



---

최단경로 탐색 기술이 적용된 건물 그림자 이용 길찾기 서비스

# COOLWAY

김정권의 손칼국수

발표자: 김철호, 조장: 권용현, 팀원: 김도원, 김제윤, 정윤찬



# 목차

## 프로젝트 개요 및 목표요약

- 프로젝트명: COOLWAY
- 목표: 건물 그림자 이용 길찾기
- 관련 SDGs: 지속가능한 도시발전

## 기능 및 구현

- 전체 시스템 구조도
- 주요 기능 및 사용 기술
- UI/UX 스크린샷

## 문제 정의 및 배경

- 도시 열섬 효과 문제
- 현재 네비게이션 앱의 한계
- 선행 사례: COOL WALKS 앱

## 변경 사항 및 개선, 구현결과/시연

- 건물 API 변경 과정
- 경로 탐색 방식 개선
- 성능 향상 결과, 시연

## 개발 목표

- 정량적 목표: 경로탐색 속도, 그림자 최대화
- 정성적 목표: 사용성, 실제 사용 유도

## 사용자 테스트 및 피드백 결과

- 사용자 인터뷰
- 피드백 반영

## 프로젝트 개요 및 목표

### COOLWAY: 그림자 길찾기 서비스

COOLWAY는 최단경로 탐색 기술을 활용하여 건물 그림자  
를 이용한 길찾기 서비스를 제공합니다. 이 혁신적인 접근은  
도시 열섬 효과 완화와 보행자의 열적 쾌적성 향상을 목표로  
합니다. 사용자에게 그늘진 경로를 제안함으로써, 특히 무더운  
날씨에 효과적인 솔루션을 제공합니다.

### SDGs : 목표 11 지속가능한 도시발전

본 프로젝트는 지속가능발전목표(SDGs) 중 '목표 11: 지속가  
능한 도시'에 부합합니다. 도시 환경 활용, 기후변화 대응, 취  
약계층 보호 등 다양한 측면에서 지속가능한 도시 발전에 기여  
할 것으로 기대됩니다.

# 문제 정의 및 배경

## 도시 열섬 효과와 네비게이션의 한계

기후변화로 인한 도시 온도 상승과 도시 열섬 효과(URBAN HEAT ISLAND)가 심각한 사회 문제로 대두되고 있다.

펜실베니아 주립대학교 연구에 따르면, 도시의 건물 높이 대 도로 폭 비율이 높을수록, 즉 높은 건물이 좁은 도로에 그림자를 만들수록 평균 복사온도가 낮아지고 열 쾌적성이 향상된다고 밝혀졌다.

특히 저소득층, 유색인종, 고령자들이 도시 열섬 효과에 불균형적으로 영향을 받고 있어 사회적 형평성 측면에서도 해결책이 필요하다.

현재 대부분의 네비게이션 앱들은 자동차 이용자를 위해 설계되어 최단 거리나 최단 시간 경로만을 제공하며, 보행자의 열적 쾌적성은 고려하지 않는다.



# 개발 목표

## 정량적 목표

- 경로탐색 속도: 3초 이내 완료
- 최단경로 10개 중 최대 그림자 경로 선택
- 실제 건물 폭과 높이 반영한 정확한 그림자 계산
- 사용자 체감 온도 5°C 이상 감소 효과

## 정성적 목표

- 사용할수 있는 UI/UX 구현 ex)출발, 목적지 입력
- 실제 사용 유도를 위한 사용자 경험 최적화
- 팀원 간 협업 만족도 향상
- 지속가능한 도시 발전에 기여하는 서비스 구현

# 전체 시스템 구조

```
2 1개의 사용 위치
3 def is_lat_lon(input_str):
4     try:
5         parts = input_str.replace(' ', ',').split(',')
6         parts = [p.strip() for p in parts if p.strip()]
7         if len(parts) != 2:
8             return False
9         lat = float(parts[0])
10        lon = float(parts[1])
11        return -90 <= lat <= 90 and -180 <= lon <= 180
12    except:
13        return False
14
15 5 개의 사용 위치
16 def get_coords(input_str):
17     if is_lat_lon(input_str):
18         parts = input_str.replace(' ', ',').split(',')
19         parts = [p.strip() for p in parts if p.strip()]
20         lat, lon = float(parts[0]), float(parts[1])
21         return lat, lon
```

