

mbc아카데미

# TRAVEL AGENT

## -개인 맞춤형 AI 여행 서비스-

TEAM 1조 "서울촌놈들"

이용환/ 이대식/ 길명철/ 조시현/ 이재준/ 강휘준/ 한광원



# 목 차

01

프로젝트 개요

02

프로젝트 팀 구성 및 역할

03

프로젝트 수행 절차 및 방법

04

프로젝트 수행 경과

05

자체 평가 의견

# 01. 프로젝트 개요

- 프로젝트 주제 및 선정 배경, 기획의도:** K-CULTURE 체험을 원하는 외국인에게 서울 시내 **A I 맞춤형 여행 가이드 플 랜** 제공
- 프로젝트 내용 :** MBTI 기반으로 명소·음식·숙소·액티비티를 추천하고 성향이 유사한 사용자 간 **동행 매칭 서비스** 구현
- 활용 프로그램 :** PYTHON / DJANGO / 딥러닝(추천 알고리즘, 시각화 모델 구축) / 오픈API
- 프로젝트 일정 :** 10월 13일(월)~ 11월 7일(금)
- 활용방안 및 기대 효과 :** 한 번의 클릭으로 여행 일정과 동행 매칭을 자동화해 **편리하고 즐거운 서울 여행 가이드** 제공

## 02. 프로젝트 팀 구성 및 역할

훈련생	역할	담당 업무
이용환	팀장	지도, 추천시스템, LLM가이드 [시각화, 지도, 여행 가이드]
이대식	팀원	데이터리, 데이터DB, 웹크롤링 [ 크롤링 데이터화, 설계 구축]
길명철	팀원	데이터 DB, 디자인, 모든 기능 통합 및 오류 확인, 웹크롤링 [데이터DB 통합, 크롤링 데이터화]
조시현	팀원	디자인, PJ계획 수립 [디자인 설계 구축, PJ계획 수립]
이재준	팀원	채팅 시스템, 매칭 시스템, 날씨DB [챗 모델 구축, 날씨DB 데이터구축]
강휘준	팀원	회원가입 시스템 [User DB 구축]
한광원	팀원	데이터 수집, 웹서칭 [여행 관련 데이터 수집, 디자인 수집]

# 03. 프로젝트 수행 절차 및 방법

프로젝트의 사전 기획 과 프로젝트 수행 및 완료 과정으로 이원화 작성

구분	기간	활동	비고
사전 기획	10/13(월) ~ 10/15(수)	프로젝트 기획 및 주제 선정	아이디어 선정
데이터 수집	10/15(수) ~ 10/17(금)	필요 데이터 정의/ 수집	
데이터 전처리	10/20(월) ~ 10/21(화)	데이터베이스 구축을 위한 데이터 정제 및 전처리	<b>팀원 각자 교육 및 TESTING 실행 (최적화 자료 별도 취합)</b>
모델링	10/20(월) ~ 10/28(화)	추천 플랜, 다이어리, 채팅 기능 작업	
디자인/ 서비스 안정화 및 시연	11/3(월) ~ 11/7(금)	장고에서 가능 MERGE 장고 디자인 및 시스템 TEST	최적화, 오류 수정
총 개발기간	10/13(월) ~ 11/7(금)(약 1달 )		

# 04. 프로젝트 데이터 수집 및 장고 데이터베이스 구축

## 데이터 수집

### 1) Google API:

- 구글에서 제공하는 API 를 활용하여 장소 데이터 수집

### 2) NAVER:

- 네이버 검색한 내용을 바탕으로 블로그 본문 크롤링

### 3) 여행 관련 사이트

- 여행 관련 사이트 정보 크롤링

## 장소 성격 LLM 분석

### 1) OpenAI GPT API:

- 수집된 데이터를 기반으로 장소의 성격을 LLM 을 활용하여 분석

### 2) Hugging Face:

- 네이버 블로그 데이터로 LLM 사용

## 데이터 DB

### 1) Django Framework:

- 장소 (숙소, 음식점, 관광 등) 데이터 생성
- 장소의 계절, MBTI, 동행 그룹, 나이, 성별 컬럼의 데이터 생성

## 다국어 데이터

### 1) Django Framework:

- GNU gettext 시스템 사용
- 장고에서 표준 번역 포맷으로 .po 파일로 번역문 관리 .mo 로 컴파일 자동 번역

# 04. 프로젝트 서비스 실행 순서 및 주요 사용 프로그램

## 로그인 및 회원가입

### 1) Django Auth System:

- 사용자 인증 및 세션 관리

### 2) Transaction.atomic:

- 회원가입 시 User·Profile 동시 생성  
(데이터 일관성 보장)

### 3) Messages Framework:

- 로그인·회원가입 성공 시 환영 메시지 출력

## 방문 계획 & 루트 시각화

### 1) Google Gemini API:

- LLM 기반 자연어 생성

### 2) Mapbox API :

- 지도 동선 시각화 구현

### 3) HTML / JavaScript :

- 최종 시각화 OUTPUT

## 파트너 매칭 서비스(동행)

### 1) ASGI\_APPLICATION:

- APP 진입점 비동기 환경 설정

### 2) WebSocket :

- 서로 1대1 통신연결

### 3) HTML / JavaScript :

- 최종 시각화 OUTPUT

## 여행 다이어리

### 1) Django Framework:

- travel\_diary\_detail 뷰를 통해 여행 계획과 실제 기록 통합
- map\_locations\_json 데이터 생성으로 위치·일정별 매핑 & 시각화

### 2) Kakao Maps API:

- 지도 상 위치 데이터 병합 및 마커 필터링

### 3) Vanilla JavaScript:

- 일정별/지도별 뷰 전환, 날짜별 필터링 및 마커 클러스터링

### 4) OpenAI GPT API:

- 다이어리 요약 및 태그 자동 생성 (AI 기반 콘텐츠 요약 기능)

# 05. 프로젝트 수행 경과

## 1. 로그인 및 회원가입 플로우 차트

① 회원가입

기본 계정 생성 → 프로필 생성 → 회원가입 완료 표시

② 로그인

사용자 입력 → 인증 처리 → 세션 생성 → 로그인 성공 메시지 출력 → 메인 화면 표시

③ 닉네임 표시

로그인 후 사용자 정보 조회 → 닉네임 기반 환영 문구 출력

④ 메시지 출력

회원가입·로그인 성공 시 → 메시지 생성 → 화면에 표시 → 일정 시간 후 자동 사라짐

⑤ 로그아웃

로그아웃 요청 → 세션 종료 → 로그아웃 완료 메시지 표시 → 로그인 화면 이동

# 05. 프로젝트 수행 경과

## 1. 로그인 및 회원가입 주요 코드 1

DB 구조설계

```
class UserProfile(models.Model):
    unique_id = models.UUIDField(default=uuid.uuid4, editable=False, unique=True)
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    nickname = models.CharField(max_length=30)
    gender = models.CharField(max_length=10, blank=True)
    age_range = models.CharField(max_length=20, blank=True)
    country = models.CharField(max_length=50, blank=True)
    languages = models.CharField(max_length=100, blank=True)
    travel_style = models.CharField(max_length=50, blank=True)
    budget = models.CharField(max_length=50, blank=True)
    smoking = models.CharField(max_length=20, blank=True)
    drinking = models.CharField(max_length=20, blank=True)
    sns = models.CharField(max_length=100, blank=True)
    bio = models.TextField(blank=True)
    mbti = models.CharField(max_length=4, blank=True, null=True)

    def __str__(self):
        return f'{self.nickname} ({self.user.username})'
```

Admin 설정 등록

Select user profile to change													
Action:	UNIQUE ID	USER	NICKNAME	GENDER	AGE RANGE	COUNTRY	LANGUAGES	TRAVEL STYLE	BUDGET	SMOKING	DRINKING	SNS	MBTI
<input type="checkbox"/>	fc0f636f-fe98-4d5e-ac32-c6893e3d103f	돈건쓰	김동건	남성	40대 이상	대한민국	한국어	미식	중간	흡연	가끔	-	ISTJ
<input type="checkbox"/>	Saa47670-0dc2-4db7-9ce1-f1b94d94cae	광원이	한광원	남성	20대	대한민국	한국어	액티비티	고급	비흡연	즐김	dd	ENTJ
<input type="checkbox"/>	bcb46ffb-9136-4f14-9270-5e569d82e6e3	강휘준	강휘준	남성	20대	대한민국	한국어	휴양형	저예산	비흡연	즐김	dd	ESTP
<input type="checkbox"/>	9c132d6d-9b36-4c4a-8cc9-b5b91f71f4d3	qkqh	바보	남성	20대	대한민국	한국어	문화탐방	저예산	비흡연	즐김	ㄴㄴ	ESTJ
<input type="checkbox"/>	f8cb0307-7d71-4788-92b2-856335f27181	roskfl	개나리	남성	20대	대한민국	한국어	휴양형	저예산	비흡연	즐김	s	ESTJ

관리자 확인

```
@admin.register(UserProfile)
class UserProfileAdmin(admin.ModelAdmin):
    list_display = [
        "unique_id",
        "user",
        "nickname",
        "gender",
        "age_range",
        "country",
        "languages",
        "travel_style",
        "budget",
        "smoking",
        "drinking",
        "sns",
        "mbti"
    ]
```

# 05. 프로젝트 수행 경과

## 1. 로그인 및 회원가입 주요 코드 2

### 입력 및 검증 단계

```
# ===== 회원가입 =====
def signup_view(request):
    print("Signup view called")

    if request.method == "POST":
        email = request.POST.get("email")
        userid = request.POST.get("userid")
        password = request.POST.get("password")
        nickname = request.POST.get("nickname")
        gender = request.POST.get("gender")
        age_range = request.POST.get("age_range")
        country = request.POST.get("country")
        languages = request.POST.get("language") # form 필드 이름과 일치
        travel_style = request.POST.get("travel_style")
        budget = request.POST.get("budget")
        smoking = request.POST.get("smoking")
        drinking = request.POST.get("drinking")
        sns = request.POST.get("sns")
        bio = request.POST.get("bio")
        mbti = request.POST.get("mbti")

        # 1. username 중복 체크
        if User.objects.filter(username=userid).exists():
            return render(request, f"{userid}", {"error": "이미 가입된 아이디입니다.."})


# ===== 회원가입 =====
```

### 계정 생성 및 프로필 저장 단계

```
try:
    # 2. User 객체 생성 (이때 Signal이 UserProfile 객체를 자동 생성함)
    # 트랜잭션을 사용하여 User 생성 실패 시 UserProfile 생성도 롤백
    with transaction.atomic():
        user = User.objects.create_user(username=userid, email=email, password=password)
        print("===== 지금 (User 생성 완료) =====")

    # 3. 자동으로 생성된 UserProfile 객체를 가져와서 업데이트
    # Signal이 작동하지 않는 경우를 대비해 get() 대신 filter().first()를 쓰거나,
    # Signal이 확실하다면 user.userprofile (또는 user.profile)를 바로 사용합니다.

    # Note: 'userprofile'은 UserProfile 모델이 User와 연결된 기본 이름입니다.
    profile = user.userprofile

    # 4. 폼에서 받은 데이터로 profile 객체 업데이트
    profile.nickname = nickname
    profile.gender = gender
    profile.age_range = age_range
    profile.country = country
    profile.languages = languages
    profile.travel_style = travel_style
    profile.budget = budget
    profile.smoking = smoking
    profile.drinking = drinking
    profile.sns = sns
    profile.bio = bio
    profile.mbt = mbti

    profile.save() # UserProfile 객체 저장
```

# 05. 프로젝트 수행 경과

## 1. 로그인 및 회원가입 구현

The image shows a sequence of four screenshots illustrating the user interface for account creation and login.

