교통 안내 비서

4박 - 박호윤, 박지수, 박지호, 박현민

0. 목차

1. 팀원 소개

박호윤
프로젝트 전체 플로우, 로직 설계
기상정보 데이터 검색기능
대중교통경로, 역혼잡도검색기능
대중교통 가중치 계산로직 설계

박지호
역혼잡도 가중치 로직설계
데이터 시각화

박지수
자동차 경로 검색 기능
자동차 가중치 계산로직 설계

박현민
각종 데이터 필터링, 쿠킹
발표자료 작성

2. 개발기획

개발 배경

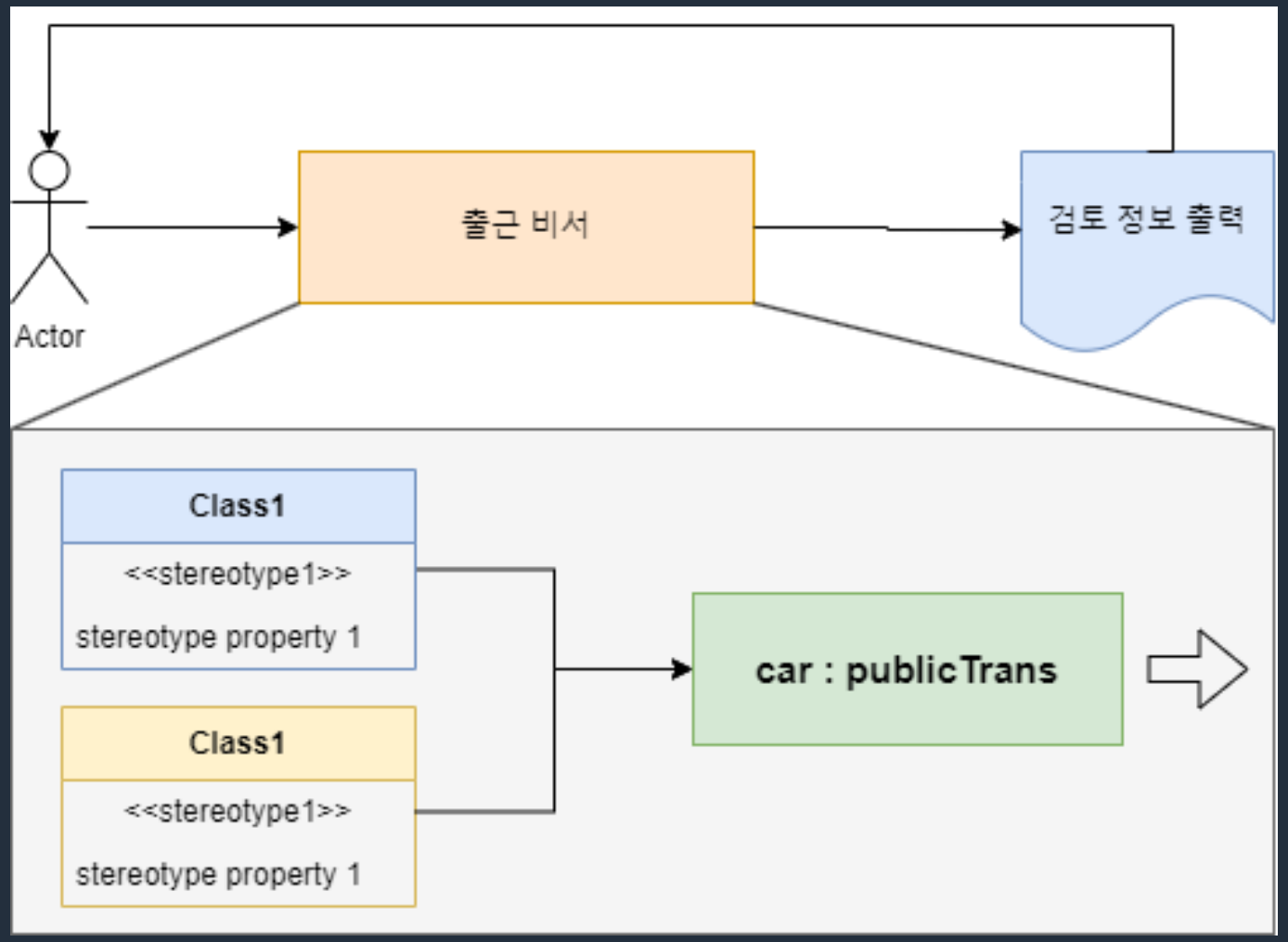
비 오는 명절 2시간 지옥철 vs 6시간 자차 운전

폭설 내리는 날 1시간 지하철 3회 환승 vs 2시간 자차 운전

개발 배경

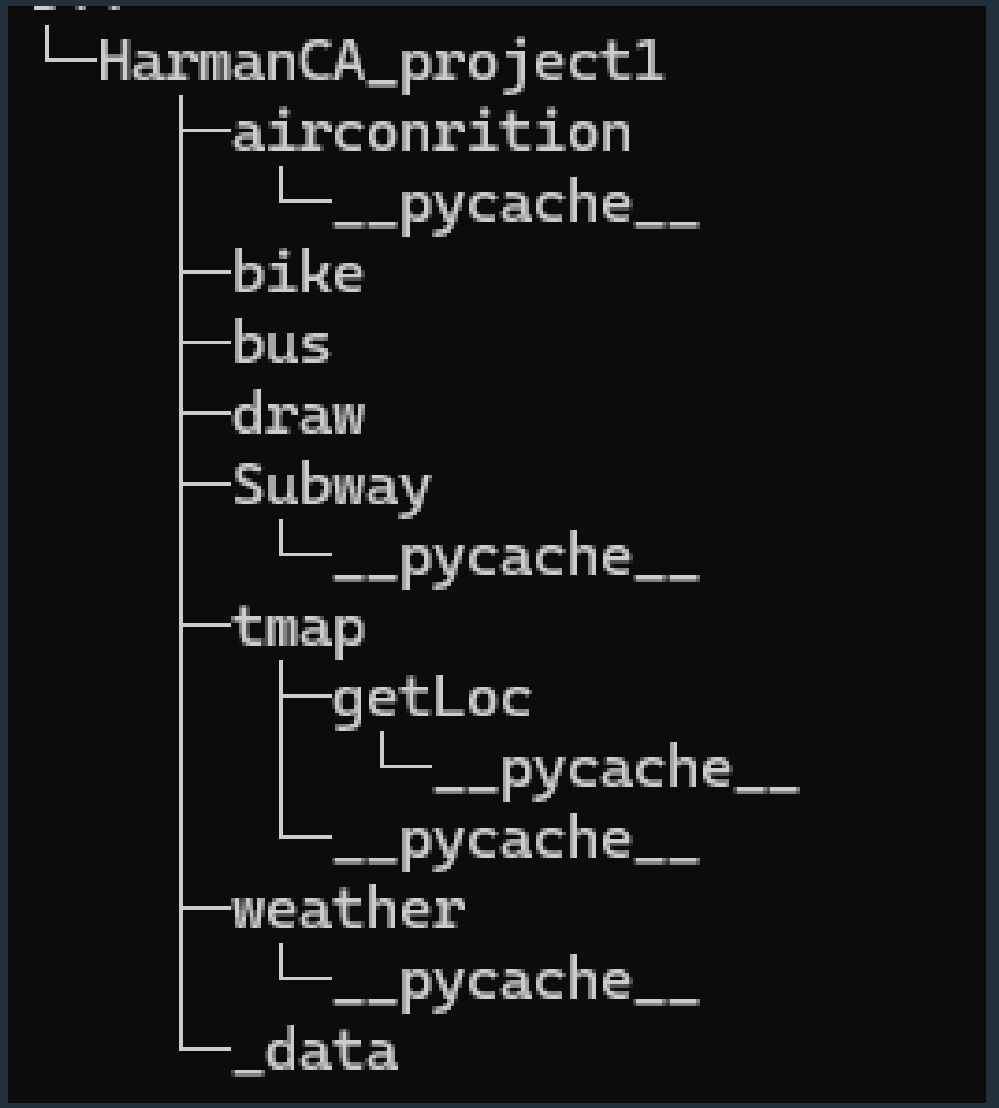
- 기존 경로 추천 서비스의 한계
- 날씨, 혼잡도와 같은 조건을 고려한 맞춤형 경로 추천 필요
- 가장 빠른 길 보다 **가장 편리한 길**을 제공하자!