## 1. 사용자 입력 (input\_handler.py)

출발지/도착지 텍스트 입력 (장소명 또는 위경도 좌표)  
geocode\_place를 통해 장소명을 위경도 좌표로 변환

```
1  import osmnx as ox
2  import networkx as nx
3
4  2 개의 사용 위치
5  def map_point_to_nearest_node(G, lat, lon, max_dist=500):
6      node, dist = ox.nearest_nodes(G, X=lon, Y=lat, return_dist=True)
7      if dist > max_dist:
8          print(f"[경고] 입력 좌표에서 도로 네트워크까지 거리가 {dist:.1f}m로 너무"
9          " 멀어 경로 탐색을 수행하지 않습니다.")
10         return None
11     return node
12
13  3 개의 사용 위치
14  def get_k_shortest_routes(start_lat, start_lon, end_lat, end_lon, dist=10):
15      """
16          OSMnx 기반 도보 네트워크에서 최단 경로 후보 k개를 반환
17          :return: [ [(lon, lat), ...], ... ] # 경로별 좌표 리스트
18      """
19      G_multi = ox.graph_from_point(center_point=(start_lat, start_lon), dist=dist,
20                                     # MultiDiGraph → DiGraph 변환 (가장 짧은 옛지만 남김)
21                                     G = nx.DiGraph()
22                                     for u, v, data in G_multi.edges(data=True):
23                                         if G.has_edge(u, v):
24                                             if data.get('length', float('inf')) < G[u][v].get('length',
```

## 2. 경로 탐색 (osmnx\_route.py)

get\_k\_shortest\_routes 함수 호출  
OpenStreetMap(OSM)에서 도보 네트워크 데이터를 다운로드  
출발지와 도착지 사이의 최단 경로 후보 k개(예: 10개)를 생성

```
1  import osmnx as ox
2  import geopandas as gpd
3  from .config import OSM_DATA_PATH, DEFAULT_BUILDING_HEIGHT
4
5  3 개의 사용 위치
6  def fetch_building_polygons(center_lat, center_lon, dist=500):
7      """
8          OSM에서 중심좌표 기준 반경(dist, m) 내 건물 폴리곤 데이터를 가져옵니다.
9      """
10     gdf = ox.features_from_point(
11         center_point=(center_lat, center_lon),
12         tags={"building": True},
13         dist=dist
14     )
15     buildings = gdf[gdf.geometry.type.isin(['Polygon', 'MultiPolygon'])].copy()
16     buildings['height'] = DEFAULT_BUILDING_HEIGHT
17     return buildings
18
19  def save_buildings_to_file(buildings_gdf, path=OSM_DATA_PATH):
20      """
21          건물 폴리곤 GeoDataFrame을 파일로 저장(GeoJSON 등)
22      """
23      buildings_gdf.to_file(path, driver="GeoJSON")
```

## 3. 지리 데이터 수집 (osm\_buildings.py)

탐색된 모든 경로를 포함하는 경계 상자 (Bounding Box) 계산

fetch\_building\_polygons 함수를 통해 경계 상자 내의 건물 폴리곤(외곽선) 데이터를 OSM에서 다운로드

# 전체 시스템 구조

```
f get_sun_position(lat, lon, date_time=None):
    """
    위도, 경도, 날짜/시간을 받아 태양의 고도와 방위각을 계산합니다.
    :param lat: 위도
    :param lon: 경도
    :param date_time: datetime 객체 (기본값: 현재 시간)
    :return: (고도, 방위각) 튜플 (단위: 도)
    """

    if date_time is None:
        date_time = datetime.now()
    # 반드시 한국 시간대(Asia/Seoul)로 변환
    if date_time.tzinfo is None:
        date_time = pytz.timezone('Asia/Seoul').localize(date_time)
    else:
        date_time = date_time.astimezone(pytz.timezone('Asia/Seoul'))
    times = pd.DatetimeIndex([date_time])
    solpos = solarposition.get_solarposition(times, lat, lon)
    altitude = float(solpos['apparent_elevation'].iloc[0])
    azimuth = float(solpos['azimuth'].iloc[0])
    return altitude, azimuth
```

## 4. 태양 위치 및 그림자 계산

태양 위치 계산 (sun\_position.py):

get\_sun\_position 함수로 현재 시간 기준 태양의 고도(alitude)와 방위각(azimuth)을 계산.