- Screenshot 1: Login Page (Korean)**  
The page is titled "로그인" (Login). It features two input fields: "아이디" (ID) and "비밀번호" (Password), both with placeholder text "아이디를 입력하세요" (Please enter ID) and "비밀번호를 입력하세요" (Please enter password). A large blue "로그인" (Login) button is at the bottom. At the bottom right, there are links for "회원가입" (Sign Up) and "비밀번호 찾기" (Forgot Password). The top navigation bar includes language options: "ko 한국어" (highlighted with a red dashed box), "us English", and "es Español".
- Screenshot 2: Registration Form (Korean)**  
The page is titled "회원가입" (Sign Up). It contains numerous input fields for personal information: Full Name / Nickname, Username, Password, Confirm Password, Email, Gender, Age Group, Country, Languages Spoken, Travel Style, Budget Level, Smoking Preference, Drinking Preference, MBTI Type, SNS (optional), and About Me. Each field has its corresponding English label above it. A large blue "계속하기" (Next) button is at the bottom. The top navigation bar includes language options: "ko 한국어" (highlighted with a red dashed box), "us English", and "es Español".
- Screenshot 3: Registration Form (English)**  
The page is titled "Create Your Travel Account". It has identical fields to the Korean version but with English labels. A large blue "Sign Up" button is at the bottom. The top navigation bar includes language options: "ko 한국어" (highlighted with a red dashed box), "us English", and "es Español".
- Screenshot 4: Registration Form (Spanish)**  
The page is titled "Registro de viaje". It has identical fields to the English version but with Spanish labels. A large blue "Registrarse" button is at the bottom. The top navigation bar includes language options: "ko 한국어" (highlighted with a red dashed box), "us English", and "es Español".

# 05. 프로젝트 수행 경과

## 2. 방문 계획&루트 시각화 플로우 차트

### 사용자 여행정보 Parsing

```

7 def parse_user_request(request) -> Dict[str, Any]:
40     # 테마들
41     themes = request.POST.getlist("tema")
42
43     # 새 날짜 설정
44     start_date = request.POST.get("start_date", "").strip()
45     end_date = request.POST.get("end_date", "").strip()
46
47     # 여행 일수 계산
48     # start_date, end_date가 YYYY-MM-DD 들어온다고 가정
49     # 둘 다 있으면 실제 날짜 차이로 total_days / nights 계산
50     nights = 1
51     total_days = 2
52     if start_date and end_date:
53         try:
54             from datetime import datetime
55             fmt = "%Y-%m-%d"
56             d0 = datetime.strptime(start_date, fmt)
57             d1 = datetime.strptime(end_date, fmt)
58             delta_days = (d1 - d0).days + 1 # 예: 10/01~10/03 -> 3일
59             if delta_days < 1:
60                 delta_days = 1
61             total_days = delta_days
62             nights = max(0, delta_days - 1)
63         except Exception:
64             pass # 파싱 실패하면 기본값 유지
65
66     # 동행/분위기 추정 (임시 로직은 기준 그대로)
67     group = "couple" if ("커플" in raw_text or "데이트" in raw_text) else "friends"
68
69     mbti_guess = "ENFP"
70     season = "autumn"
71
72     # 영문 구 코드 -> 한글 구 이름 변환
73     DISTRICT_MAP = {
74         "gangnam": "강남구",
75         "seocho": "서초구",
76         "songpa": "송파구",
77         "gangdong": "강동구",
78         "yongsan": "용산구",
79         "mapo": "마포구",
80         "jongno": "종로구",
81         "jung": "중구",
82         "seongdong": "성동구",
83         "gwangjin": "광진구",
84     }

```

### 자동 일정 생성

```

def split_into_days(ranked: List[Dict[str, Any]], total_days: int) -> List[List[Dict[str, Any]]]:
    """
    요구사항 기반 일정 생성 로직

    규칙 요약:
    - total_days = nights + 1
    - nights == 0 (당일치기): 숙소 제외. 그냥 상위 장소들을 순서대로 잘라 일수만큼 분배.
    - nights >= 1:
        - Day1 첫 장소 = 숙소 1곳 (category에 "accommod" 가 포함된 Place)
        - 나머지 장소들은 음식/관광 교차: restaurant -> attraction -> restaurant -> ...
        - Day2 이후도 교차 패턴 계속 이어짐 (숙소는 더 안 나옴)
        - 각 Day에는 가능한 균등하게 분배하되, 최소 1곳은 들어가게.
    """

    # 안정성 가드
    if total_days <= 0:
        total_days = 1

    # 내부 헬퍼: 카테고리 판별 (소문자 비교)
    def is_accommodation(p: Any) -> bool:
        cat = (p.category or "").lower()
        return "accommodation" in cat or "accommodations" in cat

    def is_food(p: Any) -> bool:
        cat = (p.category or "").lower()
        return "restaurant" in cat or "restaurants" in cat

    def is_attraction(p: Any) -> bool:
        cat = (p.category or "").lower()
        return "attraction" in cat or "attractions" in cat

    # ranked = [{"place": Place, "analysis": PlaceAnalysis, "score": float}, ...]
    accommodations = []
    foods = []
    attractions = []
    etcs = [] # 혹시 위 세 분류에 안 들어간 것들 fallback

    for item in ranked:
        place = item["place"]
        if is_accommodation(place):
            accommodations.append(item)

```

# 05. 프로젝트 수행 경과

## 2. 방문 계획&루트 시각화 구현

여행 날짜 선택

여행 날짜

출발일부터 마지막 날까지 선택해주세요

날짜를 선택하세요

날짜를 선택하세요

여행지 선택



여행 그룹 과 여행 테마 선택

여행 그룹  
여행의 동행이 있는지 알려주세요

혼자 친구 가족 연인

여행 테마  
최대 4개까지 선택 해주세요

카페 맛집 축제 자연/산책  
문화/전시 테마파크 힐링/스파 쇼핑

AI 추천 코스

# 05. 프로젝트 수행 경과

## 2. 방문 계획&루트 시각화 구현

Day별 스케줄링 구현

이 플랜으로 저장하기

Day 1

1 K-Grand Hostel Gangnam (accommodations)  
85m  
힐링/휴식, 익스터민/액티비티, 역사/문화탐방, SNS/핫플레이스, 미식/맛집투어 - 커풀선호 50 · 가을매력 95  
대한민국 서울특별시 강남구 논현동 강남대로11길 11

2 것텐스시 강남점 (restaurants)  
3 km  
힐링/휴식, 익스터민/액티비티, 역사/문화탐방, SNS/핫플레이스, 미식/맛집투어 - 커풀선호 85 · 가을매력 90  
대한민국 서울특별시 강남구 테헤란로 109

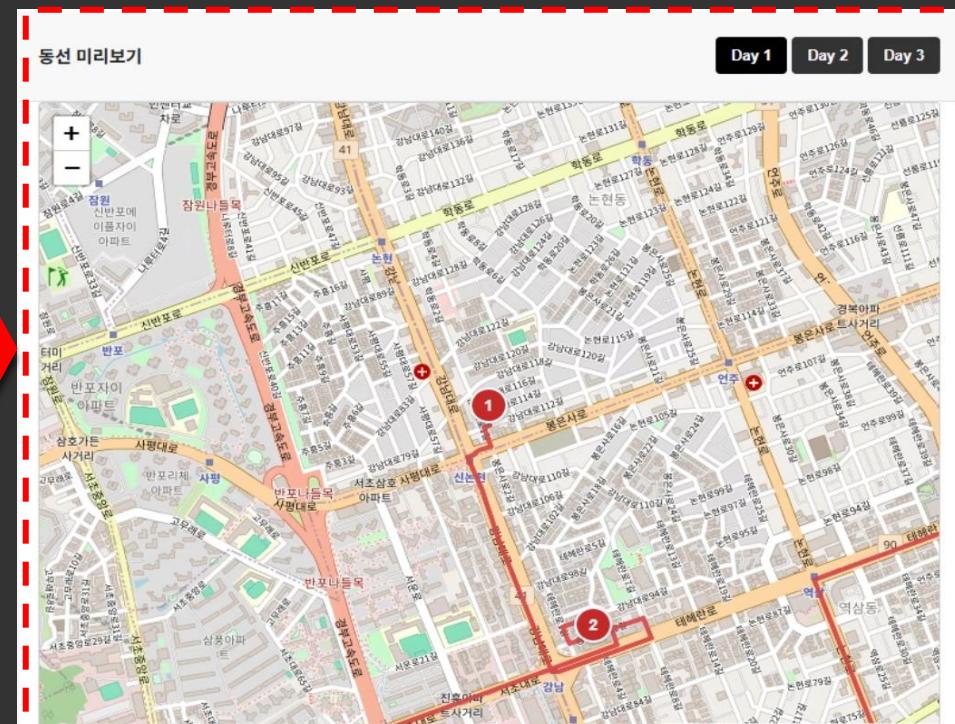
3 뷔페공원 (attractions)  
4 km  
힐링/휴식, 가족, SNS/핫플레이스, 미식/맛집투어, 익스터민/액티비티 - 커풀선호 60 · 가을매력 85  
대한민국 서울특별시 서초구 방배4동