교통 안내 비서

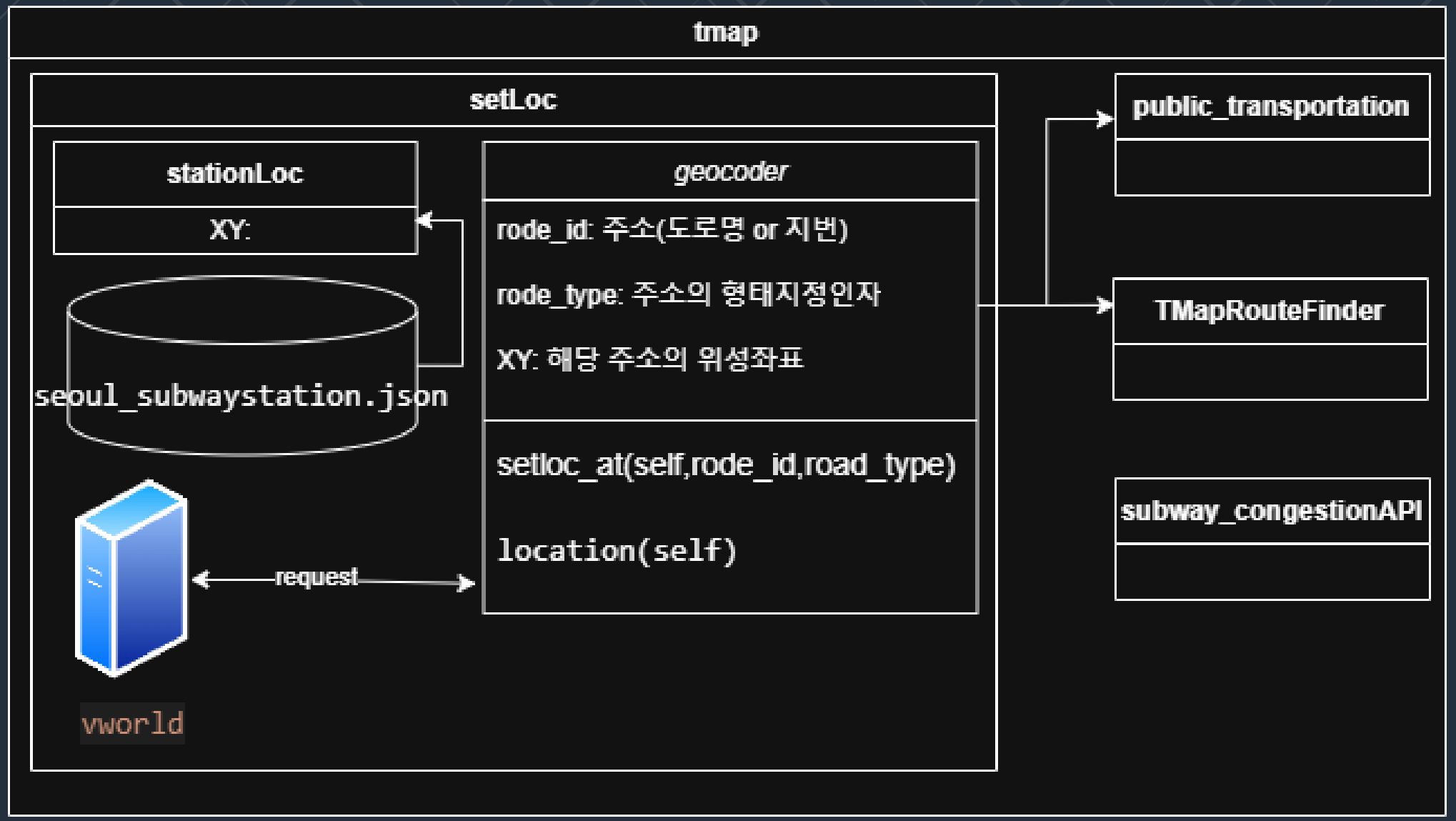

출근 비서
검토 정보 출력
Actor
Class1
<<stereotype1>>
stereotype property 1
Class1
<<stereotype1>>
stereotype property 1
car : publicTrans

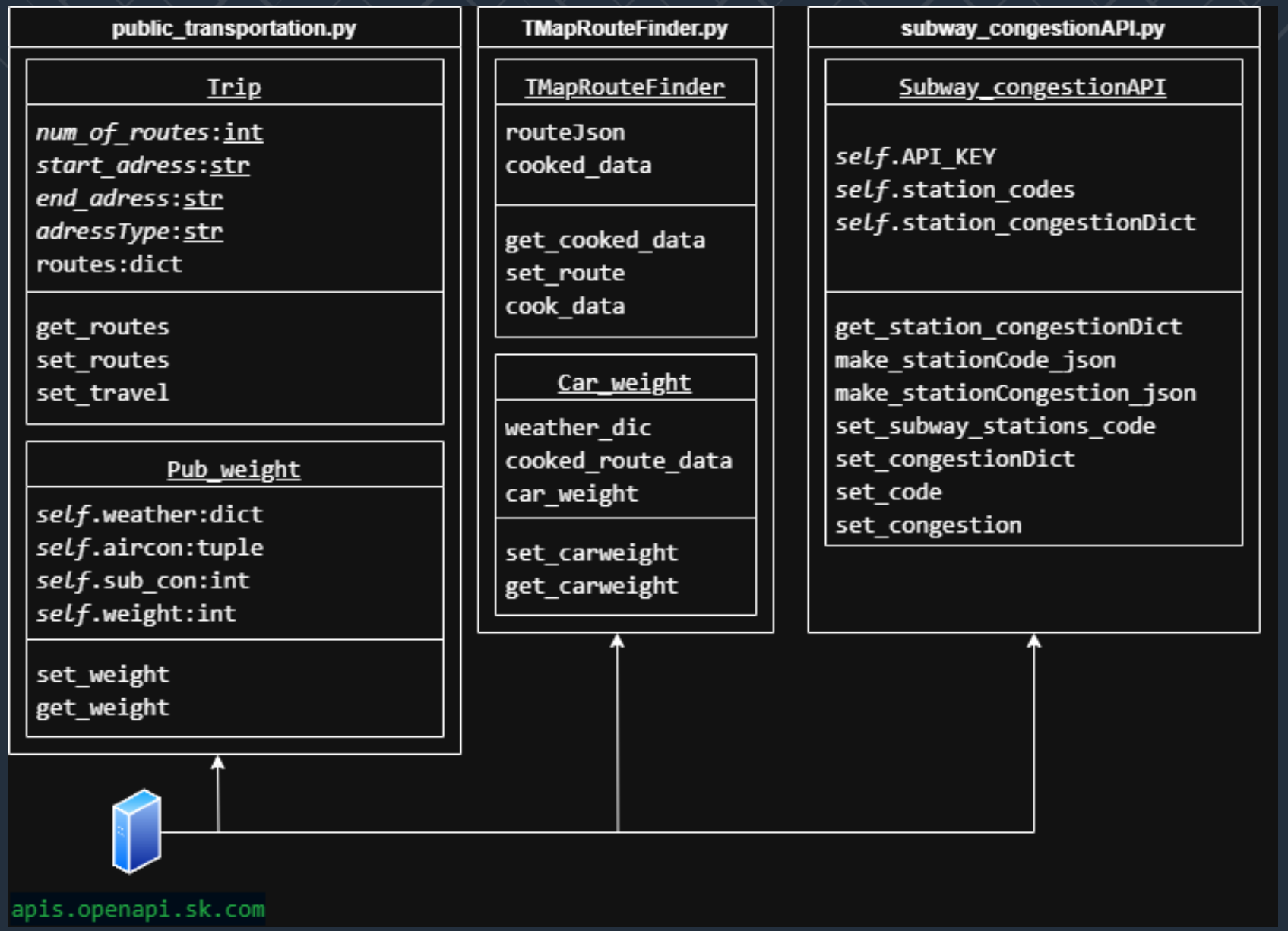
3. 프로그램

- 클래스구조


HarmanCA_project1
airconrition
__pycache__
bike
bus
draw
Subway
__pycache__
tmap
getLoc
__pycache__
__pycache__
weather
__pycache__
_data

