

한성대학교
컴퓨터 공학부
2024-2 SW프리캡스톤디자인
최종보고서

마음의 날씨 - 감정 기록 플랫폼

팀원
임하늘
남궁민
이민주
김동현

담당교수: 김성동

프로젝트 계획서

1. 프로젝트 개요

1.1 프로젝트 이름: 마음의 날씨 - 감정 기록 플랫폼

1.2 문제점의 문

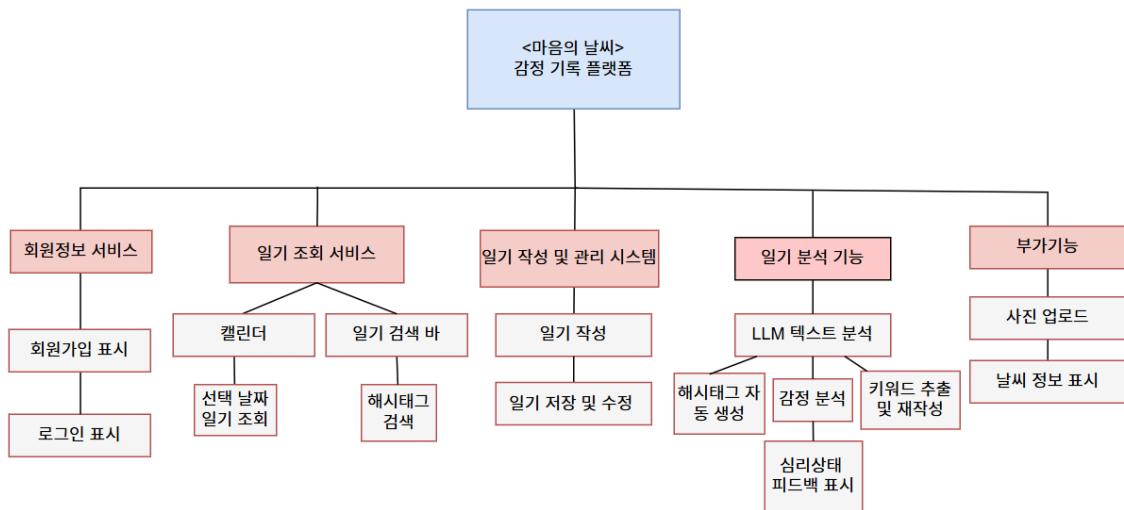
감정 분석과 심리 상담을 지원하는 간편한 일기 플랫폼

1.3 프로젝트 배경

기존 일기 앱은 감정 분석을 통한 심리 연계 기능이 부족해서 일기 작성 중 자신의 감정을 이해하거나 과거 일기를 보고 그날의 감정을 떠올리기 어려울 수 있다. 또한, 매번 일기 작성 시 날짜, 날씨, 감정 등을 수동으로 입력해야 하는 번거로움이 있으며, 과거 일기를 찾아보는 과정도 불편하다. 본 프로젝트는 LLM을 활용한 감정 분석과 심리 상담 기능을 통해 감정 상태를 반영한 일기 작성을 지원하고, 캘린더 및 해시태그 생성 기능으로 간편한 사용자 경험을 제공하여 일기를 통해 감정 기록뿐 아니라 심리 문제 해결 방법도 제시한다.

2. 기능 구조도 (Functional Decomposition Diagram)

2.1 기능 구조도



2.2 기능 설명 – 각 기능 간단 설명

2.2.1 회원정보 서비스

- firebase를 사용하여 전화번호 인증을 통한 회원가입과 로그인
- firebase userId를 조회하여 중복 확인, 데이터베이스에 없는 id는 새로운 사용자 등록

2.2.2 일기조회서비스

- 캘린더 : 각 일자에 일기의 해시태그를 미리보기로 제공
- 일기 검색바 : 해시태그 검색

2.2.3 일기 작성 및 관리 시스템

- 일기 작성창: 사용자가 형식에 상관없이 일기 작성
- 날씨 정보 표시: OpenWeatherMap API 사용 날씨 표시
- 일기 저장 및 수정: 일기 작성창에서 작성한 내용 저장 및 재작성된 일기 내용 수정

2.2.4 일기 분석 기능

- LLM 텍스트 분석 : 기업 API 사용

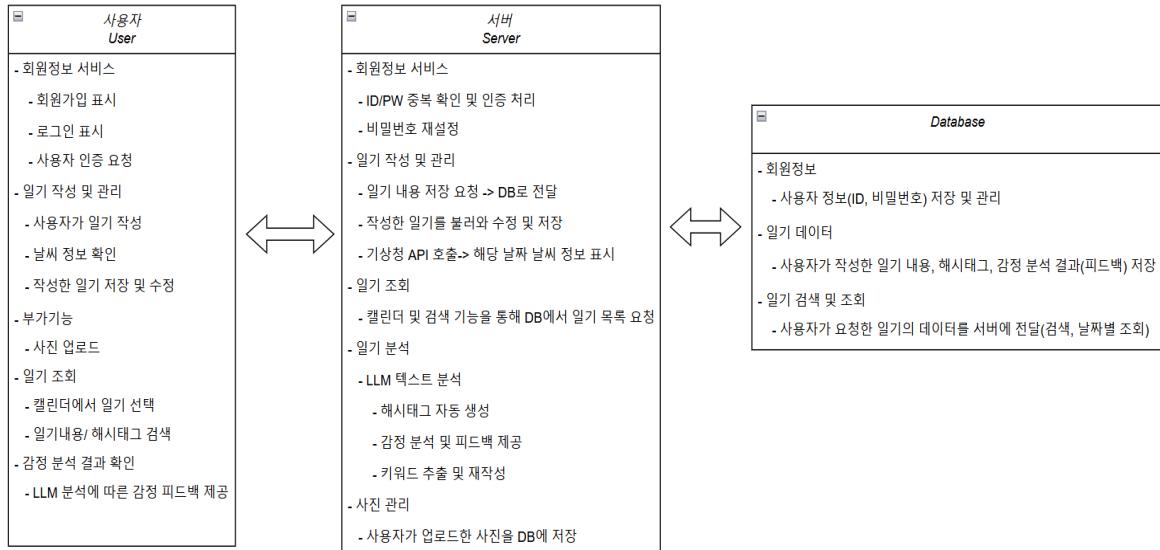
- 해시태그 자동 생성 : 텍스트를 분석하여 적절한 해시태그를 추출하여 일기 데이터에 추가
- 감정 분석 : 텍스트를 분석하여 감정을 분석하여 사용자에게 심리 상태 피드백 제공
- 키워드 추출 및 재작성 : 텍스트 내용 전체 분석 후 형식에 따라서 내용을 정리