4 양재역맛집 포하야 베트남쌈국수 서초도곡강남점 (restaurants)  
304 m  
힐링/휴식, 익스터민/액티비티, 역사/문화탐방, SNS/핫플레이스, 미식/맛집투어 - 커풀선호 70 · 가을매력 80

동선 미리보기

Day 1 Day 2 Day 3

Map 이동경로 구현



일자 별 여행가이드 제공

Day 2

일일 시간표

08:00 - 08:30 | 기상 & 준비 (숙소) | 가볍게 씻고 옷 착용  
08:30 - 09:30 | 체크아웃 & 이동 (HHZib -> 연트럴파크) | 숙소 정리 후 활기찬 연남동으로  
09:30 - 11:30 | 관광 (연트럴파크) | 자유롭게 산책하며 연남동 감성 즐기기  
11:30 - 12:10 | 이동 (연트럴파크 -> 갓텐스시 강남점) | 신선한 스시를 맛보러 강남으로  
12:10 - 13:20 | 점심식사 (갓텐스시 강남점) | 활기찬 분위기에서 즐기는 맛있는 점심  
13:20 - 13:50 | 이동 (갓텐스시 강남점 -> 올림픽공원) | 가을 경치를 즐기러 공원으로  
13:50 - 16:20 | 관광 (올림픽공원) | 드넓은 공원에서 가을 단풍 만끽하며 산책  
16:20 - 17:20 | 자유시간 및 휴식 (올림픽공원 주변) | 잠시 쉬어가며 친구와 담소 나누기  
17:20 - 18:20 | 이동 (올림픽공원 -> 강촌숯불닭갈비 본점) | 저녁 식사를 위해 구로동으로 이동  
18:20 - 19:40 | 저녁식사 (강촌숯불닭갈비 본점) | 숯불 닭갈비로 여행의 대미를 장식  
19:40 - 20:40 | 귀가 준비 & 이동 | 아쉬움을 뒤로하고 집으로

상세 가이드

● 아침  
⌚ 08:30-11:30 (총 3시간 머무르기)  
😊 체크아웃 후 트렌디한 거리에서 친구와 함께 인증샷 남기기 좋은 코스  
⚠ 대중교통으로 이동 편리, 아침 산책과 브런치를 동시에 해결

● 점심  
⌚ 12:10-13:20 (총 1시간 10분 머무르기)  
😊 신선하고 다채로운 스시로 여행 중 입맛을 돋우는 시간  
⚠ 변화가에 위치하여 접근성 좋음, 식사 후 주변 구경 가능

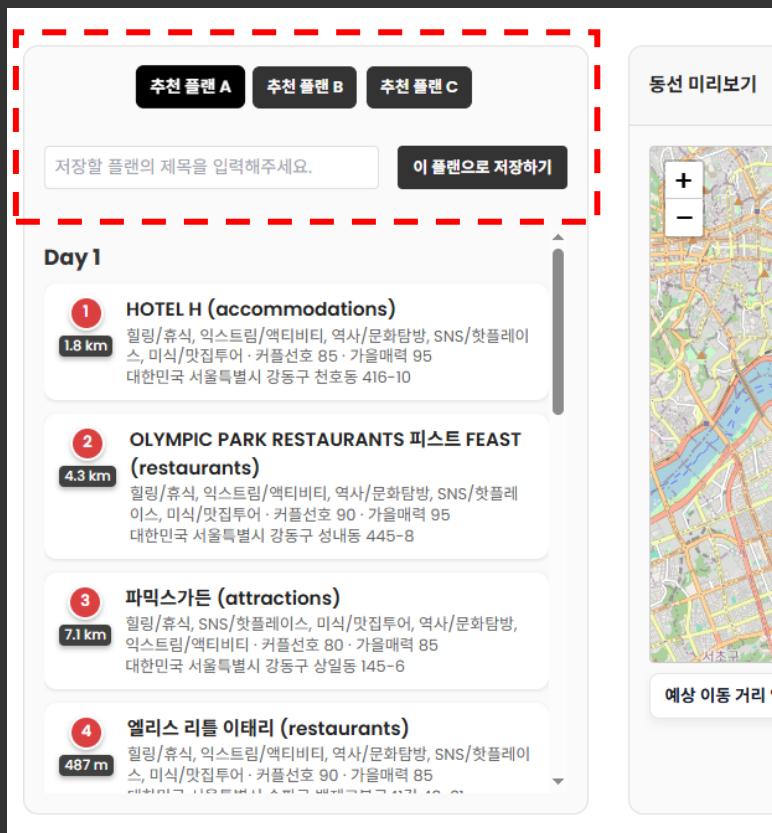
● 오후  
⌚ 13:50-16:20 (총 2시간 30분 머무르기)  
😊 넓은 공원에서 가을을 온몸으로 느끼며 친구와 액티비티 즐기기  
⚠ 가을 단풍이 절정인 시기, 산책하기 좋고 사진 찍을 곳 많음