tmap


tmap
setLoc
stationLoc
XY:
seoul_subwaystation.json
vworld
request
geocoder
rode_id: 주소(도로명 or 지번)
rode_type: 주소의 형태지정인자
XY: 해당 주소의 위성좌표
setloc_at(self,rode_id,road_type)
location(self)
public_transportation
TMapRouteFinder
subway_congestionAPI


public_transportation.py
Trip
num_of_routes:int
start_adress:str
end_adress:str
adressType:str
routes:dict
get_routes
set_routes
set_travel
Pub_weight
self.weather:dict
self.aircon:tuple
self.sub_con:int
self.weight:int
set_weight
get_weight
TMapRouteFinder.py
TMapRouteFinder
routeJson
cooked_data
get_cooked_data
set_route
cook_data
Car_weight
weather_dic
cooked_route_data
car_weight
set_carweight
get_carweight
subway_congestionAPI.py
Subway_congestionAPI
self.API_KEY
self.station_codes
self.station_congestionDict
get_station_congestionDict
make_stationCode_json
make_stationCongestion_json
set_subway_stations_code
set_congestionDict
set_code
set_congestion
apis.openapi.sk.com

CLASS SubwayCongestion

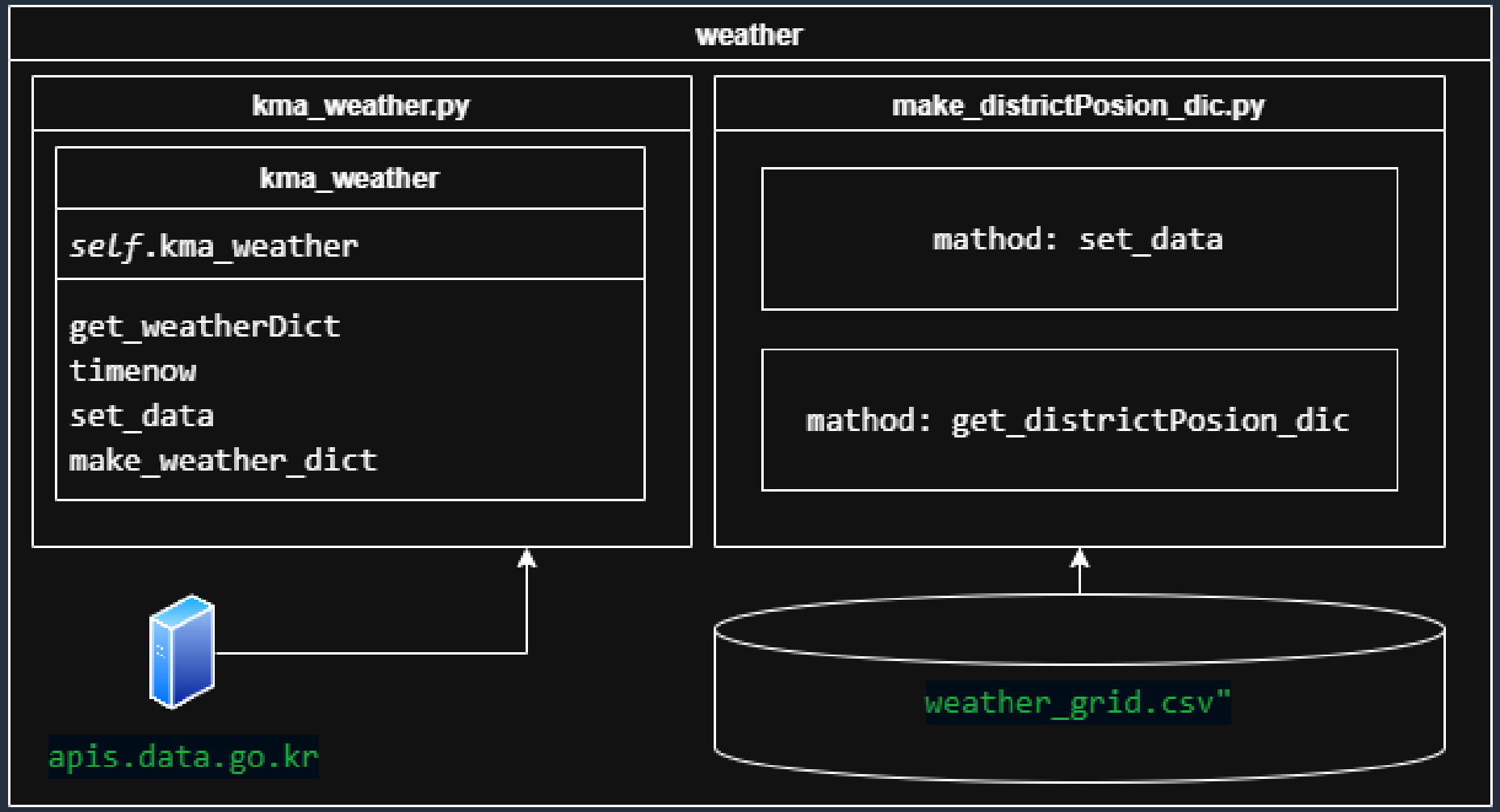
Subway_congestion.py

SubwayCongestion

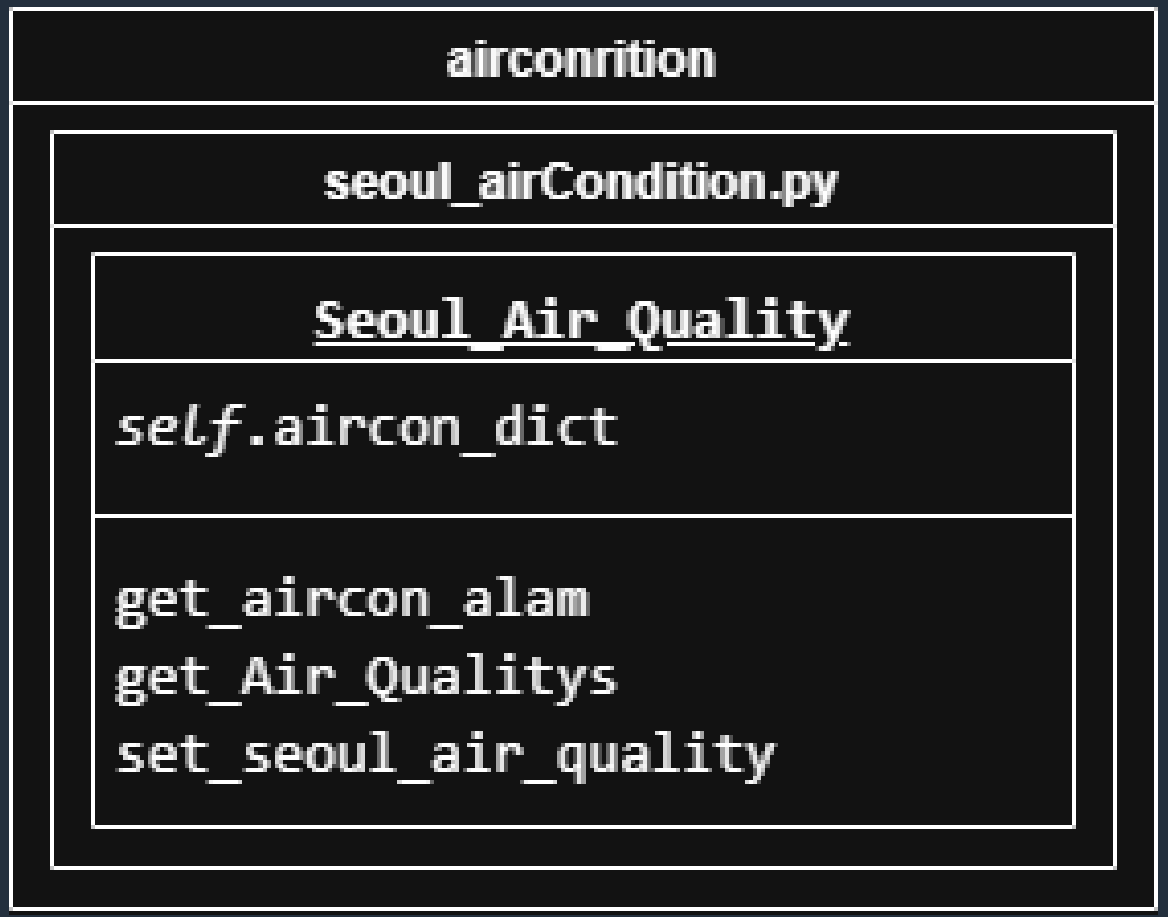
self.stations
self.condict
self.con_avg
self.weight

