kafka-test\MDproject3\module3\module\module\moduleapi\views.py changed, reloading.
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
March 18, 2021 - 17:19:17
Django version 3.1.7, using settings 'module.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[18/Mar/2021 17:20:13] "GET /MyProjectStock/ HTTP/1.1" 200 246572
[18/Mar/2021 17:20:40] "GET /MyProjectWeather/ HTTP/1.1" 200 337990
[18/Mar/2021 17:21:02] "GET /Savestockprice/ HTTP/1.1" 200 29508
[18/Mar/2021 17:21:30] "GET /Savestockratio/ HTTP/1.1" 200 14861
[18/Mar/2021 17:21:43] "GET /Saveweathertemp/ HTTP/1.1" 200 17025
[18/Mar/2021 17:22:16] "GET /Saveweathertempwithhumid/ HTTP/1.1" 200 32786
[18/Mar/2021 17:22:40] "GET /Saveweathertempwithhumid/ HTTP/1.1" 200 32786
[18/Mar/2021 17:27:21] "GET /static/17day%2011hour%20cities's%20temper.png HTTP/1.1" 200 99867
Api Root – Django REST framew
localhost:8000
Django REST framework
admin
Api Root
Api Root
OPTIONS
GET
The default basic root view for DefaultRouter
GET /
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
"MyProjectStock": "http://localhost:8000/MyProjectStock/",
"MyProjectWeather": "http://localhost:8000/MyProjectWeather/",
"Savestockprice": "http://localhost:8000/Savestockprice/",
"Savestockratio": "http://localhost:8000/Savestockratio/",
"Saveweathertemp": "http://localhost:8000/Saveweathertemp/",
"Saveweathertempwithhumid": "http://localhost:8000/Saveweathertempwithhumid/"
}



구현 - 암호화폐(주식)

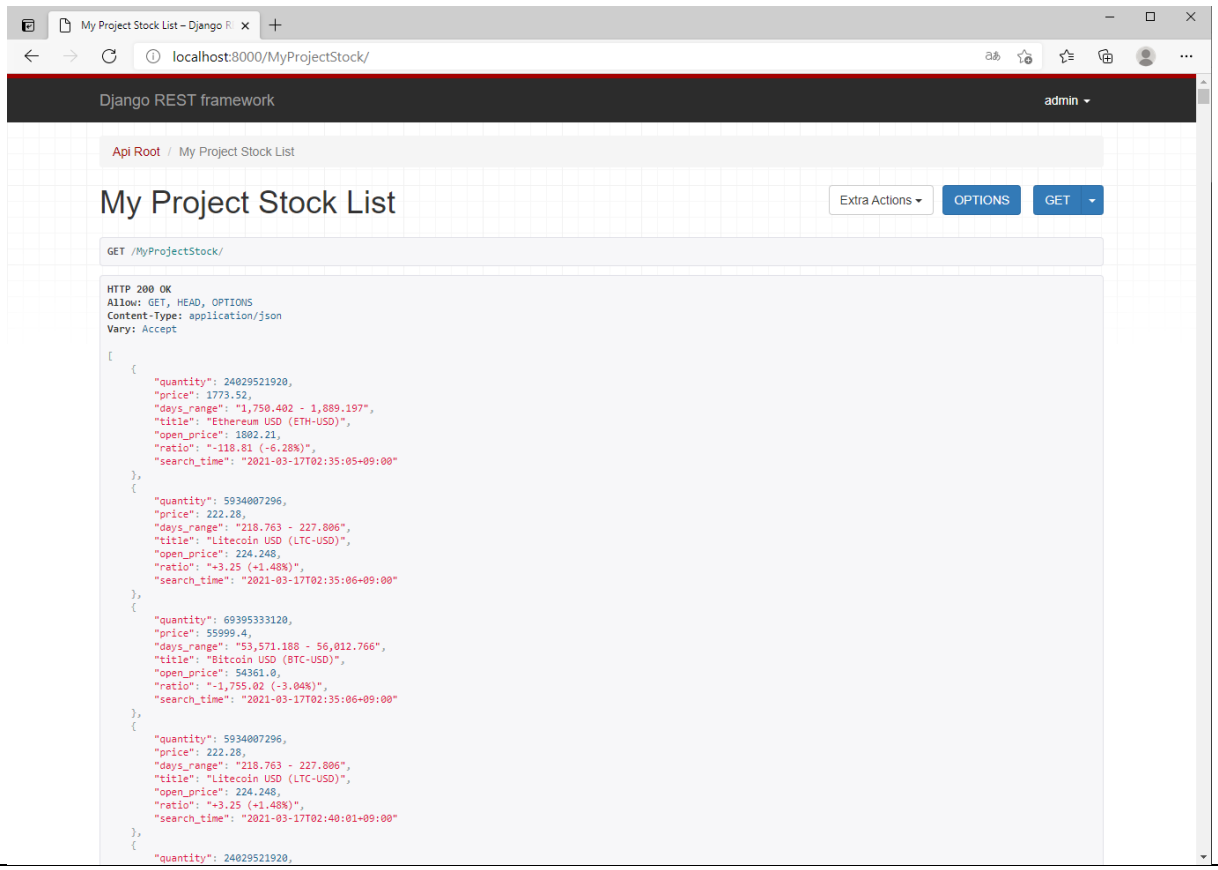
1. Python의 matplotlib 모듈과 RDS의 데이터를 활용하여 그래프로 시각화 구현 (사진은 기능 1, 기능 2 순서)(★기능 2의 경우 2개의 데이터가 동일한 데이터를 가져오는 것을 확인하였으며, 수정해보려 했으나, 결국 실패했습니다.)