● 저녁  
⌚ 18:20-19:40 (총 1시간 20분 머무르기)  
😊 숯불 향 가득한 닭갈비와 함께 여행의 마지막 밤을 불태우는 시간

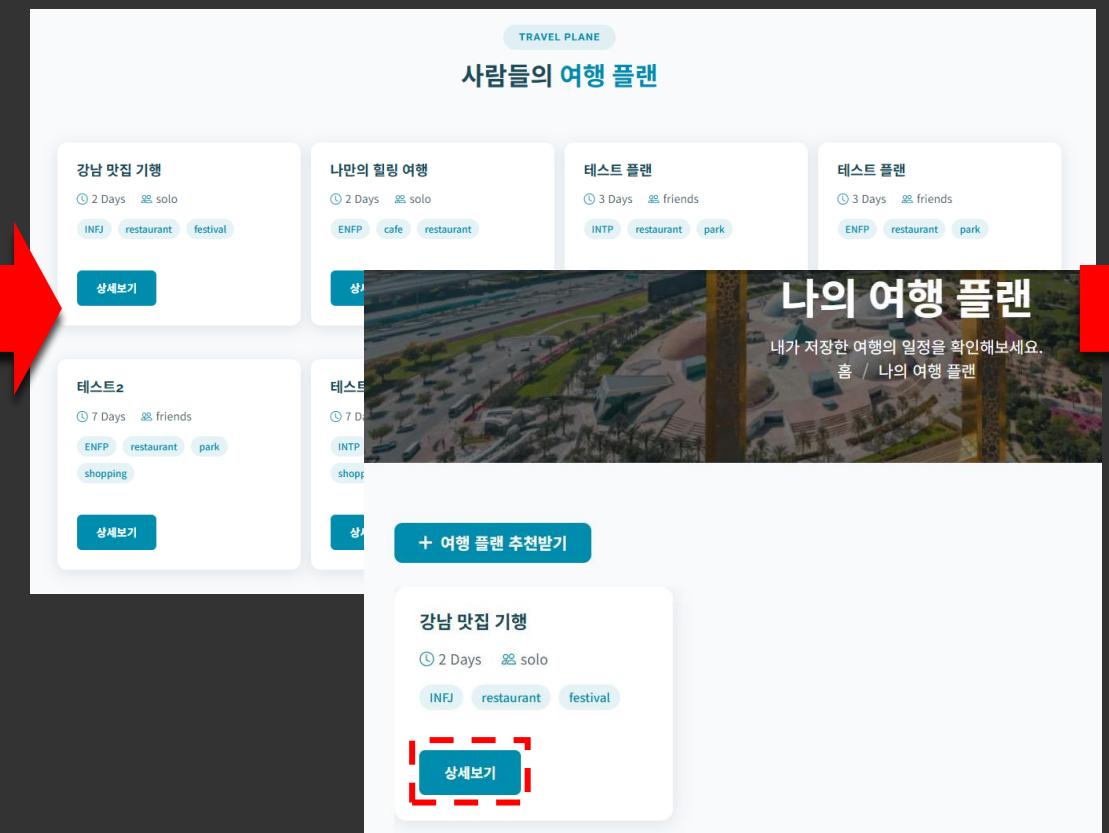
# 05. 프로젝트 수행 경과

## 2. 저장된 플랜 공유 및 확인

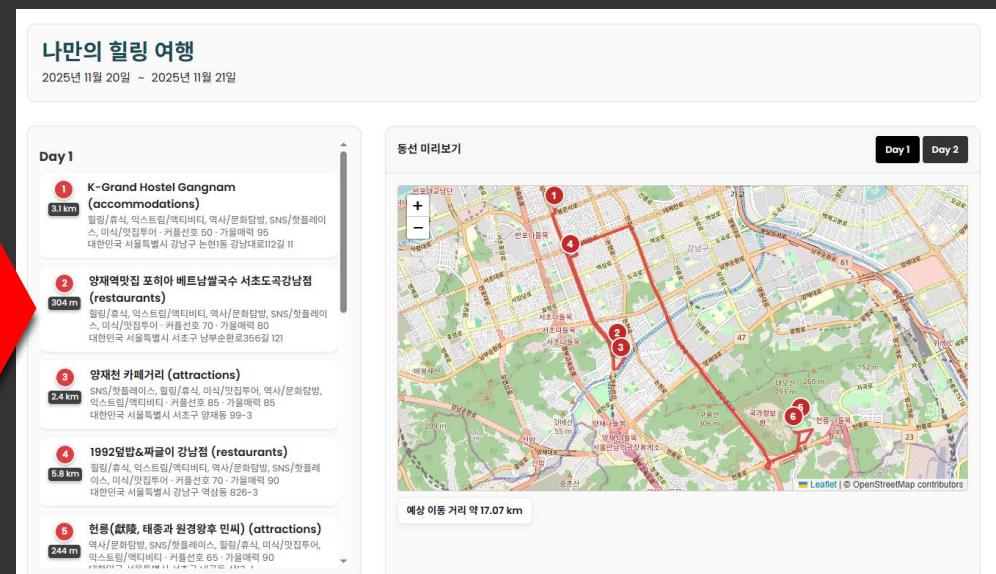
원하는 추천 플랜 저장



저장된 여행 플랜을 공유



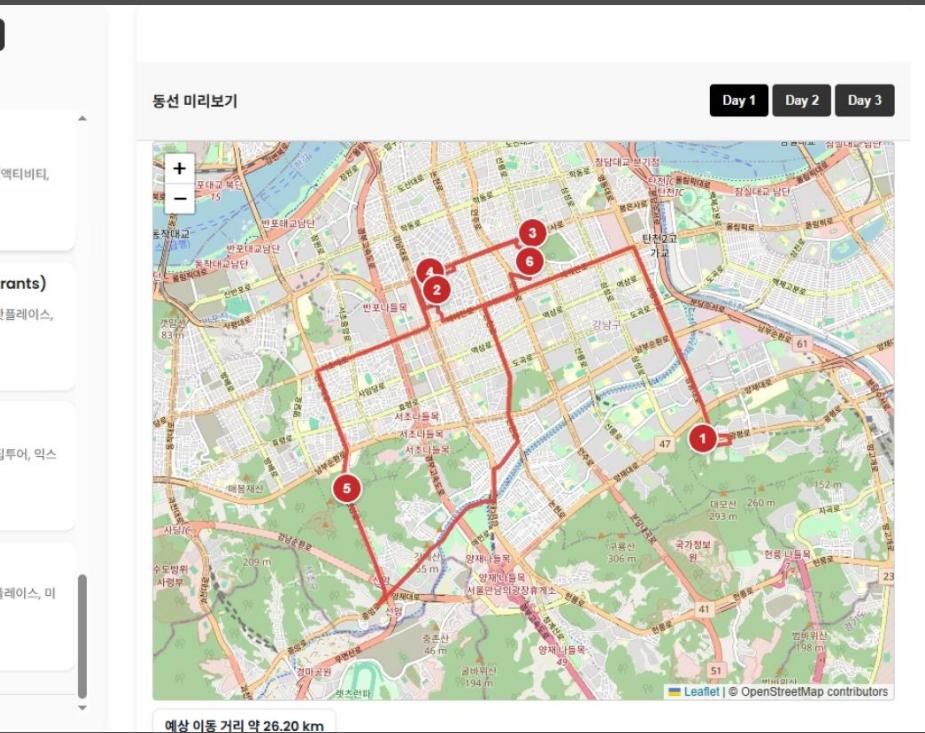
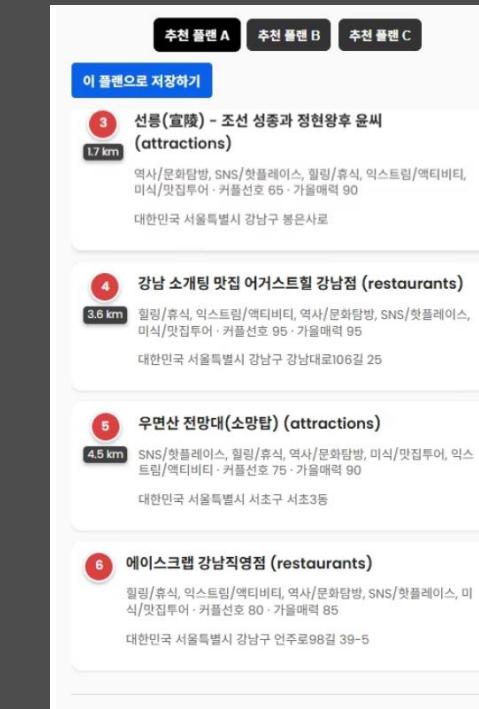
추천 받은 내용을 다시 확인



# 05. 프로젝트 수행 경과

## 2. 방문 계획&루트 시각화 피드백 및 적용

### 동선 중복 RISK



# 05. 프로젝트 수행 경과

## 3. 파트너 매칭 서비스 플로우 차트

1. 자동 매칭 및 방 생성
2. 채팅방 뷰
3. WebSocket 연결 및 그룹화
4. 메시지 송수신 및 저장
5. 메시지 신고 및 관리

### 단계별 과정

여행 계획 등록 → 1:1 매칭 상대 탐색 → 채팅방 생성  
매칭된 방 진입 → 파트너 정보 추출 → 채팅 화면 렌더링  
클라이언트JS 호출 → 서버 연결 → Room Group 참여  
메시지 입력 → 서버 전송 → 비동기 DB 저장 → 실시간 방송  
부적절 메시지 신고(AJAX) → 운영기록 → 관리자 모니터링

# 05. 프로젝트 수행 경과

## 3. 파트너 매칭 서비스 주요 코드1

### 실시간 채팅환경 구성

```
from django.urls import re_path
from . import consumers

websocket_urlpatterns = [
    re_path(r'ws/chat/(?P<room_id>\d+)/$', consumers.ChatConsumer.as_asgi()),
]

import json, datetime, traceback
from channels.generic.websocket import AsyncWebSocketConsumer
from channels.db import database_sync_to_async
from .models import ChatRoom, ChatMessage

class ChatConsumer(AsyncWebSocketConsumer):
    async def connect(self):
        try:
            self.room_id = int(self.scope['url_route']['kwargs']['room_id'])
            self.room_group_name = f"chat_{self.room_id}"
            await self.channel_layer.group_add(self.room_group_name, self.channel_name)
            await self.accept()
            print(f"[CONNECT] channel {self.channel_name} joined {self.room_group_name}")
        except Exception as e:
            print("[ERROR][connect]", e)
            traceback.print_exc()
            await self.close(code=1011)
```

### 채팅 기능의 데이터 모델링

```
# ----- 채팅 관련 모델 -----
class ChatRoom(models.Model):
    room_name = models.CharField(max_length=100, unique=True)
    participants = models.ManyToManyField(User, related_name='chatrooms')
    travel_plan1 = models.ForeignKey(TravelPlan, on_delete=models.CASCADE, related_name='chatrooms_as_plan1')
    travel_plan2 = models.ForeignKey(TravelPlan, on_delete=models.CASCADE, related_name='chatrooms_as_plan2')
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"ChatRoom({self.id}): {self.travel_plan1} <-> {self.travel_plan2}"

class ChatMessage(models.Model):
    room = models.ForeignKey(ChatRoom, on_delete=models.CASCADE, related_name='messages') # ✅ 추가
    sender = models.ForeignKey(User, on_delete=models.CASCADE)
    message = models.TextField()
    timestamp = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.sender.username} -> {self.room.room_name}"

class ChatRequest(models.Model):
    STATUS_CHOICES = [
        ('pending', '대기중'),
        ('accepted', '수락'),
        ('rejected', '거절'),
    ]
    sender = models.ForeignKey(User, on_delete=models.CASCADE, related_name='sent_requests')
    receiver = models.ForeignKey(User, on_delete=models.CASCADE, related_name='received_requests')
    status = models.CharField(max_length=10, choices=STATUS_CHOICES, default='pending')
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.sender.username} -> {self.receiver.username} [{self.status}]"

# ----- 채팅 신고 모델 -----
class ChatReport(models.Model):
    reporter = models.ForeignKey(User, on_delete=models.CASCADE, related_name='reports_made')
    message = models.ForeignKey(ChatMessage, on_delete=models.CASCADE, related_name='reports')
    reason = models.TextField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.reporter} -> {self.message.id}"
```

# 05. 프로젝트 수행 경과

## 3. 파트너 매칭 서비스 주요 코드2

### 자동 매칭 알고리즘

```
def dates_overlap(start1, end1, start2, end2):
    """날짜 범위가 겹치는지 체크"""
    return max(start1, start2) <= min(end1, end2)

def auto_match_and_create_room(new_plan: TravelPlan):
    """자동 매칭: 목적지 + 날짜 겹침 + 방 없는 사용자"""

    # 1. 매칭 후보군 검색 (자신 제외, 같은 목적지)
    candidates = TravelPlan.objects.filter(
        location_city=new_plan.location_city # models.py에서 location_city 필드 사용
    ).exclude(user=new_plan.user) # 같은 사용자 제외

    created_rooms = []

    for candidate_plan in candidates:
        # 2. 날짜 겹치는지 확인
        if dates_overlap(
            new_plan.start_date, new_plan.end_date,
            candidate_plan.start_date, candidate_plan.end_date
        ):
            user1 = new_plan.user
            user2 = candidate_plan.user

            # 3. 중복 채팅방 확인 (이미 두 사용자 간 채팅방이 있는지 체크)
            # participants 필드에 user1과 user2가 모두 포함된 방을 찾음
            if ChatRoom.objects.filter(
                Q(travel_plan1_user=user1, travel_plan2_user=user2) |
                Q(travel_plan1_user=user2, travel_plan2_user=user1)
            ).exists():
                continue # 이미 방이 있으면 다음 후보로 넘어감

            # 4. 새로운 채팅방 생성 및 필드 채우기
            room_name = f"Chat_{user1.username}_vs_{user2.username}"

            # ChatRoom 생성 시 travel_plan1, travel_plan2를 모두 지정
            room = ChatRoom.objects.create(
                room_name=room_name,
                travel_plan1=new_plan,
                travel_plan2=candidate_plan
            )

            # participants ManyToManyField에 사용자 추가
            room.participants.add(user1, user2)
            created_rooms.append(room)

    return created_rooms # 생성된 방 목록을 반환하도록 변경
```