get_weight
set_weight
print_congestion_result

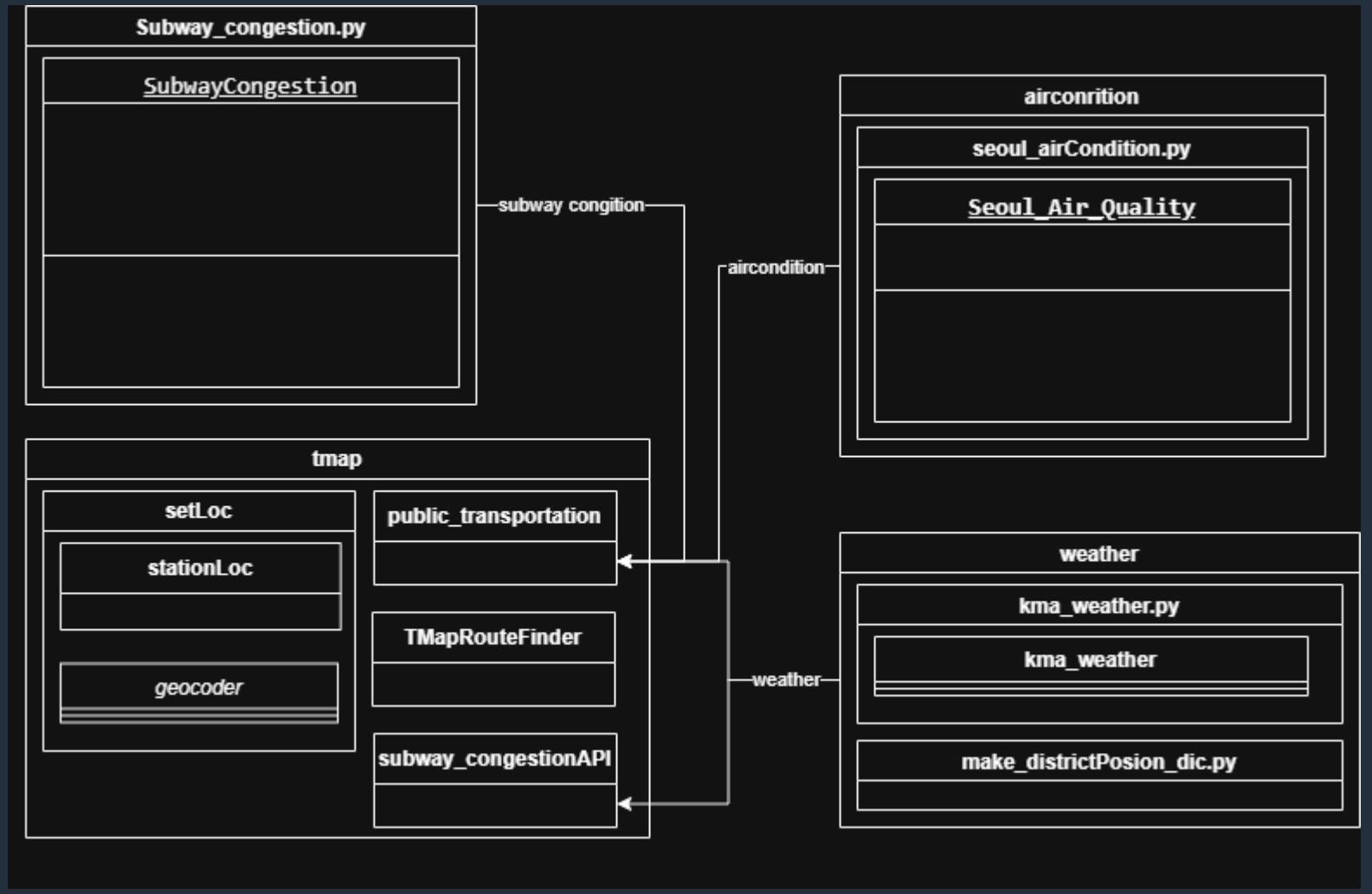
weather


weather
kma_weather.py
kma_weather
self.kma_weather
get_weatherDict
timenow
set_data
make_weather_dict
make_districtPosion_dic.py
mathod: set_data
mathod: get_districtPosion_dic
apis.data.go.kr
weather_grid.csv"

aircondition



전체 클래스 구조


Subway_congestion.py
SubwayCongestion
subway congition
airconrition
seoul_airCondition.py
Seoul_Air_Quality
aircondition
tmap
setLoc
stationLoc
geocoder
public_transportation
TMapRouteFinder
subway_congestionAPI
weather
kma_weather.py
kma_weather
make_districtPosion_dic.py
weather

가중치 설정

대중교통 가중치

혼잡도
혼잡도 40 미만: **가중치 1** 혼잡도 40 이상: **가중치 0.5** 혼잡도 60 이상: **가중치 0.3** 혼잡도 80 이상: **가중치 0.2**

```python
def set_weight(self):

    self.con_avg
    effective = self.con_avg

    # if/elif를 통해 단위 계단 함수의 조건에 따른 multiplier 결정
    if effective >= 80:
        multiplier = 0.2
    elif effective >= 60: #혼잡
        multiplier = 0.3
    elif effective >= 40: #보통
        multiplier = 0.5
    else:
        multiplier = 1  #여유(1에 가까울수록 여유로움움)


    weight = multiplier
    return weight
```

대중교통 가중치

기온	강수량
~-5°C : **가중치 0.2** -5°C ~ 0°C: **가중치 0.4** 0°C ~ 5°C: **가중치 0.6** 5°C ~ 20°C: **가중치 1** 20°C ~ 30°C **가중치 0.5** 그 외: **가중치 0.3**	60mm 이상: **가중치 0.5** 그 외: **가중치 1**

```
def set_weight(self):
    pm25w,pm10w = self.air_con

    temp, rain, wet, rainform, _ = self.weather.values()

    weather_w = 1
    if temp <= -5: weather_w = 0.2
    if temp <= 0: weather_w = 0.5
    if temp <= 5: weather_w = 0.6
    if temp <= 25: weather_w = 1
    if temp <= 30: weather_w = 0.5
    else: weather_w = 0.3

    rain_w = 1
    if rain >= 60: 0.5

    wlist = [self.weight,self.sub_con,pm25w,pm10w,weather_w,rain_w]
    self.weight = sum(wlist)/len(wlist)

def get_weight(self):
    return self.weight
```

대중교통 가중치

미세먼지
PM10 150㎍/㎥ ~ 300㎍/㎥ : 가중치 0.7 300㎍/㎥ ~ : 가중치 0.5 PM25 75㎍/㎥ ~ 150㎍/㎥ : 가중치 0.7 150㎍/㎥ ~ : 가중치 0.5

```
def get_aircon_alam(self):
    pm10a,pm25a = 1, 1
    pm10 = self.aircon_dict["PM10"]
    pm25 = self.aircon_dict["PM25"]
    if pm10 >= 150: pm10a = 0.7
    elif pm10 >= 300: pm10a = 0.5
    if pm25 >= 75: pm25a = 0.7
    elif pm25 >= 150: pm25a = 0.5
```

자차 가중치

요금
0원 ~ 1,000원 : 가중치 1
1,000원 ~ 1,500원 : 가중치 0.75
1,500원 ~ 2,000원 : 가중치 0.5
2,000원 ~ 3,000원 : 가중치 0.25
3,000원 ~ 5,000원 : 가중치 0.1
5,000원 ~ : 가중치 0

```
if total_fare <= 1000: fare_weight = 1
elif total_fare <= 1500: fare_weight = 0.75
elif total_fare <= 2000: fare_weight = 0.5
elif total_fare <= 3000: fare_weight = 0.25
elif total_fare <= 5000: fare_weight = 0.1
else: fare_weight = 0

print(fare_weight)

# rain_factor = 1 / (self.RN1 + 100)  # ☔ 비/눈 가중치
snow_factor = 1
if self.weather_dic["PTY"] == "3": snow_factor = 0.001 # 눈 올때 자차 운행 안함
```