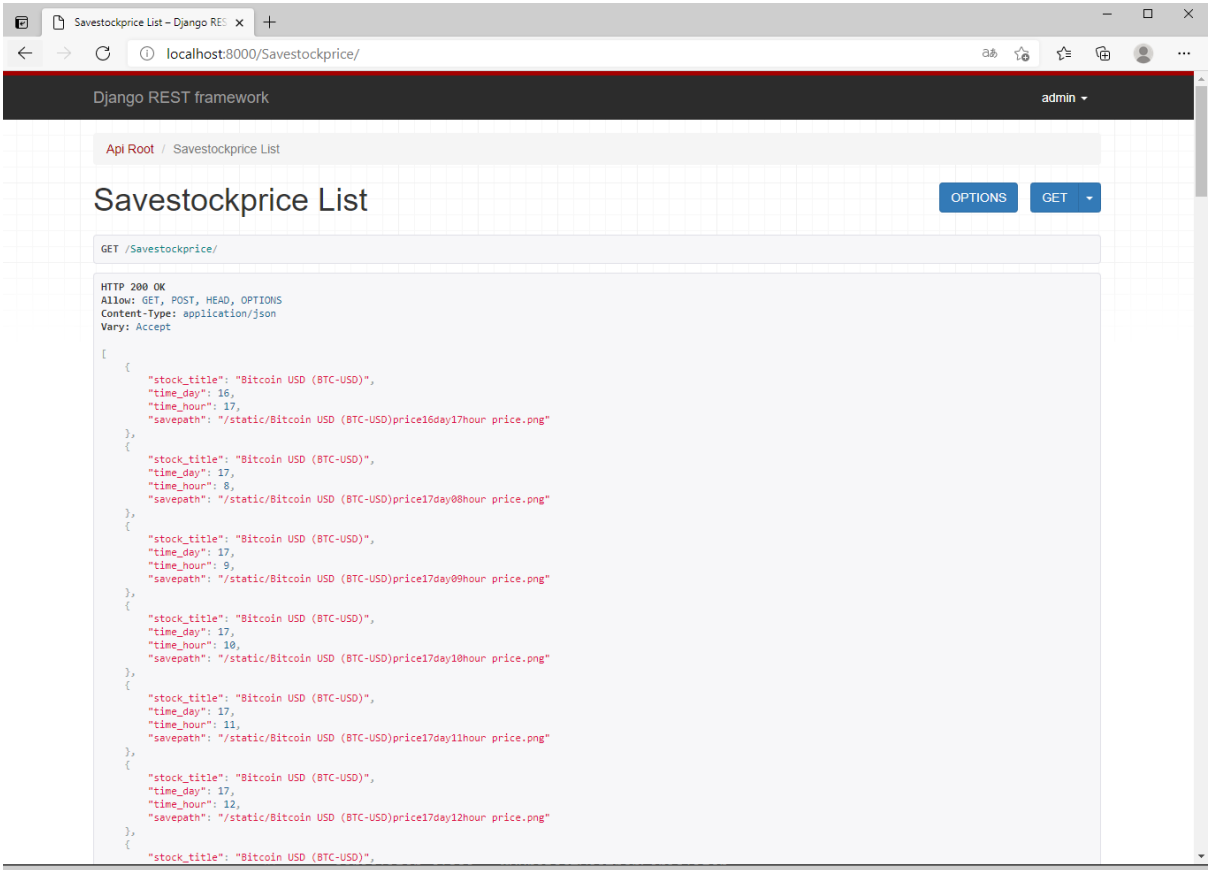


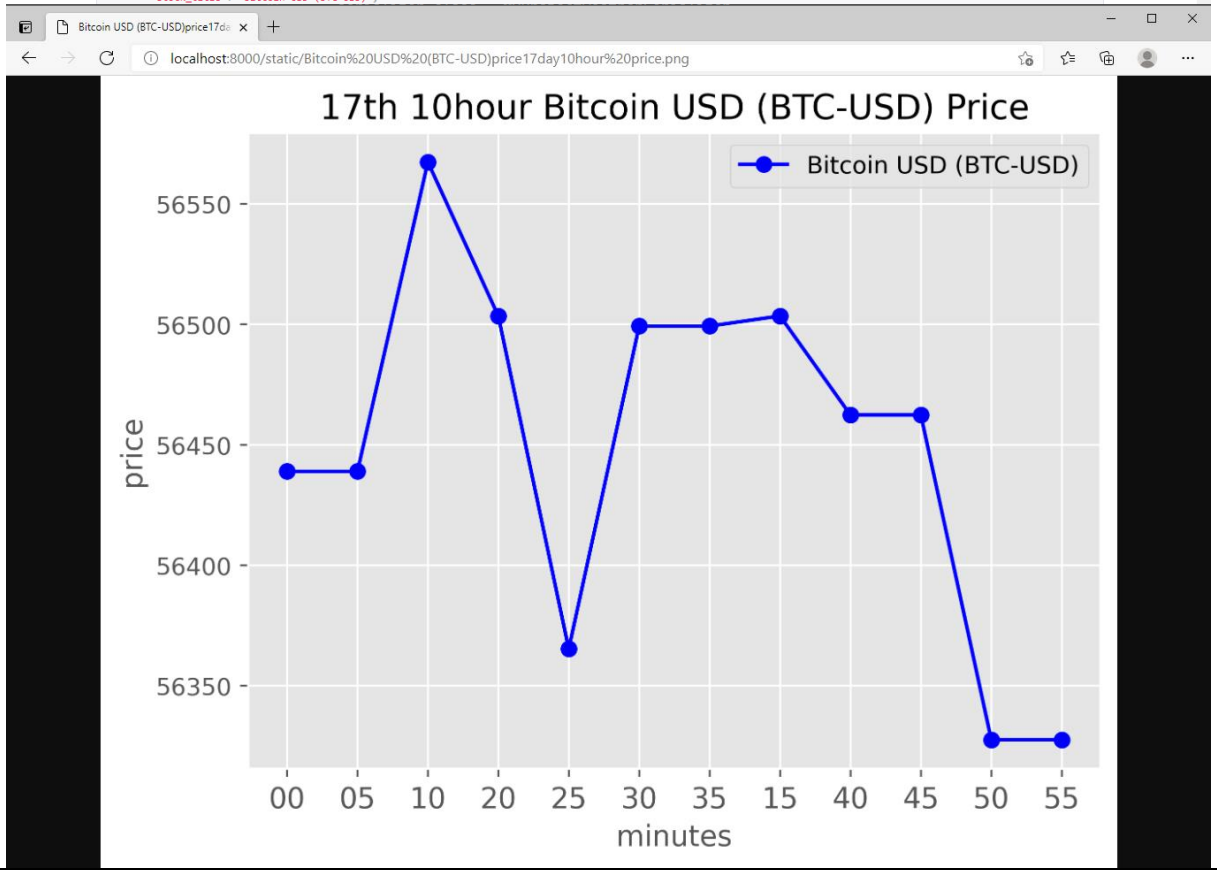
2. Python Django를 활용하여 RestfulAPI에 데이터 JSON파일과 경로 입력 시, 그래프 출력 기능 구현
3. 그래프 출력을 위한 사진이 모여 있는 Django의 Static 폴더