2.2.5 부가 기능

- 사진업로드: 사용자가 로컬 기기에서 사진을 가져와 일기 데이터에 추가, DB에 저장

3. 시스템 구조도

3.1 시스템 구조도



3.2 시스템 구성 요소 설명

구성 요소	설명	개발 환경 및 구현 방법
사용자	<p>웹 애플리케이션을 통해 일기를 작성하고, 감정 피드백 수용</p> <p>OpenWeatherMap API로 날씨 정보를 받아 일기 작성에 활용하고 텍스트 분석 기능으로 일기 내용을 재작성하고 감정 분석 기능 사용</p>	<p>React: SPA (Single Page Application) 구조로 개발</p> <p>React Context API: 상태 관리 용이성을 위해 사용</p> <p>React Calendar 라이브러리: 일기 관리 기능 구현</p>

서버	<p>사용자의 요청을 처리하고 데이터베이스와 상호작용하여 일기 관리</p> <p>외부 API와 연동하여 날씨 정보 제공</p> <p>LLM을 호출하여 텍스트 분석 기능 수행</p>	<p>OpenAI API: LLM 텍스트 분석 기능 구현</p> <p>Node.js: 서버 환경 구축 및 RESTful API 개발</p> <p>OpenWeatherMap API: 날씨 데이터 제공을 위해 사용</p>
DB	<p>사용자 정보, 일기 내용, 분석결과 데이터를 저장하고 관리</p> <p>일기 작성 시 관련 데이터를 저장하며, 해시태그와 연동된 정보를 효율적으로 관리</p>	<p>MySQL 및 Firebase: 관계형 데이터베이스 시스템으로 데이터 저장, 관리</p> <p>SQL 쿼리: CRUD 작업 수행 (생성, 조회, 수정, 삭제)</p>

4. 일정 계획

4.1 간트 차트



4.2 일정 계획

주차	마일스톤	작업 설명	프론트엔드	백엔드
7주차	프로젝트 환경 설정	- 개발 환경 구성 (코드베이스 설정, 서버 구축, DB 설계 및 연결) - API 연동 준비 (OpenWeatherMap API, LLM API)	프론트엔드 1: 프로젝트 구조 설정, 초기 컴포넌트 구성 프론트엔드 2: 스타일 및 레이아웃 설정	백엔드 1: Node.js 서버 구축 백엔드 2: MySQL DB 설계 및 Firebase 연동 준비

8주차	회원관리 및 로그인 시스템 구현	- 회원가입, 로그인 기능 개발 - 사용자 인증 및 권한 처리 - DB와 연동하여 사용자 정보 관리	프론트엔드 1: 회원가입 및 로그인 UI 개발 프론트엔드 2: 사용자 인증 및 권한 처리	백엔드 1: 사용자 정보 DB 연동 백엔드 2: 헤더로 사용자 UID 받아오는 로직 작성성
9주차	일기 작성 및 조회 기능 개발	- 일기 작성 UI 개발 - 일기 작성 시 날씨 정보 자동 입력 - 작성된 일기 DB에 저장 및 조회 기능 구현	프론트엔드 1: 일기 작성 UI 개발 (날씨 표시 포함) 프론트엔드 2: 캘린더 UI 및 일기 조회 화면 개발	백엔드 1: 일기 작성 및 수정 API 개발 백엔드 2: 날씨 API 연동 및 DB 저장 처리
10주차	일기 분석 기능 및 부가기능 개발	- LLM API 연동하여 텍스트 분석 기능 구현 - 해시태그 자동 생성 및 감정 분석 - 사진 업로드 기능 개발	프론트엔드 1: 텍스트 분석 결과 UI 개발 (감정 분석 및 해시태그 생성 표시) 프론트엔드 2: 사진 업로드 UI 및 기능 개발	백엔드 1: LLM API 연동 및 텍스트 분석 처리 백엔드 2: 사진 업로드 처리 및 저장
11주차	검색 기능 개발	- 캘린더 기능을 활용한 일기 관리 시스템 개발 - 해시태그 및 텍스트 기반 일기 검색 기능	프론트엔드 1: 일기 관리 캘린더 UI 및 기능 개발	백엔드 1: 일기 관리 기능 API 개발 백엔드 2: 검색

		개발	프론트엔드 2: 검색 바 UI 및 검색 결과 화면 구현	API 개발 (해시태그 및 텍스트 검색)
12주차	통합 테스트 및 디버깅, 최종 배포	- 각 기능 통합 테스트 및 디버깅 - 성능 최적화 및 최종 배포 준비	프론트엔드 1: 버그 수정 및 성능 최적화	

5. 역할 분담

<임하늘> - 백엔드 서버 및 DB 연동
LLM API 연동 및 텍스트 분석 처리
이미지 관리(업로드, 저장 등) API 개발
검색 API 개발(해시태그 및 텍스트 검색)
백엔드 AWS EC2 배포

<남궁민> - 백엔드 서버 및 DB 연동
MySQL DB 설계
Node.js 서버 구축
사용자 정보 DB 연동
일기 관리(작성, 수정 등) API 개발
날씨 API 연동 및 DB 저장 처리

<김동현> - 프론트엔드 UI/UX 개발
프론트엔드 UI/UX 개발
파이어베이스 연동 및 프론트엔드 배포
전화번호 로그인 구현
캘린더 컴포넌트 구현
로딩 화면 표시
페이지 전환 애니메이션 표시
context 설계
백엔드 연동 및 데이터 관리, 렌더링

<이민주> - 프론트엔드 스타일 설계
일기 작성 UI 개발
텍스트 분석 결과 UI 개발
스타일 및 레이아웃 설정
사용자 인증 및 권한 처리
사진 업로드 UI 및 기능 개발

SW상세설계서

1. 문서개요

1.1 목적

이 문서의 목적은 “마음의 날씨: 감정 기록 플랫폼”의 상세 설계를 제시하여, 개발팀과 이해관계자가 시스템의 구조와 기능을 명확히 이해하도록 돕는 것이다.