### 웹소켓으로 채팅방에 입장시키는 과정

```
// WebSocket 연결
function connectWebSocket(roomId){
    if(chatSocket) chatSocket.close();
    const protocol = location.protocol==='https'? 'wss': 'ws';
    const wsURL = `${protocol}//${location.host}/ws/chat/${roomId}`;
    chatSocket = new WebSocket(wsURL);

    chatSocket.onopen = ()=> displaySystem(`채팅방 ${roomId} 접속`);
    chatSocket.onmessage = e=>{
        const data = JSON.parse(e.data);
        displayMessage(data.message, data.username, data.timestamp);
    };
    chatSocket.onclose = e=> displaySystem(`채팅 종료 (code: ${e.code})`);
    chatSocket.onerror = e=> displaySystem(`채팅 에러 발생`);
}

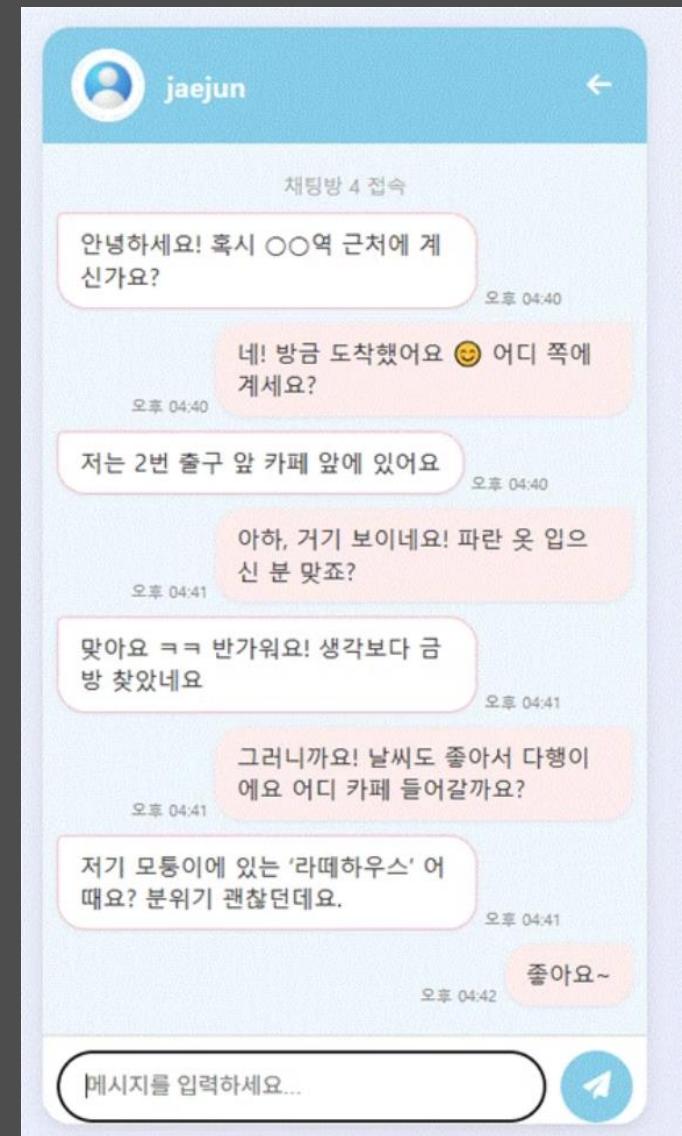
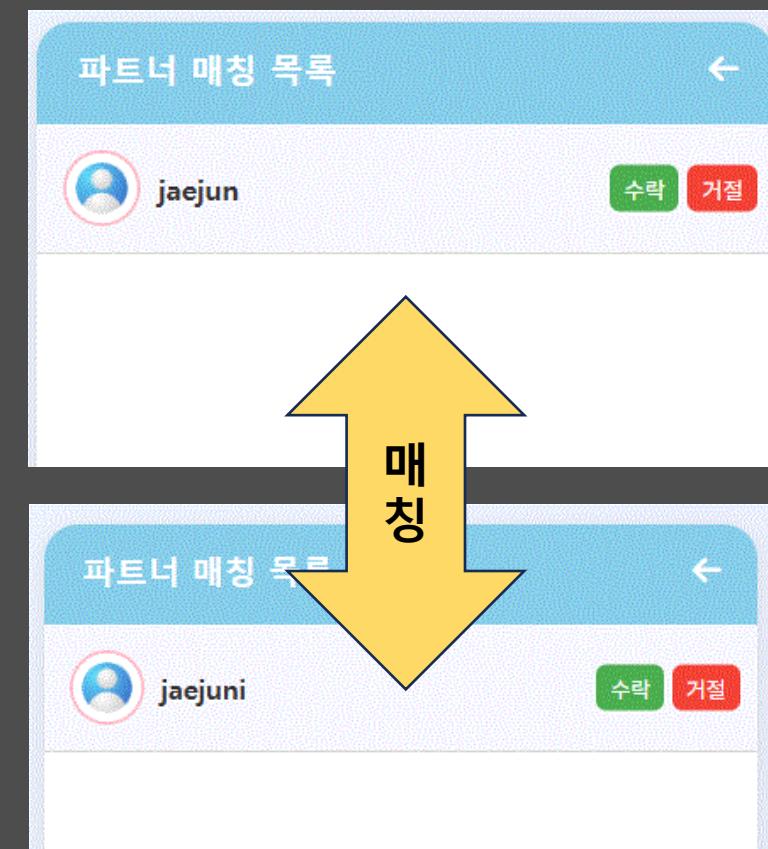
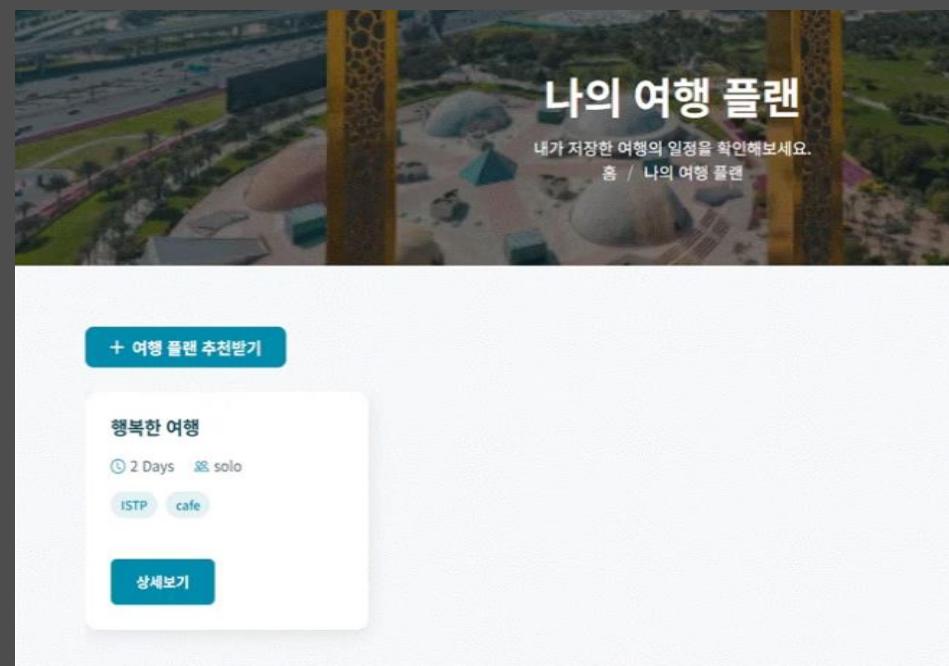
// 메시지 전송
function sendMessage(){
    if(!msgInput.value.trim() || !chatSocket || chatSocket.readyState!=WebSocket.OPEN) return;
    chatSocket.send(JSON.stringify({'message':msgInput.value,'sender':MY_USERNAME,'user_id':MY_USER_ID,'room_id':currentRoom}));
    msgInput.value='';
}

import json, datetime, traceback
from channels.generic.websocket import AsyncWebsocketConsumer
from channels.db import database_sync_to_async
from .models import ChatRoom, ChatMessage

class ChatConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        try:
            self.room_id = int(self.scope['url_route']['kwargs']['room_id'])
            self.room_group_name = f"chat_{self.room_id}"
            await self.channel_layer.group_add(self.room_group_name, self.channel_name)
            await self.accept()
            print(f"[CONNECT] channel {self.channel_name} joined {self.room_group_name}")
        except Exception as e:
            print("[ERROR][connect]", e)
            traceback.print_exc()
        await self.close(code=1011)
```