4. 기능 3에 해당하는 주식 데이터 JSON 출력

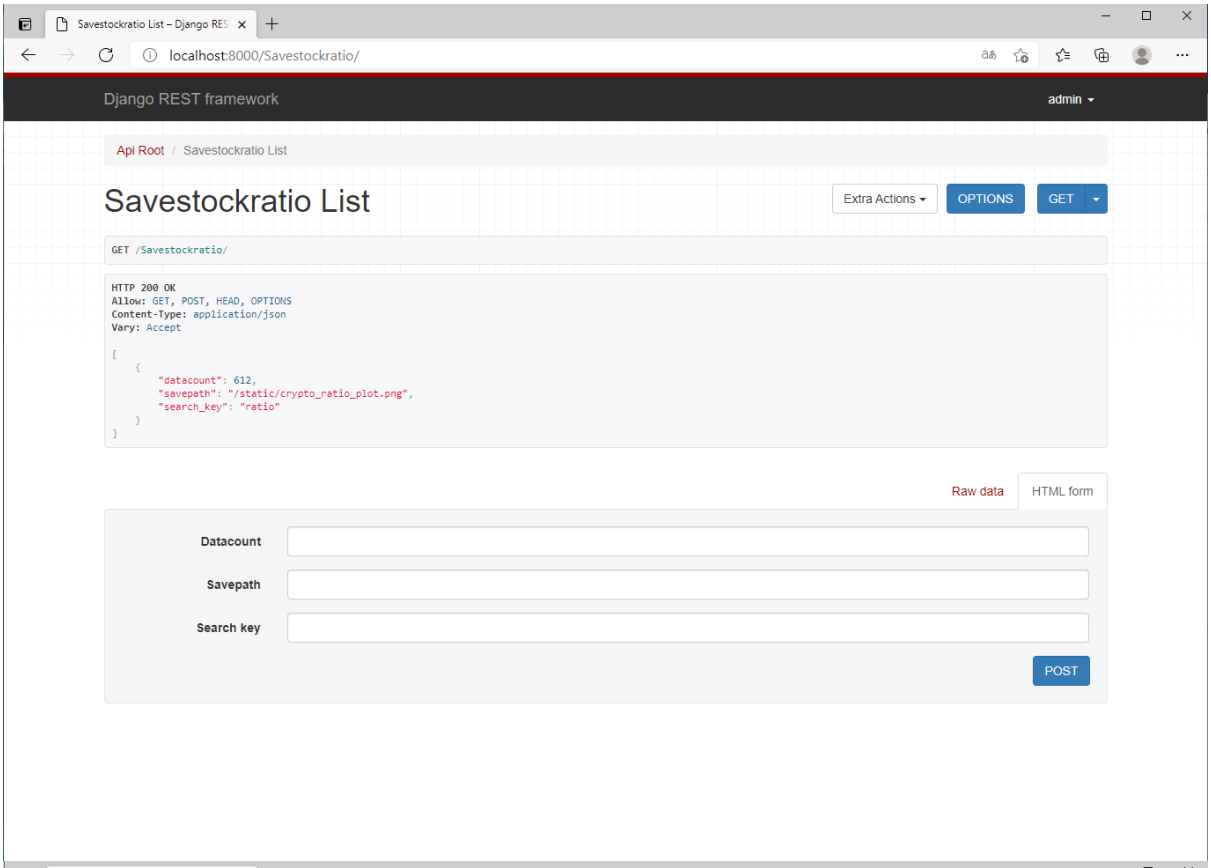


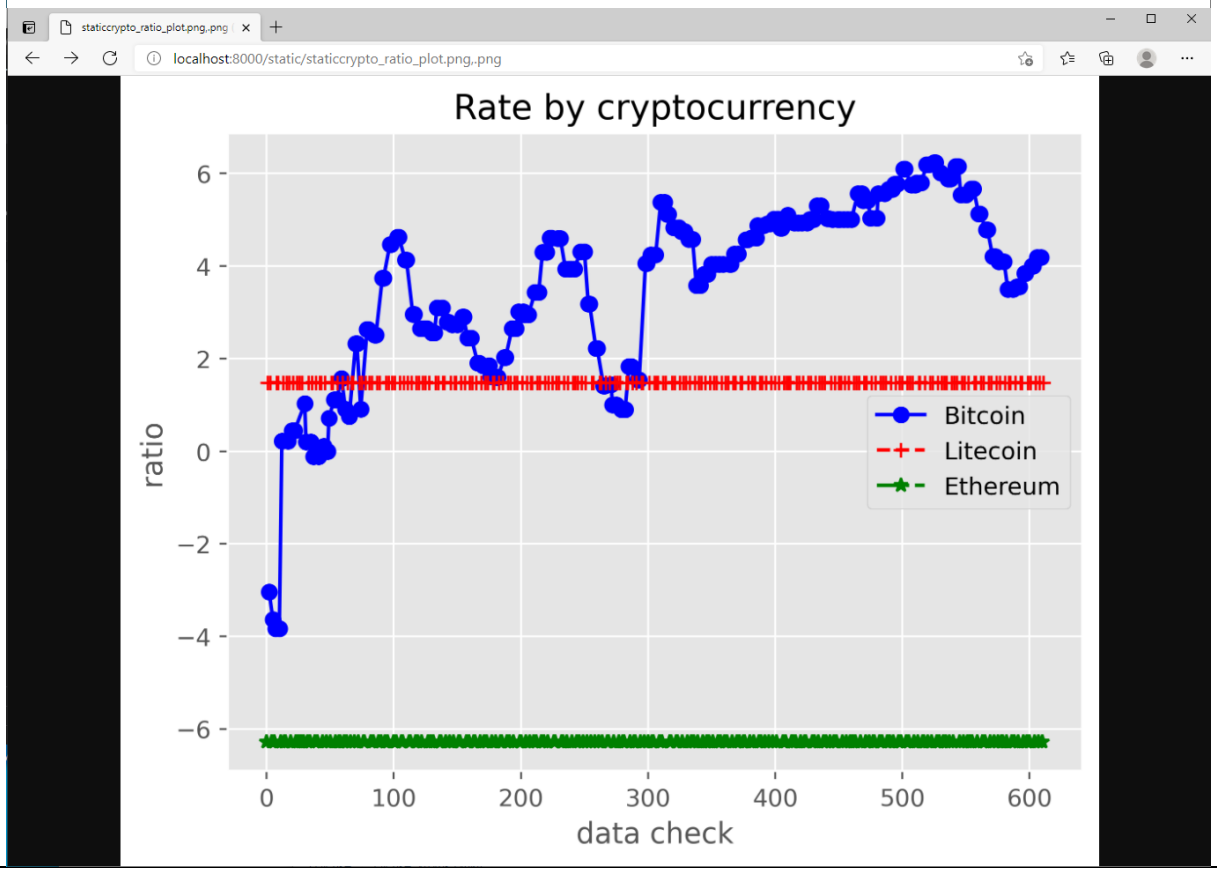
5. 기능 1에 해당하는 데이터 JSON과 웹페이지에 주소 입력 시, 그래프 출력