1.2 범위

본 설계서는 시스템 구조, 데이터베이스 설계, 주요 모듈 및 기능을 포함하며, 사용자의 요구사항을 충족하는 솔루션을 제공하기 위한 기반이 된다.

1.3 참조 문서

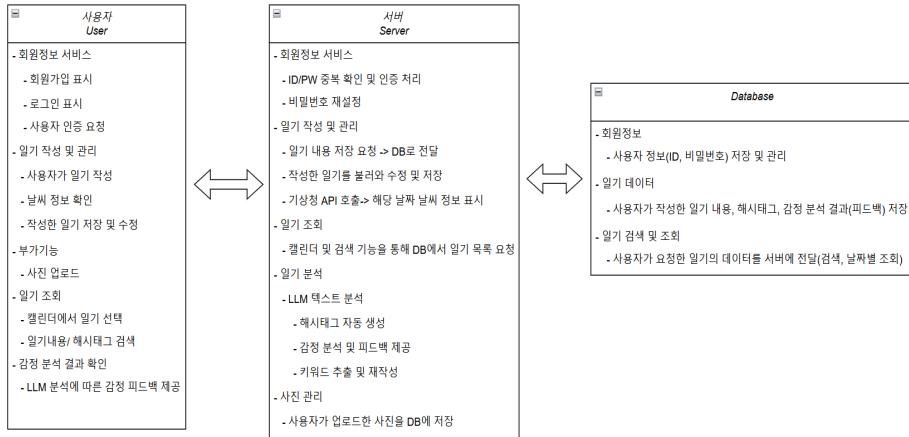
- 기능구조도 문서
- 시스템 구조도 문서
- 각각의 API관련 문서
 - OpenWeatherAPI 문서
 - LLM API문서
 - Firebase 인증 및 데이터베이스 문서
 - React, Node.js, MySQL 개발 문서 및 가이드

1.4 용어 정의

- 일기: 사용자가 작성하는 개인 기록
- API: 응용 프로그램 프로그래밍 인터페이스
- LLM: 대규모 언어 모델

2. 시스템 구조

2.1 전체 시스템 구조도



본 시스템은 사용자 관리, 일기 작성 및 조회, 일기 분석 기능을 제공해, 사용자는 자신의 일기를 작성하고 관리할 수 있으며, 일기 분석 기능을 통해 감정 분석 및 해시태그 자동 생성을 지원하고 또한, 사진 업로드 및 날씨 정보 표시와 같은 부가기능도 제공한다.

2.2 시스템 아키텍쳐

- **Presentation Layer**: 웹 및 모바일 인터페이스 제공, 사용자의 요청을 받아 서버로 전달
- **Business Logic Layer**: 사용자의 요청을 처리하고, 필요한 비즈니스 로직을 수행 데이터베이스와의 상호작용을 통해 정보를 처리
- **Data Layer**: 데이터베이스와의 직접적인 연결을 통해 데이터를 생성, 읽기, 업데이트 및 삭제 사용자 정보, 일기 데이터, 분석 결과 등을 저장 및 관리

2.3 인터페이스 정의

- 사용자 인터페이스: 웹 기반 UI를 통해 사용자에게 직관적인 경험을 제공한다.
- 외부 API 연동: OpenWeather API를 사용하여 실시간 날씨 정보를 가져온다.

3. 모듈설계

3.1 회원 정보 모듈

기능 설명: Firebase UID를 기반으로 사용자 인증 및 관리를 수행합니다.

입력: `firebase_uid` 출력: 사용자 정보, 처리 결과 메시지

프로세스 흐름:

1. Firebase UID를 사용하여 사용자를 확인합니다.
2. 새로운 사용자일 경우, 사용자 정보를 데이터베이스에 저장합니다.
3. 기존 사용자라면 환영 메시지를 반환합니다.

주요 클래스 및 메서드:

- `UserController`
 - `checkOrCreateUser(req, res)`: 사용자 확인 및 등록.
 - `getAllUsers(req, res)`: 모든 사용자 조회.
 - `deleteUser(req, res)`: 사용자 계정 삭제.

3.2 일기 관리 모듈

기능 설명: 일기 작성, 조회, 수정 및 삭제를 관리합니다.

입력: `firebase_uid, content, date, city` 출력: 일기 정보, 처리 결과 메시지

프로세스 흐름:

1. 새로운 일기를 작성하여 저장합니다.
2. 사용자 **UID**를 통해 정 분석을 재실행합니다.
3. 일기 특정 일기를 조회하거나 전체 일기를 가져옵니다.
4. 일기 내용을 수정하며, 수정 후 감와 관련된 모든 데이터를 삭제합니다.

주요 클래스 및 메서드:

- `DiaryController`
 - `createDiary(req, res)`: 새로운 일기 작성.
 - `getAllDiaries(req, res)`: 모든 일기 조회.
 - `getDiaryById(req, res)`: 특정 일기 조회.
 - `updateDiary(req, res)`: 일기 수정.
 - `deleteDiary(req, res)`: 일기 삭제 및 관련 데이터 정리.

3.3 일기 분석 모듈

기능 설명: OpenAI API를 통해 일기 내용을 분석하고 감정 및 해시태그를 생성합니다.

입력: `content, diary_id` 출력: 분석 결과, 피드백

프로세스 흐름:

1. 일기 내용을 OpenAI API를 통해 분석합니다.
2. 분석 결과(감정 및 해시태그)를 저장합니다.
3. 사용자가 요청 시 분석 결과를 반환합니다.

주요 클래스 및 메서드:

- AnalysisController
 - analyzeContent(req, res): 일기 내용 분석.
 - saveFeedback(req, res): 분석 결과 피드백 저장.
 - getFeedbackByDiaryId(req, res): 특정 일기 피드백 조회.
 - deleteFeedback(req, res): 피드백 삭제.