# 05. 프로젝트 수행 경과

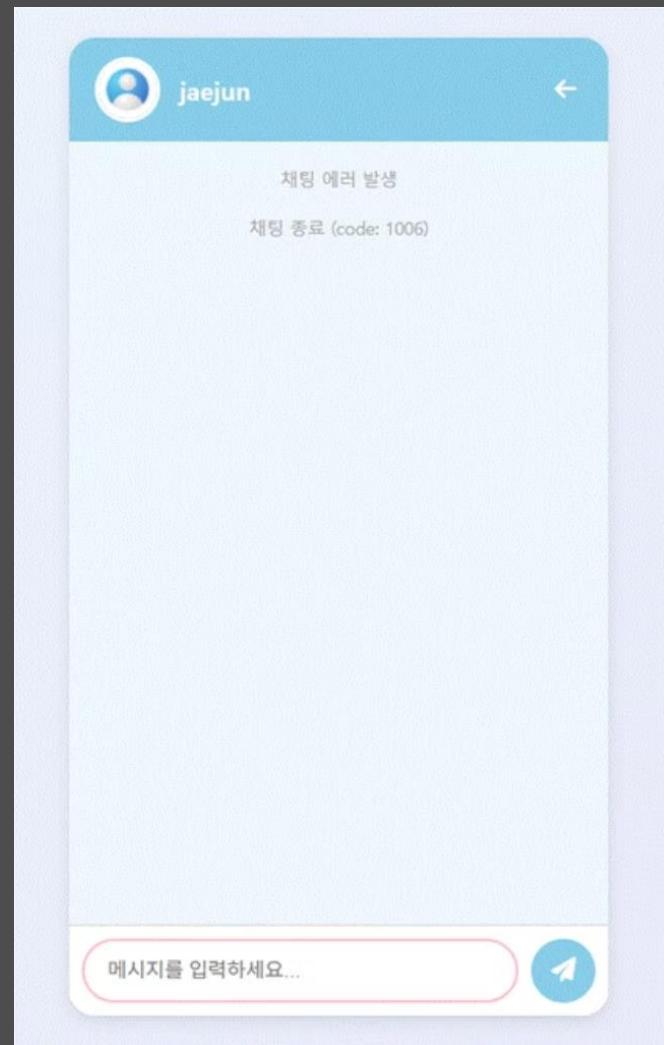
## 3. 파트너 매칭 서비스 프로세스 시작화 구현



# 05. 프로젝트 수행 경과

## 3. 파트너 매칭 서비스 피드백 반영

일반 Django 서버 환경의 한계

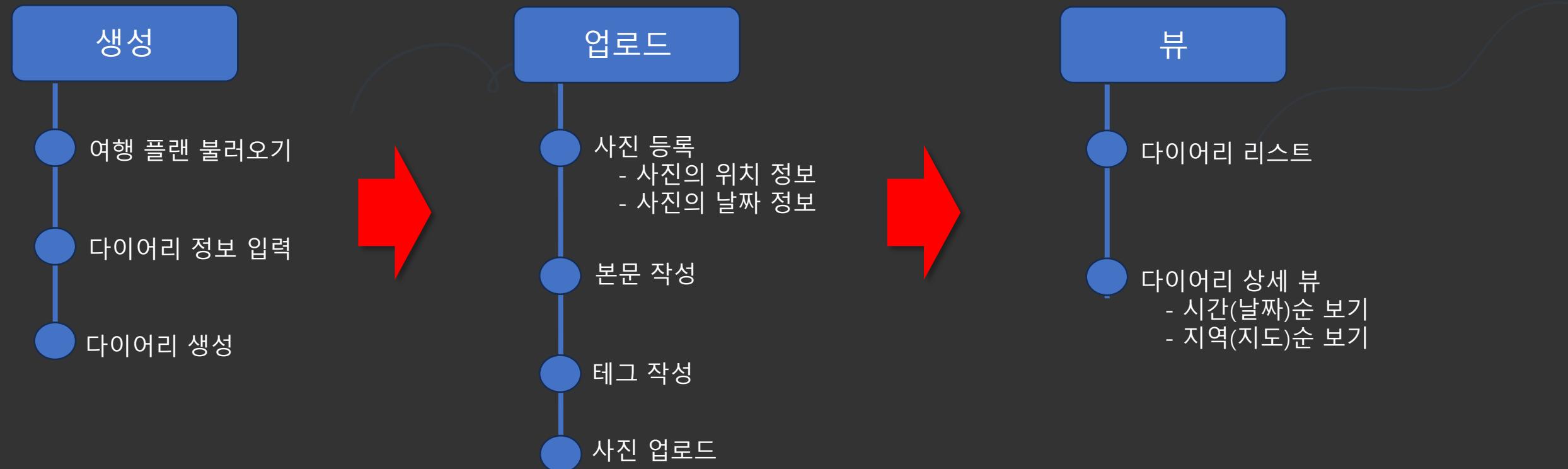


Daphne 서버 및 ASGI 채택



# 05. 프로젝트 수행 경과

## 4. 여행 다이어리 플로우차트



# 05. 프로젝트 수행 경과

## 4. 여행 다이어리 플로우차트

- ① 다이어리 생성
- ② 다이어리 상세 보기
- ③ AI 요약 보기
- ④ 다이어리 수정/ 삭제
- ⑤ 지도 보기

여행 계획 기반으로 새 다이어리 생성  
사진 · 영상 업로드 및 위치 자동 인식  
여행 기록 요약 및 코멘트 생성  
항목 편집, 태그 변경, 삭제 가능  
표시 및 마커/ 클러스터 클릭으로 위치 확인

# 05. 프로젝트 수행 경과

## 4. 여행 다이어리 주요코드1

### 여행 플랜 기반 다이어리 자동 생성

```
@login_required
def create_diary_from_plan(request, plan_id):
    user_plan = get_object_or_404(UserSelectedPlan, id=plan_id, user=request.user)
    travel_plan = user_plan.plan

    # 기존 다이어리 존재 여부 확인
    existing_diary = Travel.objects.filter(plan=travel_plan, author_id=request.user.id).first()
    if existing_diary:
        messages.info(request, "이 플랜에 대한 다이어리가 이미 존재합니다.")
        return redirect('travel:travel_diary_detail', pk=existing_diary.pk)

    # 새 다이어리 생성
    new_diary = Travel.objects.create(
        plan=travel_plan,
        name=travel_plan.title,
        start_date=user_plan.start_date,
        end_date=user_plan.end_date,
        author_id=request.user.id
    )

    # 플랜의 각 장소를 DiaryEntry로 자동 생성
    if 'day_plans' in travel_plan.data:
        for day_index, day_plan in enumerate(travel_plan.data['day_plans']):
            current_date = user_plan.start_date + timedelta(days=day_index)
            for place_data in day_plan:
                DiaryEntry.objects.create(
                    diary=new_diary,
                    author_id=request.user.id,
                    location=place_data.get('name'),
                    timestamp=datetime.combine(current_date, datetime.min.time()).replace(hour=12),
                    latitude=place_data.get('lat'),
                    longitude=place_data.get('lng'),
                )
    )

    messages.success(request, "여행 플랜에서 다이어리를 성공적으로 생성했습니다.")
    return redirect('travel:travel_diary_detail', pk=new_diary.pk)
```

# 05. 프로젝트 수행 경과

## 4. 여행 다이어리 주요코드2

### 다이어리 상세 보기 뷰

```
@login_required
def travel_diary_detail(request, pk):
    # travel_diary = get_object_or_404(Travel, pk=pk, author=request.user)
    travel_diary = get_object_or_404(Travel, pk=pk, author_id=request.user.id)
    diary_entries = travel_diary.diary_entries.all().order_by('timestamp')

    # 날짜별로 항목 그룹화
    grouped_entries = defaultdict(list)
    for entry in diary_entries:
        if entry.timestamp:
            grouped_entries[entry.timestamp.date()].append(entry)
    sorted_dates = sorted(grouped_entries.keys())
    entries_by_date = [(date, grouped_entries[date]) for date in sorted_dates]

    # JavaScript 지도 표시를 위한 데이터 직렬화
    diary_entries_data = []
    for entry in diary_entries:
        diary_entries_data.append({
            'id': entry.id,
            'location': entry.location,
            'timestamp': entry.timestamp.strftime("%Y년 %m월 %d일 %H시 %M분") if entry.timestamp else '',
            'latitude': entry.latitude,
            'longitude': entry.longitude,
            # 'photo_url': entry.photo.url if entry.photo else '',
            # 'comment': entry.comment
            'media_url': entry.media_file.url if entry.media_file else '',
            'media_type': entry.media_type,
            'comment': entry.comment,
            'tags': [tag.name for tag in entry.tags.all()]
        })
    diary_entries_json = json.dumps(diary_entries_data)
```

```
return render(request, 'travel/travel_diary_detail.html', {
    'travel_diary': travel_diary,
    'travel_plan': travel_diary.plan, # Pass the plan to the template
    'entries_by_date': entries_by_date,
    'diary_entries_json': diary_entries_json,
})
```

# 05. 프로젝트 수행 경과

## 4. 여행 다이어리 주요코드3

### 다이어리 뷰 전환 및 지도 표시

```
// 뷰에서 전달받은 다이어리 항목 JSON 데이터
const diaryEntries = JSON.parse(document.getElementById('diary-entries-data').textContent);

function showTimeSortView() {
    timeSortView.classList.remove('hidden');
    locationSortView.classList.add('hidden');
    timeSortBtn.classList.add('bg-amber-500', 'hover:bg-amber-600');
    timeSortBtn.classList.remove('bg-slate-500', 'hover:bg-slate-600');
    locationSortBtn.classList.add('bg-slate-500', 'hover:bg-slate-600');
    locationSortBtn.classList.remove('bg-amber-500', 'hover:bg-amber-600');
}

function showLocationSortView() {
    timeSortView.classList.add('hidden');
    locationSortView.classList.remove('hidden');
    locationSortBtn.classList.add('bg-amber-500', 'hover:bg-amber-600');
    locationSortBtn.classList.remove('bg-slate-500', 'hover:bg-slate-600');
    timeSortBtn.classList.add('bg-slate-500', 'hover:bg-slate-600');
    timeSortBtn.classList.remove('bg-amber-500', 'hover:bg-amber-600');
}

// 위치별로 다이어리 항목 그룹화
const locations = {};
diaryEntries.forEach(entry => {
    if (entry.latitude !== null && entry.longitude !== null) {
        const key = `${entry.latitude},${entry.longitude}`;
        if (!locations[key]) {
            locations[key] = { lat: entry.latitude, lon: entry.longitude, name: entry.location, entries: [] };
        }
        locations[key].entries.push(entry);
    }
});

for (const key in locations) {
    const locationData = locations[key];
    const position = new kakao.maps.LatLng(locationData.lat, locationData.lon);
    const marker = new kakao.maps.Marker({
        position: position,
        title: locationData.name
    });
}
```