그림자 생성 (shadow\_calc.py):

project\_shadow 함수가 건물 데이터와 태양 위치 정보를 이용해 각 건물이 만드는 그림자 폴리곤을 기하학적으로 계산.

```
def calculate_shadow_coverage(route_coords, shadow_gdf):
    """
    도보 경로와 그림자 영역이 얼마나 겹치는지 계산합니다.
    :param route_coords: [(lon, lat), ...] 형태의 경로 좌표 리스트
    :param shadow_gdf: 그림자 영역 GeoDataFrame
    :return: (전체 경로 길이, 그림자 구간 길이, 그림자 비율)
    """

    route_line = LineString(route_coords)
    route_gdf = gpd.GeoDataFrame(geometry=[route_line], crs=shadow_gdf.crs)

    # 그림자 영역과 경로의 교차 구간 추출
    shadow_union = shadow_gdf.unary_union
    shadow_section = route_line.intersection(shadow_union)

    # 전체 경로 길이
    total_length = route_line.length
    # 그림자 구간 길이
    shadow_length = 0.0
    if shadow_section.is_empty:
        shadow_length = 0.0
    elif shadow_section.geom_type == "LineString":
        shadow_length = shadow_section.length
    elif shadow_section.geom_type == "MultiLineString":
```

## 5. 경로 분석 및 최적 경로 선정 (shadow\_analysis.py)

calculate\_shadow\_coverage 함수를 통해 각 경로 후보와 생성된 그림자 영역의 겹치는 길이를 분석.

각 경로의 '그림자 비율'(전체 길이 대비 그림자 구간 길이)을 계산.

그림자 비율이 가장 높은 경로를 최적 경로로 선정.

```
# 지도 중심 자동 설정
if map_center is None:
    mid_idx = len(route_coords) // 2
    map_center = (route_coords[mid_idx][1], route_coords[mid_idx][0])

m = folium.Map(location=map_center, zoom_start=zoom_start)

# 건물 폴리곤(회색)
folium.GeoJson(
    buildings_gdf,
    name="Buildings",
    style_function=lambda x: {"color": "gray", "weight": 1, "fillOpacity": 0.2}
).add_to(m)

# 그림자 영역(파란색)
folium.GeoJson(
    shadow_gdf,
    name="Shadows",
    style_function=lambda x: {"color": "blue", "weight": 0, "fillOpacity": 0.5}
).add_to(m)
```

## 6. 결과 시각화 (visualization.py)

plot\_route\_and\_shadow 함수 호출

Folium 라이브러리를 사용해 지도 위에 최적 경로, 건물, 그림자 영역, 출발/도착 지점을 표시하는 HTML 파일 생성.

생성된 result\_map.html 파일을 웹 브라우저에서 자동으로 실행하여 사용자에게 결과를 보여줌.

# 기능 및 구현 설명

## 주요 기능

runner.py: 전체 시스템 실행을 제어하는 메인 스크립트 파일

input\_handler.py: 사용자 입력을 처리하고 위경도 좌표인지 장소명인지 판별하는 파일

osmnx\_route.py: OSMnx와 NetworkX로 도로망 그래프를 만들고, 출도착지에 대해 k개의 최단 경로를 생성하는 파일

osm\_buildings.py: 특정 지역 건물 데이터를 다운로드하고 GeoDataFrame으로 가공하는 파일

sun\_position.py: 특정 시간과 장소에서 태양 고도와 방위각을 계산하는 파일

shadow\_calc.py: 태양 위치와 건물 데이터를 기반으로 그림자 영역을 생성하는 파일

shadow\_analysis.py: 경로와 그림자 영역의 교차를 분석해 그림자 덮임 비율을 도출하는 파일

visualization.py: 분석 결과를 Folium으로 시각화하는 파일

utils.py: 공통으로 사용하는 유틸리티 함수들을 모아둔 파일

# 기능 및 구현 설명

## 주요 사용 기술(라이브러리)

OSMnx: OpenStreetMap의 도로망, 건물 등 지리 데이터를 쉽게 다운로드하고 네트워크 그래프로 변환하는데 사용하는 라이브러리 파일

GeoPandas & Shapely: 지리 정보(점, 선, 면)를 다루기 위한 필수 라이브러리 파일. 건물/그림자 폴리곤, 경로 라인 등의 기하학적 데이터를 생성하고 공간 분석(교차, 버퍼 등)을 수행하는데 사용