3.4 태그 관리 모듈

기능 설명: 해시태그를 생성, 검색 및 관리합니다.

입력: tags, diary_id, tagName 출력: 태그 목록, 처리 결과 메시지

프로세스 흐름:

1. 일기에 새로운 태그를 추가합니다.
2. 특정 태그를 기반으로 일기를 검색합니다.
3. 일기와 관련된 모든 태그를 삭제합니다.

주요 클래스 및 메서드:

- TagController
 - addTags(req, res): 일기에 태그 추가.
 - getDiariesByTag(req, res): 특정 태그로 일기 검색.
 - getTagsByDiaryId(req, res): 특정 일기의 태그 조회.
 - deleteTagsFromDiary(req, res): 특정 일기에서 태그 삭제.

3.5 사진 관리 모듈

기능 설명: 일기에 첨부된 사진을 업로드, 조회 및 삭제합니다.

입력: photo, diary_id 출력: 파일 경로, 처리 결과 메시지

프로세스 흐름:

1. 사용자가 업로드한 사진을 저장합니다.
2. 특정 일기에 첨부된 사진을 조회합니다.
3. 일기 삭제 시 관련된 모든 사진을 삭제합니다.

주요 클래스 및 메서드:

- PhotoController
 - uploadPhoto(req, res): 사진 업로드 및 경로 저장.
 - getPhotosByDiaryId(req, res): 특정 일기에 첨부된 사진 조회.

- `deletePhotosByDiaryId(req, res)`: 특정 일기에 첨부된 사진 삭제.

3.6 날씨 정보 모듈

기능 설명: OpenWeatherMap API를 활용해 실시간 날씨 정보를 제공합니다.

입력: `location` 출력: 날씨 정보

프로세스 흐름:

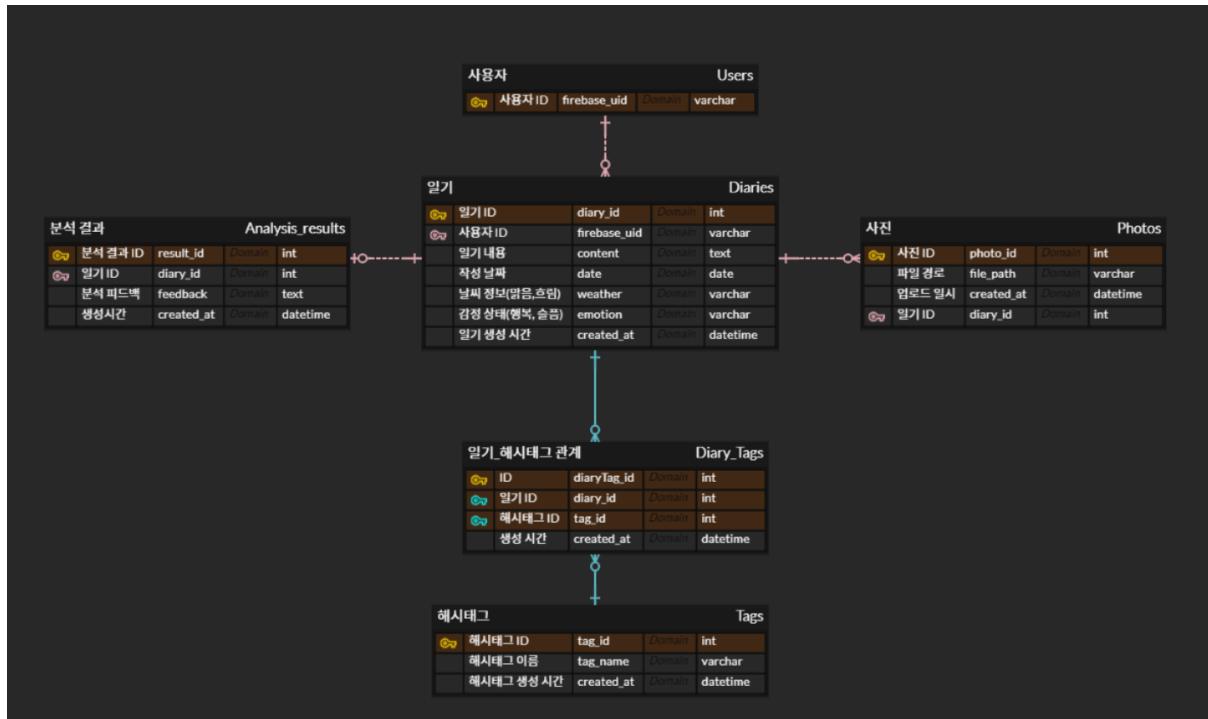
1. 사용자가 요청한 위치에 대한 날씨 정보를 조회합니다.
2. 날씨 데이터를 반환하여 일기 작성 시 활용합니다.

주요 클래스 및 메서드:

- `WeatherController`
 - `getWeatherInfo(req, res)`: 위치를 기반으로 날씨 데이터 조회

4. 데이터베이스 설계

4.1 ER 다이어그램



4.2 테이블 정의

사용자 테이블

테이블명: users

컬럼:

firebase_uid (VARCHAR) - Firebase 인증을 통한 사용자 고유 ID

일기 테이블 (Diary Table)

테이블명: diaries

컬럼:

- diary_id (PK, INT, AUTO_INCREMENT) - 일기 고유 ID
- firebase_uid (VARCHAR) - Firebase 인증을 통한 사용자 고유 ID
- content (TEXT) - 일기 내용
- date (DATE) - 일기 날짜 (사용자 지정하는 특정 날짜)
- weather (VARCHAR) - 날씨 정보 (예: 맑음, 흐림, 비 등)
- emotion (VARCHAR) - 감정 상태 (예: 행복, 슬픔, 분노 등)
- created_at (DATETIME) - 일기가 실제로 생성된 시간(?) 이게 필요할까?