자차 가중치

이동시간
40분 ~ 60분 : 가중치 0.1
20분 ~ 40분 : 가중치 0.9
10분 ~ 20분 : 가중치 0.75
60분 ~ 80분 : 가중치 0.45
80분 ~ 100분 : 가중치 0.3
0분 ~ 10분 : 가중치 0.1

```
# 🚗 주행시간 별 가중치
if distance_time <= 600: distance_weight = 0.1 # 10분
elif distance_time <= 1200: distance_weight = 0.75 # 20분
elif distance_time <= 2400: distance_weight = 0.9 # 40분
elif distance_time <= 3600: distance_weight = 1 # 60분
elif distance_time <= 4800: distance_weight = 0.45 # 80분
elif distance_time <= 6000: distance_weight = 0.3 # 100분
else: distance_weight = 0.1
```

3. 코드소개

모듈1 - KMA_Weather

```python
def timenow(self):
    """한국의 현재 시간 "20250202"형태로 반환"""
    KST = datetime.timezone(datetime.timedelta(h
    current_time_kst = datetime.datetime.now(KST
    cur_date = current_time_kst.strftime('%Y%m%d
    cur_hour = current_time_kst.strftime('%H00')
    cur_min = current_time_kst.strftime('%M')
    return (cur_date, "0600" if int(cur_min) < 3

def set_data(self):
    """API사용 대한민국 날씨전보JSON 반환"""
    KEY = os.getenv("DATAGOKR_API_KEY")
    district = "종로구"
    todayString, currentTime  = self.timenow()
    x,y = make_districtPosion_dic.get_districtPo
    URL = f"http://apis.data.go.kr/1360000/Vilag
    response = requests.get(URL)
    data = response.json()
    return data
```

```python
def make_weather_dict(self):
    """
    JSON에서 필요한 날씨정보만 추려서 dict형태로 반환
    retrunFormat -> {"T1H":기온, "RN1":강수량, "REH":습도, "PTY":강수형태, "WSD":풍속}
    """
    res = dict()
    data = self.set_data()

    #"T1H":"기온", "RN1":"강수량", "REH":"습도", "PTY":"강수형태", "WSD":"풍속"
    target_category = ["T1H", "RN1", "REH", "PTY", "WSD"]

    # print(data["response"]["body"]["items"]["item"])
    for weather_dict in data["response"]["body"]["items"]["item"]:
        if not weather_dict["category"] in target_category: continue
        res[weather_dict["category"]] = weather_dict["obsrValue"]

    return res

def get_weatherDict(self):
    """T1H=기온, RN1=강수량, REH=습도, PTY=강수형태, WSD=풍속"""
    return self.kma_weather
```

모듈2 - Seoul_Air_Quality

```python
class Seoul_Air_Quality:
    """
    서울시 실시간 미세먼지 데이터 class
    Seoul_Air_Quality_dict : 서울시 실시간 미세먼지 데이터가 저장된 dict
    """
    def __init__(self):
        self.aircon_dict = self.set_seoul_air_quality()

    def set_seoul_air_quality(self):
        api_key = os.getenv("JIHO_SEOUL_API_KEY")
        res = dict()
        url = f"http://openAPI.seoul.go.kr:8088/{api_key}/json/RealtimeCityAir/1/100/"
        response = requests.get(url)
        data = response.json()
        key_names = ["MSRSTE_NM","O3","PM10","PM25","SO2"]
        top_line = data["RealtimeCityAir"]["row"][0]
        # print(top_line)
        for key_name in key_names:
            res[key_name] = top_line[key_name]

        return res
```

모듈3 - tmap (Geocoder)

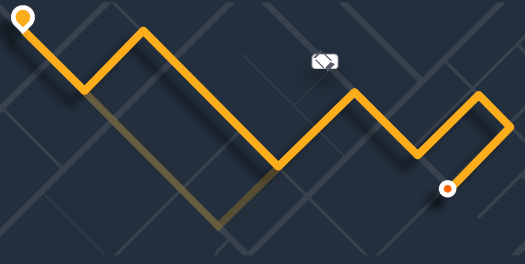
```python
def __init__(self, rode_id:str, rode_type:str):
    self.xy = self.setloc_at(rode_id,rode_type)

def setloc_at(self,rode_id,road_type): #'ROAD':도로명 주소 'PARCEL':지번 주소
    key = os.getenv("V_WORLD_KEY")
    apiurl = "https://api.vworld.kr/req/address?"
    params = {
        "service": "address",
        "request": "getcoord",
        "crs": "epsg:4326",
        "address": rode_id,
        "format": "json",
        "type": road_type,
        "key": key
    }
    response = requests.get(apiurl, params=params)
    if response.status_code != 200: # 응답완료가 아닐시 에러
        return None

    data = response.json()["response"]
    if data["status"] != "OK": #비정상 상태시 에러
        return None

    (x,y) = tuple(data["result"]["point"].values())

    return x,y
```