NetworkX: 복잡한 네트워크(그래프)를 생성하고 분석하는 라이브러리 파일. OSMnx 내부에서 k-최단 경로를 찾는데 활용

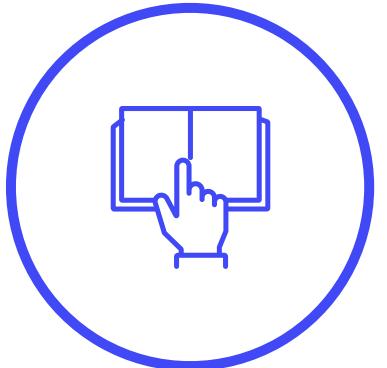
pvlib: 태양광 에너지 분야에서 사용하는 라이브러리 파일. 정확한 태양 위치 계산을 위해 사용

Folium: 지도 시각화 라이브러리 파일. 분석 결과를 웹 기반의 동적 지도로 표현하는데 사용

Geopy: 장소 이름을 위도, 경도 좌표로 변환(지오코딩)하는 기능을 제공하는 라이브러리 파일

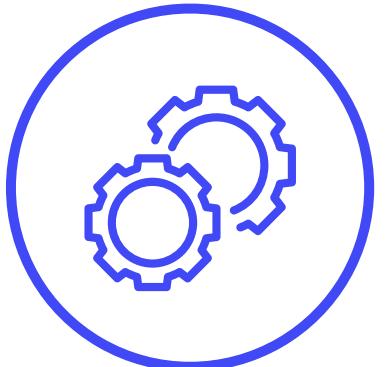
pyproj: 서로 다른 지도 투영법(좌표계) 간의 변환을 지원하는 라이브러리 파일. 본 프로젝트에서는 정확한 거리 계산을 위해 WGS84와 UTM 좌표계를 상호 변환하는데 사용

# 변경 사항 및 개선 포인트



## API 변경

- OSM → 국토부 토지 건물 API → OSM
- 국토부 API: 건물 면적, 높이 정보 우수
- OSM 재사용 이유: 국토부 api 다양한 누락정보 확인



## 기술 변경

- 경로 탐색 방식 개선 (경유지를 설정하여 여러 경로탐색하는 방식 → 여러 경로를 지원하는 osmn 사용)
- 출발자-도착지 기준 사각형내의 건물들 연산 → 경로 상 건물 데이터만 연산
- 불필요한 데이터 처리 감소



## 성능 향상

- 연산 총량 감소로 속도 향상 달성
- 사용자 경험 개선
- 실시간 그림자 경로 제공 가능성 증가

# 구현 결과 및 시연

## 주요 기능 소개

- 실시간 그림자 경로 계산
- 사용자 위치 기반 추천
- 시간대별 최적 경로 제안

## 데모 영상

- 2분 내외 시연 영상
- 실제 사용 시나리오 재현
- 주요 기능 하이라이트

## UI/UX 스크린샷

- 직관적인 사용자 인터페이스
- 그림자 영역 시각화
- 경로 정보 표시

# 구현 결과 및 시연

==== CoolWay1.3 - 그림자 경로 추천 =====

출발지(예: 부산 하단역 or 35.106217, 128.9667238): 부산 하단역

도착지(예: 동아대학교 승학캠퍼스 or 35.113752, 128.965672): 동아대학교 승학캠퍼스

[CoolWay] 최단경로 후보 10개 생성 중...

[CoolWay] OSM 건물 데이터 다운로드 중...

[CoolWay] 태양 위치 계산 중... (2025-06-11 09:46:53.209743+09:00)

태양 고도: 53.97°, 방위각: 98.71°

[CoolWay] 그림자 영역 계산 중...