인덱스: 작성 날짜에 인덱스 설정

- date 컬럼에 인덱스 설정
-

해시태그 테이블 (Tag Table)

테이블명: tags

컬럼:

- tag_id (PK, INT, AUTO_INCREMENT) - 해시태그 고유 ID
- tag_name (VARCHAR) - 해시태그 이름
- created_at (DATETIME)

인덱스:

- name 컬럼에 인덱스 설정
-

일기 해시태그 관계 테이블 (Diary Tag Relation Table)

테이블명: diary_tags

컬럼:

- diaryTag_id (PK, INT, AUTO_INCREMENT) - 고유 ID
- diary_id (FK, INT) - 일기 ID
- tag_id (FK, INT) - 해시태그 ID
- created_at (DATETIME)

인덱스: 검색용도

- diaryTag_id (PK, INT, AUTO_INCREMENT) - 고유 ID
-

일기 분석 결과 테이블 (Diary Analysis Result Table)

테이블명: diary_analysis_results

컬럼:

- result_id (PK, INT, AUTO_INCREMENT) - 고유 ID
 - diary_id (FK, INT) - 일기 ID
 - feedback (TEXT) - 감정 피드백
 - created_at (DATETIME)
-

사진 테이블 (Photo Table)

테이블명: photos

컬럼:

- photo_id (PK, INT, AUTO_INCREMENT) - 사진 ID
- diary_id (FK, INT) - 일기 ID
- file_path (VARCHAR) - 사진 파일 경로
- created_at (DATETIME)
-

4.3 테이블간 관계 정리

- 일대다 관계 (One-to-Many):
 - Diary - DiaryTag(일기와 일기태그 관계 테이블)

하나의 일기(Diary)는 여러 개의 해시태그(Tag)를 가질 수 있다.
 - Tag - DiaryTag(해시태그와 일기태그 관계 테이블)

하나의 해시태그는 여러 개의 일기에 연결될 수 있습니다.

 - Diary (일기) 1 : N DiaryTag (일기-태그 관계)
 - Tag (태그) 1 : N DiaryTag (일기-태그 관계)
 - Diary - Photo (일기와 사진 테이블)

하나의 일기(Diary)는 여러 장의 사진(Photo)을 가질 수 있습니다.

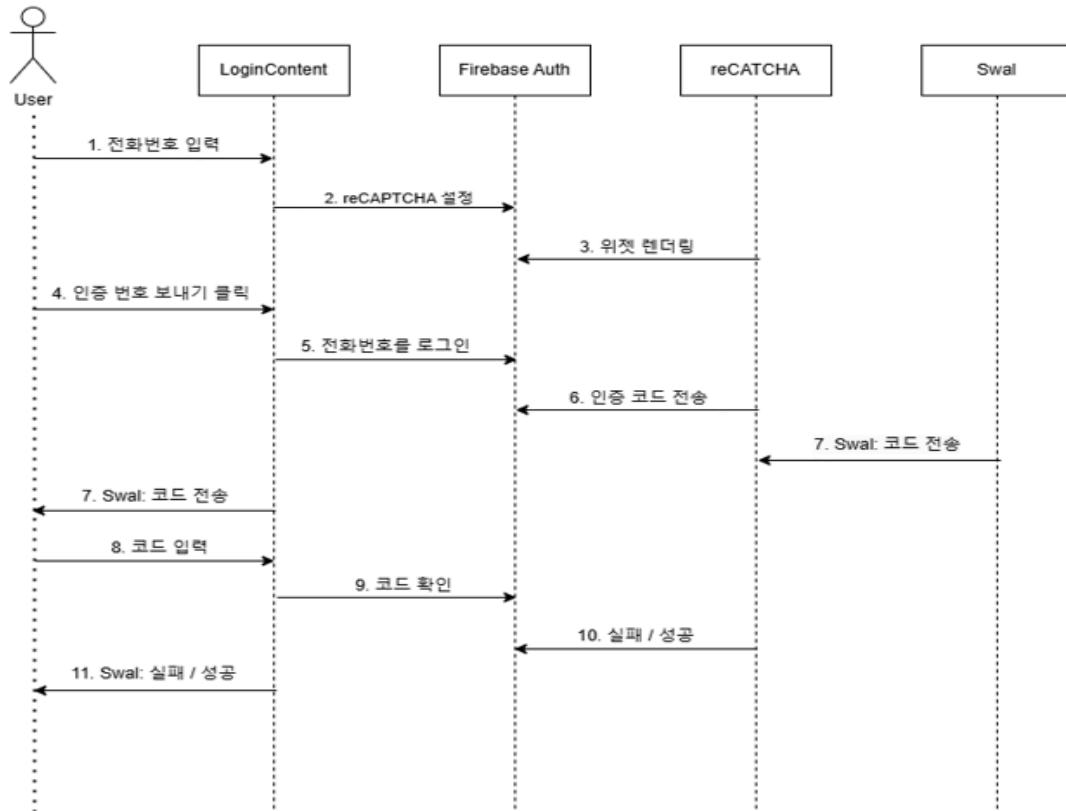
 - Diary (일기) 1 : N Photo (사진)
- 일대일 관계 (One-to-One):
 - **Diary - DiaryAnalysisResult (일기와 분석 결과 테이블)** 하나의 일기(Diary)는 하나의 분석 결과(DiaryAnalysisResult)만 가질 수 있습니다.
 - Diary (일기) 1 : 1 DiaryAnalysisResult (일기 분석 결과)
- 다대다 관계 (Many-to-Many):
 - **Diary - Tag (일기와 태그)** 일기와 태그는 다대다 관계로, 하나의 일기에는 여러 태그가 포함될 수 있고, 하나의 태그는 여러 일기에 적용될 수 있습니다.

이 다대다 관계는 DiaryTag라는 관계 테이블을 통해 구현됩니다.