모듈3 - tmap (public_tra

```python
def __init__(self, num_of_routes:int, start_adress:str, end_adress:str, adressType:str):
    try:
        self.routes = self.set_routes(num_of_routes)
    except:
        print("Fail to  open json")
        self.set_travel(start_adress, end_adress, adressType)
        self.routes = self.set_routes(num_of_routes)

def get_routes(self):
    ''' return [[fare, totalTime, totalWalkTime, station_list], ...] '''
    return self.routes
```

```python
def set_travel (self, start_adress, end_adress, adress_type):
    # API_KEY
    API_KEY = os.getenv("TMAP_PT_KEY")
    TEST_KEY = os.getenv("TMAP_TEST_KEY")

    # API URL
    URL = "https://apis.openapi.sk.com/transit/routes"

    start = Geocoder(start_adress, adress_type)
    end = Geocoder(end_adress, adress_type)


    # 두위치중 하나라도 None이라면 에러
    if not (start.location() and end.location()):
        return None

    sx,sy = start.location()
    ex,ey = end.location()
```

```python
nt):

map_publicTp.json", "r", encoding="UTF-8") as file:
(file)
etaData"]["plan"]["itineraries"]

t[i]["fare"]["regular"]["totalFare"]
route_list[i]["totalWalkTime"]
e_list[i]["totalTime"]

ist()
Y", "BUS"]
oute_list[i]["legs"]:
 = section["mode"]
ype in ["WALK","BUS"] : continue
 section["route"].replace("수도권", "")
 = section["passStopList"]["stationList"]
 = [route_name + " " + station["stationName"] for station in passStopList]
,totalTime,totalWalkTime,station_list])
```

```python
class TMapRouteFinder:
    """..."""

    def __init__(self, api_key: str):
        """
        생성자: API Key 설정
        :param api_key: TMap API Key
        """
        self.api_key = api_key
        self.url_route = "https://apis
        self.headers = {
            "accept": "application/jso
            "appKey": self.api_key,
            "content-type": "applicati
        }

    def get_route(self, start_address:
        """
        자동차 경로 탐색 메서드
        :param start_address: 출발지 주
        :param end_address: 도착지 주소
        :param address_type: 주소 변환
        :param search_option: 경로 탐색
        :return: 경로 정보를 JSON으로 반
        """
        # 🚀 Geocoder 사용하여 주소 → 좌
        start_coords = Geocoder(start_
        end_coords = Geocoder(end_addr

        if not start_coords or not end
            print("⚠️ 좌표 변환 실패로
            return None

        start_x, start_y = start_coords
        end_x, end_y = end_coords
        current_time = datetime.now().strftime("%Y%m%d%H%M%S")

        payload = {...

        # API 요청
        response = requests.pos

        # 응답 확인 및 데이터 추출
        if response.status_code
            data = response.jso

            # 🚀 전체 경로 정보
            summary = data.get(
            total_distance = su
            total_time = summar
            total_fare = summar
            taxi_fare = summary

            # 🚀 경로 세부 정보
            route_list = []
            for feature in data
                properties = fe
                if "description
                    route_list.
                        "구간 설
                        "교통상
                        "거리(m
                        "소요 시
                        "도로명"
                    })
        else:
            print(f"⚠️ API 요청 실패: {response.status_code}, {response.text}")
            return None

        # 🚀 결과 JSON 생성
        result_json = {
            "출발지": start_address,
            "도착지": end_address,
            "총 이동 거리(km)": round(total_distance / 1000, 2),
            "총 소요 시간": f"{total_time // 60}분 {total_time % 60}초",
            "총 요금 정보(원)": f"{total_fare:,}",
            "택시 예상 요금(원)": f"{taxi_fare:,}",
            "경로 상세 정보": route_list
        }

        # 🚀 JSON 데이터를 파일로 저장
        with open("car_route_data.json", "w", encoding="utf-8") as f:
            json.dump(result_json, f, indent=4, ensure_ascii=False)

        # 🚀 JSON 데이터 출력
        print("\n📌 🚗 TMap 자동차 경로 안내 데이터 (JSON 저장 완료)\n")
        print(json.dumps(result_json, indent=4, ensure_ascii=False))

        return result_json
```

모듈4 - SubwayCongestion

```python
class SubwayCongestion:
    def __init__(self, stations: list, condict: dict):
        """…
        self.stations = stations
        self.condict = condict
        self.con_avg = self.set_conavg_stations() #역 평균 혼
        self.weight = self.set_weight() #최종 혼잡도 가중치 결
```

```python
def set_weight(self):

    self.con_avg
    effective = self.c

    # if/elif를 통해 단
    if effective >= 80
    elif effective >=
    elif effective >=
    else:…
```

```python
def get_weight(self):
    return self.weight

def set_conavg_stations(self):
    """
    혼잡도가 비어있는(0)인 지하철의 혼잡도를 채워주고 평균을 계산
    """
    for i, station in enumerate(self.stations):
        if self.condict[station] != 0: break

    target_data = self.condict[self.stations[i]]
    target_idx = i

    stations_len = len(self.stations)
    if target_idx == stations_len - 1:
        target_data = 20 # 평균 CSV 보면서 값 조정해나가면 됨
        for station in self.stations:
            self.condict[station] = target_data

    for j in range(target_idx):
        self.condict[self.stations[j]] = target_data
```

```python
def print_congestion_result(self) -> str:
    """
    평균 및 가중치 출력 매소드

    Returns:
        str: 총 혼잡도와 가중치를 포함한 결과 문자열.
    """
    print (f'역 평균 혼잡도: {self.con_avg:.2f}, 최종 혼잡도 가중치 결과값은 {self.weight:.2f}입니다.')
```

결과

0.4666666666666667:0.5333333333333333

자동차 경로 정보
출발지 : 서울 강서구 화곡로 179
도착지 : 서울 양천구 목동로 201
총 이동 거리(km) : 4.76
총 소요 시간 : 756
총 요금 정보(원) : 0
택시 예상 요금(원) : 10,900

자동차 경로 정보
출발지 : 서울 양천구 목동로 201
도착지 : 서울 강서구 화곡로 179
총 걷는 시간(sec) : 209
역 경로 : ['5호선 목동', '5호선 신정', '5호선 까치산', '5호선 화곡']
총 요금 : 1400