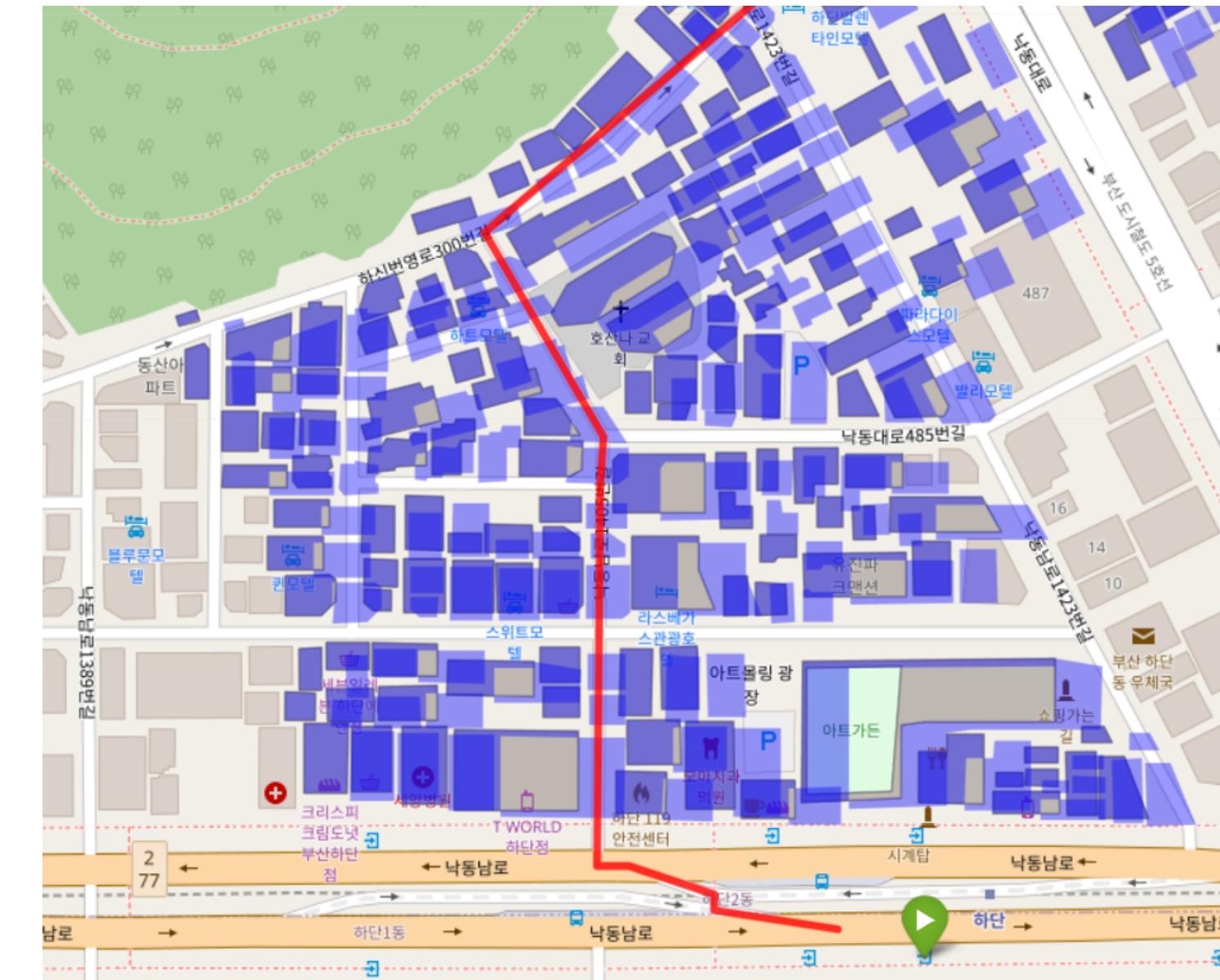
# 구현 결과 및 시연

```
[CoolWay] 최단경로 후보 10개 생성 중...
[CoolWay] OSM 건물 데이터 다운로드 중...
[CoolWay] 태양 위치 계산 중... (2025-06-11 09:46:53.209743+09:00)
태양 고도: 53.97°, 방위각: 98.71°
[CoolWay] 그림자 영역 계산 중...
```

# 구현 결과 및 시연

[CoolWay] 결과 시각화 중...