 - Diary (일기) N : M Tag (태그) (중간 테이블: DiaryTag)

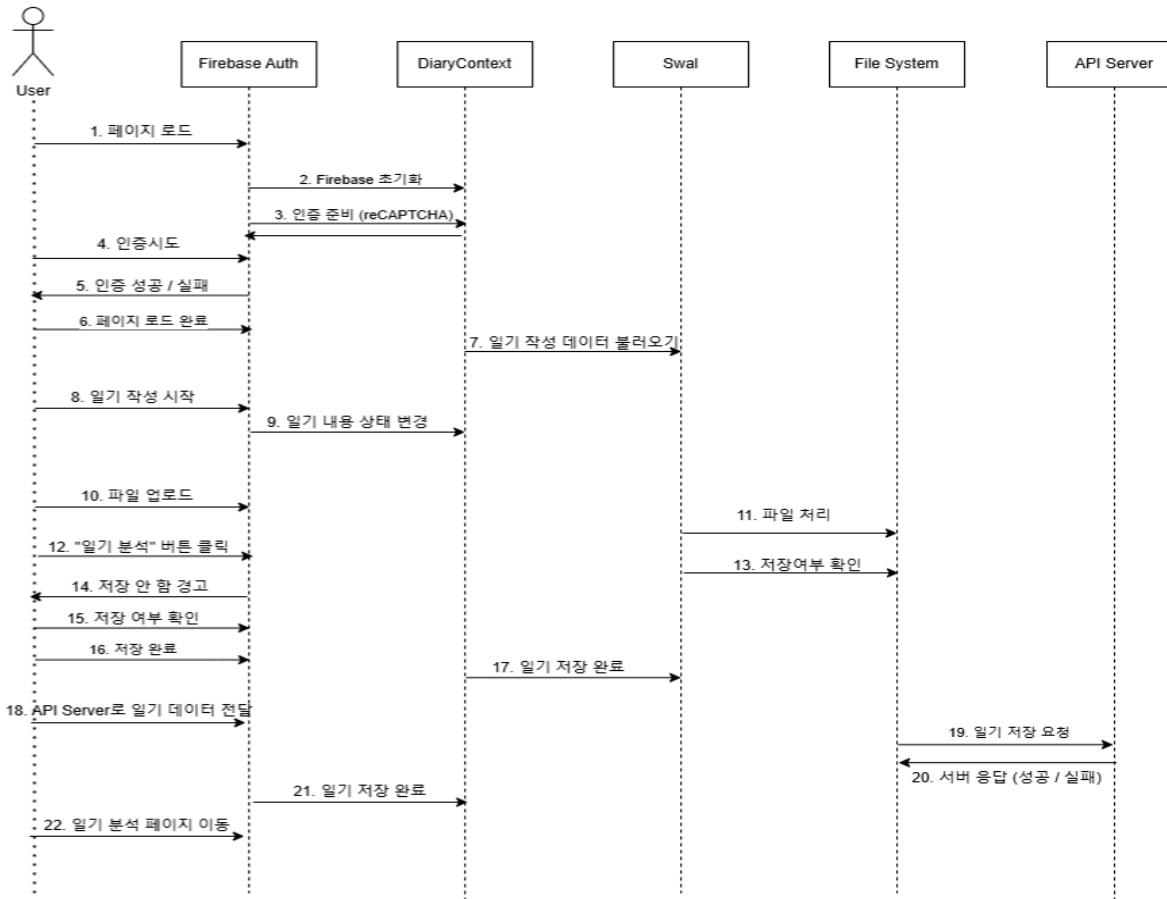
5. 시퀀스 다이어그램 (Sequence Diagrams)

5.1 사용자 등록 시퀀스 다이어그램



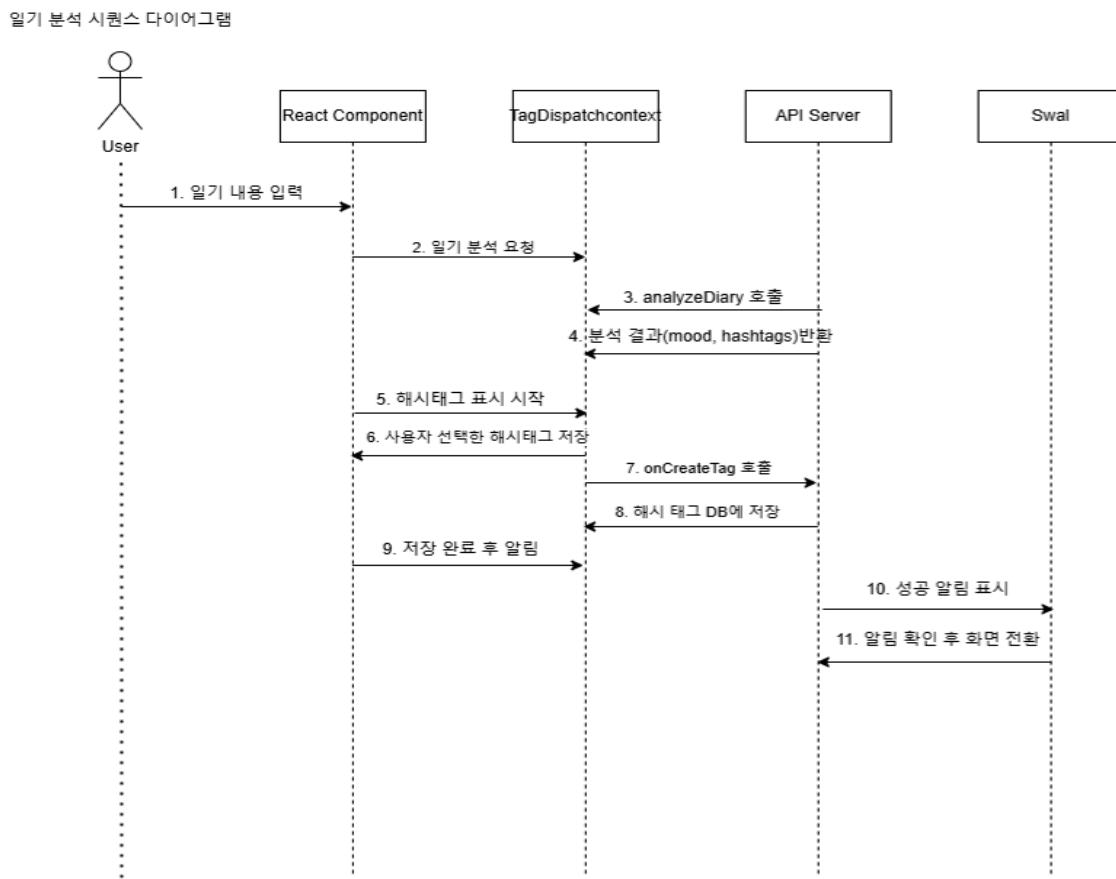
1. 사용자 전화번호 입력:
 - 사용자가 전화번호를 입력하고 전송 버튼을 클릭합니다.
2. reCAPTCHA 설정:
 - `setupRecaptchaVerifier` 함수가 호출되어 reCAPTCHA 위젯이 설정됩니다 (RecaptchaVerifier 생성).
 - 위젯은 화면에 보이지 않지만, `invisible` 모드로 설정되어 사용자에게 표시되지 않게 됩니다.
3. 전화번호 인증 코드 발송:
 - `signInWithPhoneNumber`를 호출하여 Firebase에서 인증 코드를 발송합니다.
 - Firebase가 코드를 전송하면 Swal(sweetalert2)을 사용하여 "Code Sent" 알림을 띄웁니다.
4. 사용자 인증 코드 입력 및 검증:
 - 사용자가 받은 인증 코드를 입력하고 "Verify" 버튼을 클릭하면, Firebase에서 해당 코드를 검증합니다.
 - 인증 성공/실패에 따라 Swal에서 "Success" 또는 "Fail" 알림을 띄웁니다.