# 05. 프로젝트 수행 경과

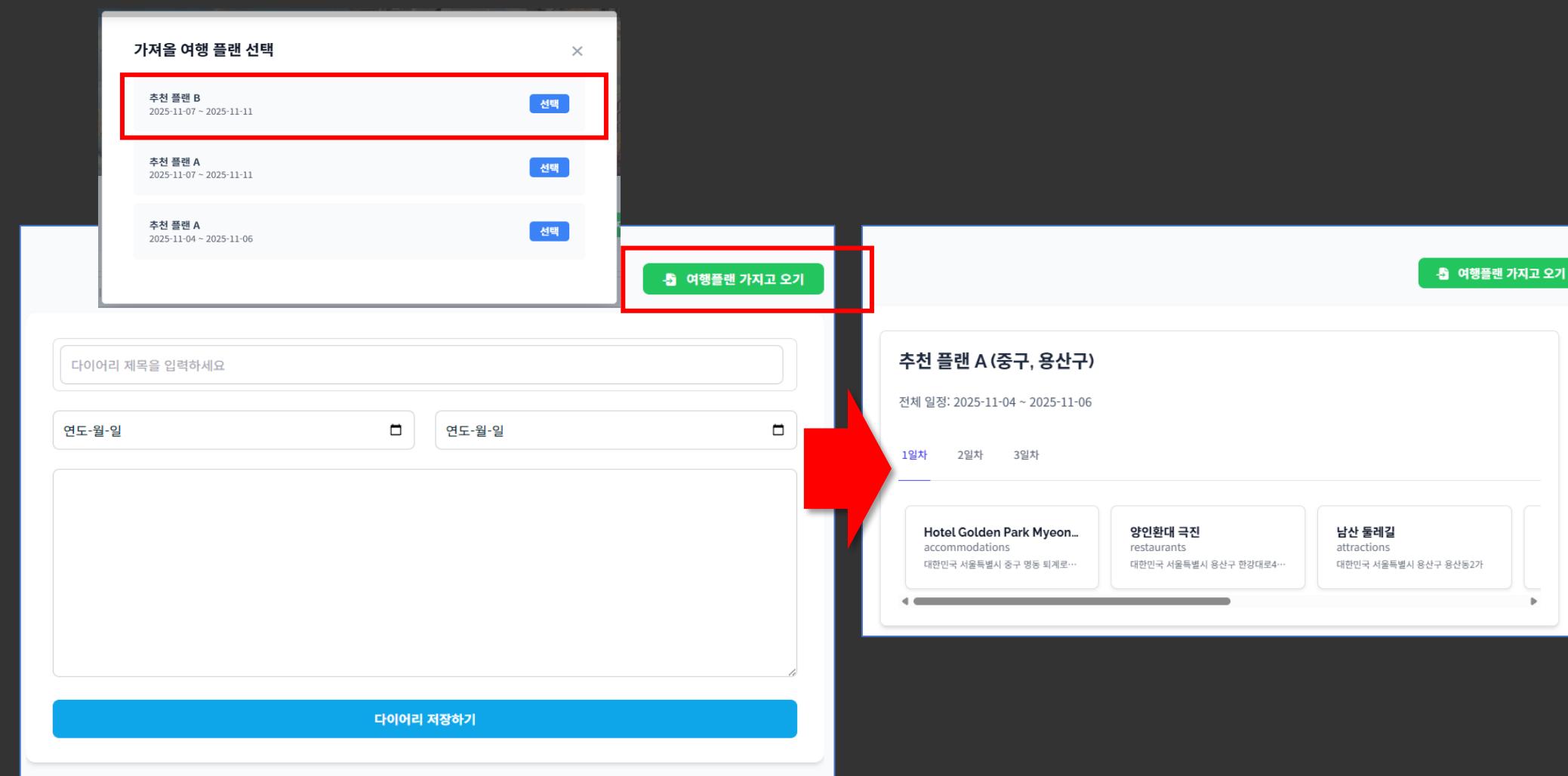
## 4. 여행 다이어리 프로세스(생성 PART)

생성

여행 플랜 불러오기

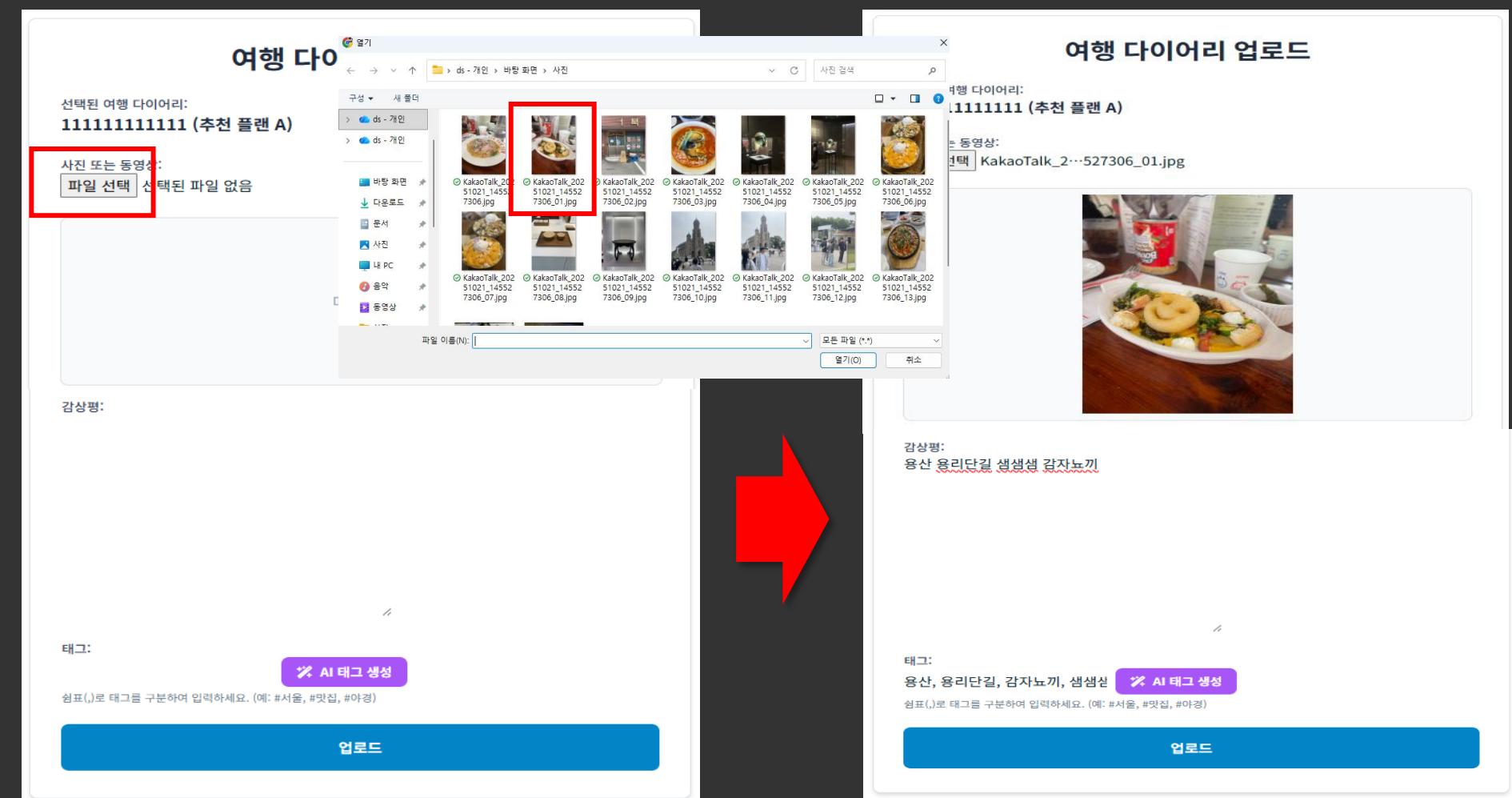
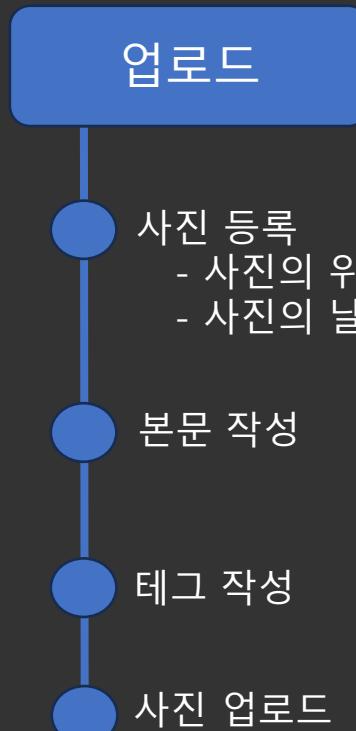
다이어리 정보 입력