브라우저에서 `result_map.html`이 자동으로 열렸습니다. (추천: 경로 1)



# 사용자 테스트 및 피드백

## 1 - 도보 시 시원함 체감

- 평소 경로와 차별화된 안내

## 2 - 그림자 가시성 개선 필요

- 지도 상 그림자 표시 과다

## 3 - UI 개선 통해 가시성 향상

- 추천 경로에만 그림자 표시

## 4 - 사용자 경험 지속적 개선

- 계절별 맞춤 기능 추가 검토

# 지속 가능성과 확장 가능성(1)



- 노약자
- 유모차/휠체어
- 반려동물 산책자
- 야외활동자
- 맞춤 경로 추천

- GPS, GIS 표준화
- 그림자 알고리즘
- 도시별 적용
- 스마트시티 연계

- 동남아, 중동
- 햇볕 강한 지역
- 현지화 전략
- 글로벌 서비스

- 실시간 AR 안내
- 그림자 경로 표시
- 가상 도시 탐방
- 현실감 강화

## 지속 가능성과 확장 가능성 (2)

### 다른 경로 탐색 기술과의 결합

COOLWAY는 다양한 경로 탐색 기술과 결합하여 더욱 강력한 서비스로 발전할 수 있습니다. 예를 들어, 안전 경로, 장애물 회피 경로, 미세먼지 적은 길 등과 결합하여 다중 조건 기반의 스마트 경로 탐색이 가능합니다. 이를 통해 사용자의 다양한 편의를 충족시키고, 보행자의 안전과 편의를 크게 향상시킬 수 있습니다.

### API 기반 외부 연동 및 수익 모델

COOLWAY를 외부 서비스와 연동하여 새로운 비즈니스 모델을 창출할 수 있습니다. 네이버 지도나 카카오맵과 같은 대형 플랫폼과의 제휴를 통해 그림자 경로 추천 플러그인을 제공할 수 있으며, 프리미엄 사용자를 대상으로 한 개인화 기능 추가로 유료화도 가능합니다. 또한, 대형 병원, 캠퍼스, 공원, 관광지 등을 대상으로 한 B2B 서비스 제공으로 수익을 다각화할 수 있습니다.

# 프로젝트 회고 및 향후 계획

## 팀원 개인 회고 및 발전 계획

### 김제윤

배운점 : 사업시작은 우리와 동떨어진게 아니라, 기발한 아이디어가 있다면 시작할 수 있다는점을 배웠다.

아쉬웠던 점 : 아이디어를 구현할 지식이 없을때의 막막함이 아쉬웠다. 코딩을 조금더 잘 배워야겠다.

회고 : 다음 팀플에는 초반부터 방향성을 잘 잡고 시작해야겠다.

### 권용현

배운점: 코드 구상을 처음부터 잘 해야 후에 가서 부족한 부분없이 탄탄하게 진행됨을 배웠음

아쉬웠던 점: 평소 공부가 부족하여 만족스러운 결과를 도출해내지 못한 것이 아쉬움

회고: 다음 프로젝트때에는 관련 지식을 공부하여 초반부터 기반을 잘 다져야겠음

### 김도원

배운 점: 여러 API를 사용하면서 많은 지식을 쌓을 수 있었고 미래 목표가 창업이라 창업에 기반이 되는 시간을 가질 수 있었다

아쉬웠던 점: 관련 지식이 부족하여 만족스러운 결과를 도출해내지 못해 아쉬웠음

회고 : 코딩 실력을 더 키워 더욱 제대로된 프로젝트를 수행하여 결과를 만들어내고 싶다.

### 정윤찬

효과적인 역할 분담의 중요성을 이번 프로젝트를 통해 깊이 깨달았는데, 이는 각 팀원이 자신의 강점과 전문성을 최대한 발휘할 수 있도록 역할을 배정함으로써 전체 업무의 효율성과 프로젝트의 완성도가 눈에 띄게 향상된다는 점을 직접 경험했기 때문입니다. 그러나 한편으로는 개발 과정에 집중하다 보니 프로젝트 후반부에 이르러서야 테스트와 사용자 피드백을 본격적으로 수집하게 되어, 최종 결과물의 완성도와 사용자 만족도를 높일 수 있는 소중한 기회를 일부 놓친 점이 아쉬움으로 남았습니다.

### 향후 계획:

- UI 개선으로 그림자 표시 가시성 향상
- Warm Way 기능 추가로 계절별 맞춤 서비스 제공
- 지속적인 사용자 피드백 수렴 및 기능 개선

### 김철호

배운점: 내가 가진 역량을 발휘하여 실제 프로젝트를 끝까지 진행해본 경험이 없어 좋은 경험을 쌓았다 생각함.

아쉬웠던점: 수준을 더욱 높여 이보다 더 완벽하고 좋은 프로젝트를 진행해보고 싶음.

회고: 더 늦기전에 많은 경험을 쌓고 실력을 키워야 할것 같음.