5.2 일기 작성 시퀀스 디어그램



- 페이지 로드:
 - 사용자가 페이지에 접근하면 Firebase 앱이 초기화되고 인증 관련 준비가 됩니다.
- Firebase 인증 초기화 및 reCAPTCHA 설정:
 - Firebase 인증을 위한 reCAPTCHA 위젯을 초기화합니다.
- 날짜 유효성 검사:
 - 페이지에서 사용자가 입력한 날짜가 유효한지 확인합니다.
 - 유효하지 않으면 오류 메시지가 표시되고, 유효하면 모달 창이 표시됩니다.
- 일기 작성:
 - 사용자는 일기를 작성합니다.
 - 파일 업로드 버튼을 클릭하면 파일을 선택하고, 이미지 미리보기가 표시됩니다.
- 일기 저장:
 - 사용자가 일기 작성과 파일 선택을 완료하면 저장 버튼을 클릭하여 일기를 저장합니다.
 - Firebase에 일기 내용을 저장하고, 선택한 파일을 서버에 업로드합니다.
- 서버로 데이터 전송:
 - 서버는 일기 데이터를 받아서 저장하고, 성공적인 응답을 반환합니다.
- 분석 페이지 이동:
 - 일기가 성공적으로 저장되면 사용자는 분석 페이지로 이동하여 작성한 일기를 분석합니다.

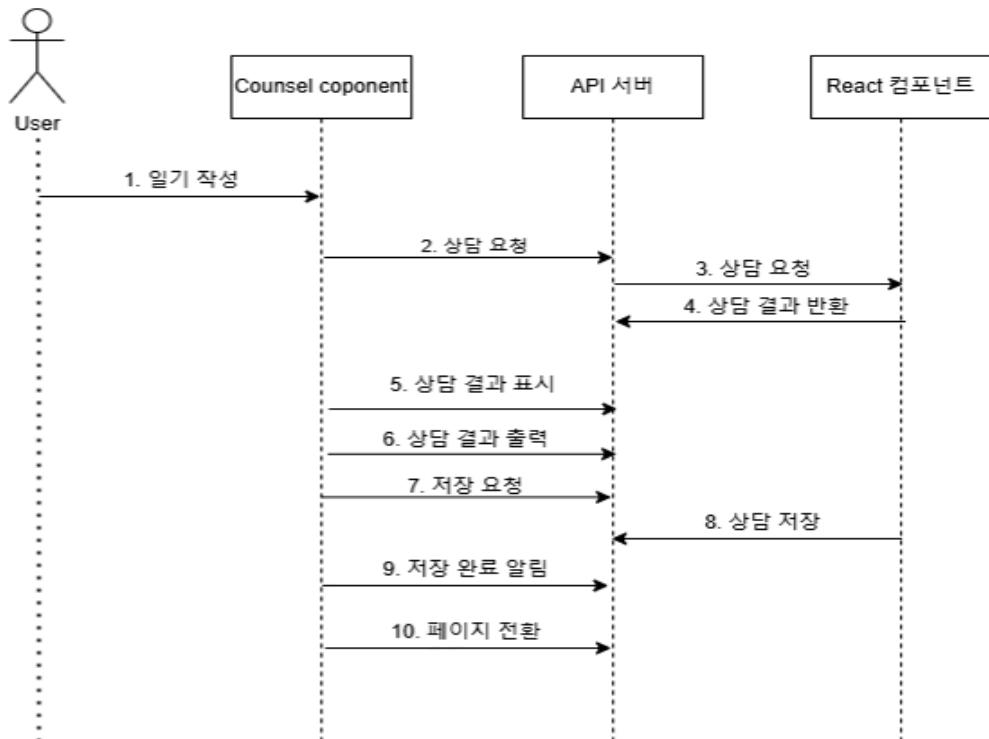
5.3 일기 분석 시퀀스 다이어그램



1. 일기 내용 입력: 사용자가 일기 내용을 작성합니다.
2. 일기 분석 요청: 일기 내용이 `analyzeDiary` 함수에 의해 분석됩니다.
3. `analyzeDiary` 호출: API 서버로 일기 내용이 전송되어 분석됩니다.
4. 분석 결과 반환: API 서버가 분석 결과인 기분(mood)과 해시태그(hashtags)를 반환합니다.
5. 해시태그 표시 시작: 해시태그를 1개씩 화면에 표시합니다.
6. 사용자 선택한 해시태그 저장: 사용자가 해시태그를 선택하고 저장을 요청합니다.
7. `onCreateTag` 호출: 선택된 해시태그를 API 서버로 전송하여 저장합니다.
8. 태그 DB에 저장: 해시태그가 DB에 저장됩니다.
9. 저장 완료 후 알림: 저장 완료 메시지가 화면에 표시됩니다.
10. 성공 알림 표시: 알림창이 화면에 나타납니다.
11. 알림 확인 후 화면 전환