6. 기능 2에 해당하는 JSON과 웹페이지에 주소 입력 시, 그래프 출력





RDS 정보	1. 주소 jinhwan-instance.cn73pxf4sggf.us-west-2.rds.amazonaws.com 2. 계정 admin 3. 비밀번호 1q2w3e4r5t

■ 프로젝트 후기

금번 프로젝트는 전체 과정에서 첫번째 팀 프로젝트였습니다. 이전에도 2인 규모의 소규모 과제를 진행한적이 있었지만, 프로젝트는 4명과 함께 이번 과정에서 배운 모든 것을 활용한 프로젝트였고, 제 인생에 있어서는 처음으로 총괄 책임을 맡아 진행해본 프로젝트였습니다.

처음에는 프로젝트를 진행하기 앞서 진행 방식을 정하고자(메인 & 서브 프로그래머 방식 / 개인 파트별로 나누어 이후 합쳐서 조율)하였으며, 개개인의 역량 향상을 위하여 개인 파트별로 나누어 이후 합쳐서 조율하는 방식을 택했습니다.

하지만 개개인의 역량 파악이 생각보다 덜 되었고, 프로젝트 구조 자체를 처음에 잘못 이해하여 파트별로 나누는 것에 있어서 문제가 발생하였고, 서로가 비슷한 속도로 진행되어 파트를 합쳐서 조율하는 것이 힘들 것으로 예상되었고, 앞서 정했던 진행 방식을 변경하여 메인 & 서브 프로그래밍 방식과 파트별로 나누어 코딩하는 방식을 결합하여, 제가 총괄 책임을 맡고, 조원 분들을 각자 메인 프로그래머로 지정하여 파트별로 작성을 부탁한 뒤, 필요한 곳마다 작성을 도우는 방식으로 진행을 하였습니다.

결과적으로, 좋은 점도 있었고 안 좋은 점도 있었습니다. 일단 프로젝트를 진행하는 속도를 향상시킬 수 있었습니다. 각자 맡은 부분을 책임자로써 필요한 부분마다 도와주어서 지연 시간을 최소화하였습니다. 하지만 아쉬우면서도 팀원들에게 미안했던 점은 너무 제 위주로 진행되어 개인의 역량 향상을 방해하고 제 방식으로 프로그램이 작성되었다는 점이였습니다.

앞으로는 진행 전에 프로젝트를 확실하게 이해하고 역할 분담을 확실하게 하여, 최대한 간섭을 덜 하여 개인의 역량 향상을 보장하고, 모두가 만족할 수 있는 프로젝트를 만들 수 있도록 노력해가겠다는 생각이 들었으며, 첫 책임자의 귀중한 경험을 잊지 않고 앞의로의 팀 프로젝트에서 책임자이든 팀원이든 좀 더 체계적인 활동에 도움이 될 경험이었습니다.