다이어리 생성



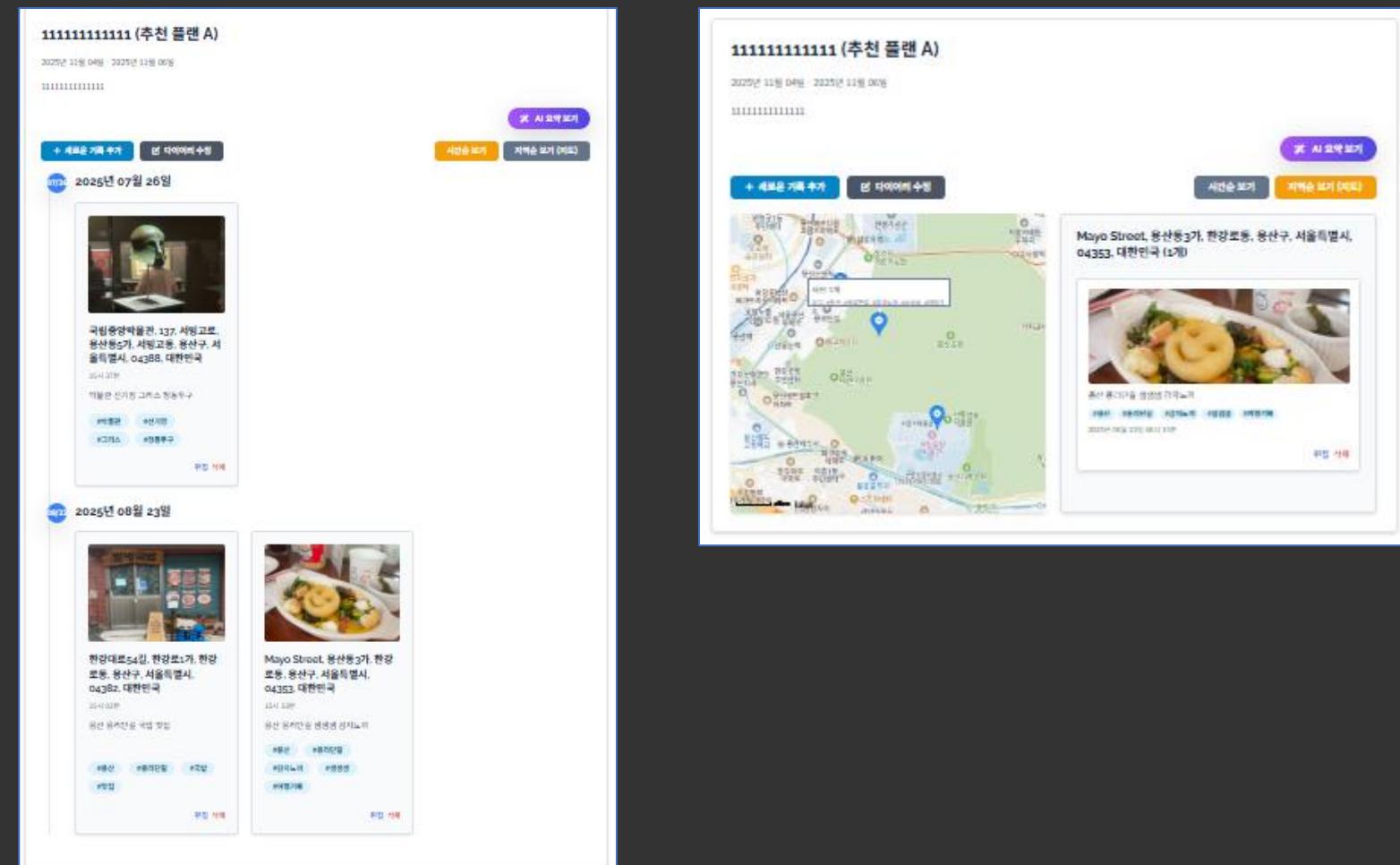
# 05. 프로젝트 수행 경과

## 4. 여행 다이어리 프로세스(업로드 PART)



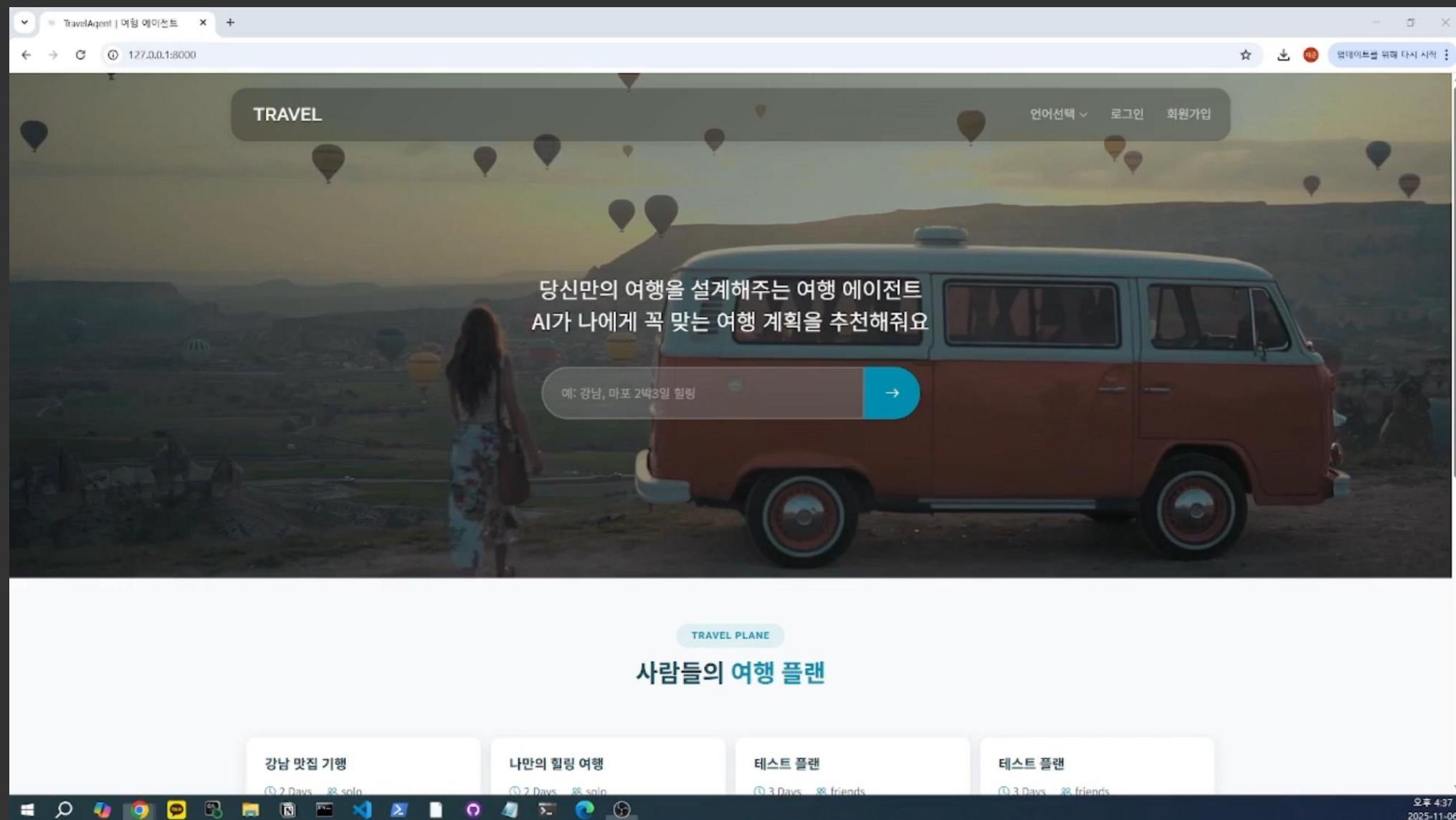
# 05. 프로젝트 수행 경과

## 4. 여행 다이어리 프로세스(뷰 PART)



# 05. 프로젝트 수행 경과

세부 기능 소개. 화면 구동 및 기능 동작 여부 시연 영상



이름	완성도 평가	개인 또는 우리 팀이 잘한 부분과 아쉬운 점	프로젝트 결과물의 추후 개선할 점	느낀점 또는 경험 성과
이용환	7	각자 맡은 기능 구현 기한 내 일정을 맞추지 못함/지도 및 플랜 부분 다양성 부족 조금 아쉬운기능	User의 선택 다양성 부족 (직접적인 User의 플랜 일정 선택) 다소 아쉬운 지도 및 플랜 UI&UX 완성도 부분	아이디어는 좋으나 경험이 적어 기능 구현에 어려움을 많이 겪음
이대식	7	잘한 점 : 빠른 의사결정, 적극적인 프로젝트 참여, 기발한 아이디어 등 아쉬운 점 : 기획이 부족해서, 최종 완성도를 높이지 못함 업무 분장 및 결과에 대한 정확한 목표가 부족했음.	1. 설정 지역 추가: 현재 서울 -> 전국 -> 해외로 확대 2. 여행에 필요한 정보 추가 : 지역, 기간에 대한 정보 추가 3. 여행에 필요한 편의 기능 추가 : 통-번역, 검색, 여행 정보(팁) 등	서비스 제작에 있어 우선순위와 작업의 순서를 알게 되었다. - 기획(설계)의 중요성 : 최종 산출물을 만들어 가는 과정의 이정표 - 작업 순서의 결정 : 작업 순서에 따라 불필요한 작업의 중복의 방지
길명철	8	잘한 점 : 최선을 다해 프로젝트를 진행한 점 아쉬운 점 : 계획을 빠르게 합의하지 못해 기능 MERGE에 어려움 발생	추가적인 장소 데이터와 한국 또는 해외까지 플랜을 제공할 수 있는 기능을 보완하면 좋을 것 같음	학원에서 배운 과정을 리마인드 할 수 있어 좋은 경험이 되었음
조시현	8	업무에 대해 각자 맡은 바 책임감이 큰 팀원들이란 점에서 박수를 보냄 여러 시도와 경험을 했지만, 많은 에러가 발생되는 등 더 깊은 공부가 부족했고 각자 맡은 바에 대한 의견 합치가 어려웠던 점은 아쉬웠음	각자의 코드를 엮어 merge를 시켜서 실제 사용에는 불안정함이 존재, 추후 한명이 코드를 통합해야 할 듯 함.	프로젝트를 통해서 코드에 대해 약간의 지식을 쌓을 수 있었으나 개인적으로 지식의 한계를 느낌
이재준	7	잘한 부분: 팀원 모두가 프로젝트에 적극적으로 참여 아쉬운 점 : 팀원 모두가 같은 기술을 사용하기에 역할 분담의 아쉬움	1. 날씨, 실시간 교통/ 이동 시간 등 연동 2. 추후 설정 지역을 전국 or 해외로 확대	본 프로젝트를 수행하면서 현대 웹 서비스 개발에 필수적인 비동기 처리 및 실시간 통신 환경에 대한 깊이 있는 이해와 실무 경험을 쌓음
강휘준	8.5	잘한 부분:Django의 기본 User 모델을 커스터마이징하여 UserProfile과 연동, 회원 데이터 구조를 설계함. 아쉬운 점 : 이메일 인증이나 비밀번호 재설정 기능은 구현하지 못함	이메일 인증 및 비밀번호 재설정 기능 추가: 실제 서비스 수준의 보안 강화	한계를 많이 느꼈음. 회원가입과 로그인 기능은 단순해 보였지만, 실제로 더 복잡하였음. 스스로 기본기 부족을 느낌
		잘한 부분: 데이터 수집 및 정리 과정에서 필요한 자료를 빠르게 확보하고, 팀원들이 사용할 수 있도록 공유한 점 아쉬운 점: 프로젝트의 흐름을 주도하는 개인의 이해하고, 주도적으로	수집한 데이터의 정확도와 일관성을 높이기 위해, 필터링 기준과 수집 철차를 더 체계화할 필요가 있음	이번 마지막 프로젝트에서는 팀의 흐름을 따라가며 함께 완성해가는 과정에서 많은 걸 배웠습니다!

감사합니다

구분		내용
팀별프로젝트결과물	결과보고서	<ul style="list-style-type: none"> <li>- [별첨2-2. 팀별 프로젝트 보고서(PPT)] 활용하여 작성</li> <li>- 목차 및 주요 내용 포함하고, 디자인 변경 가능</li> </ul>
결과물확인자료	시연영상	<ul style="list-style-type: none"> <li>- 프로젝트 결과물의 구조 및 기능을 설명하고, 실제 구현 여부를 확인할 수 있도록 제작된 영상</li> <li>- <u>팀별 5-10분 이내 (파일당 100MB 이하)</u></li> </ul>
소스코드		<ul style="list-style-type: none"> <li>- 프로젝트 결과물이 SW나 서비스인 경우 제출 권고</li> <li>- 깃허브 활용 가능/war, jar, zip 등 압축파일 형태가 아닌 압축이 풀린 tree 형태 권고</li> </ul>
데이터		<ul style="list-style-type: none"> <li>- AI/ML/데이터분석 기술이 활용된 경우 샘플데이터 제출 권고(이미지 파일 가능)</li> </ul>
접속링크		<ul style="list-style-type: none"> <li>- 링크에 접속하여 결과물 확인 가능한 환경을 말함</li> <li>- 결과물이 클라우드 기반일 경우 제출 권고</li> <li>- 결과물이 회원계정이 필요한 Web/App프로그램이나 서비스인 경우: 평가 자용 테스트 계정&amp;패스워드 제출</li> </ul>