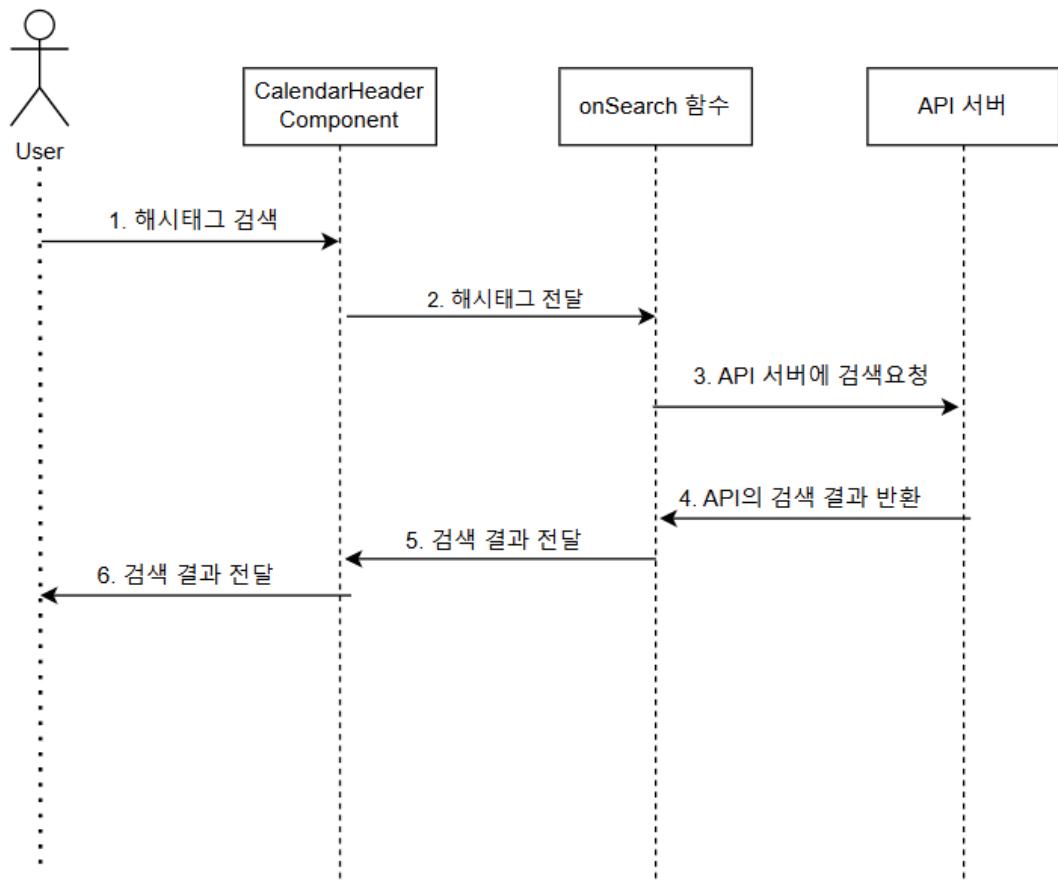
5.4 심리상담 시퀀스 다이어그램

심리상담 시퀀스



1. 일기 작성: 사용자가 일기를 작성합니다.
2. 상담 요청: 작성된 일기가 Counsel 컴포넌트로 전달되어 심리 상담을 요청합니다.
3. 상담 요청: Counsel 컴포넌트에서 일기 내용을 provideCounsel API로 전송하여 상담 요청을 보냅니다.
4. 상담 결과 반환: API 서버가 상담 피드백을 반환합니다. (기분 및 상담 피드백)
5. 상담 결과 표시: 반환된 상담 결과를 화면에 표시합니다.
6. 상담 내용 출력: 상담 내용을 화면에 출력하며, 텍스트가 순차적으로 나타나도록 처리합니다.
7. 저장 요청: 사용자가 저장 버튼을 클릭하면, 상담 피드백이 API로 전송되어 저장됩니다.
8. 상담 저장: API 서버가 상담 피드백을 저장합니다.
9. 저장 완료 알림: 상담 피드백이 성공적으로 저장되었다는 알림이 표시됩니다.
10. 페이지 전환: 상담이 종료된 후 사용자가 다른 페이지로 이동하게 됩니다.

5.5 해시태그 검색 시퀀스 다이어그램



1. 해시태그 검색 : 사용자가 해시태그를 검색
2. 해시태그 전달 : onSearch 함수를 호출하고 사용자가 검색한 해시태그를 전달
3. API서버에 검색 요청 : onSearch는 서버에 검색 요청
4. API의 검색 결과 반환 : 서버는 요청에 대한 검색 결과 반환
5. 검색 결과 전달 : 받은 검색 결과를 CalendarHeader로 전달
6. 검색 결과 전달 : 검색 결과를 사용자에게 화면에 표시

6. API설계(API 디자인)

6.1 사용자 관리 API

1. 사용자 생성

URL: /api/users/create

Method: POST

Request Body:

```
json  
코드 복사  
{  
  "firebase_uid": "abc123"  
}
```

Response:

성공 시 (201 Created):

```
json  
코드 복사  
{  
  "message": "새 사용자 등록 완료!",  
  "firebase_uid": "abc123"  
}
```

에러 처리:

- 500 Internal Server Error: 서버에서 오류 발생.

```
json  
코드 복사  
{  
  "error": "Failed to create user"  
}
```

2. 사용자 조회

URL: /api/users/check

Method: GET

Request Parameters: 없음

Response:

성공 시 (200 OK):

```
json
코드 복사
[
  {
    "firebase_uid": "abc123",
    "created_at": "2024-01-01T12:00:00Z"
  },
  {
    "firebase_uid": "xyz456",
    "created_at": "2024-01-02T14:00:00Z"
  }
]
```

에러 처리:

- 404 Not Found: 사용자가 존재하지 않는 경우.

```
json
코드 복사
{
  "error": "No users found"
}
```

3. 사용자 삭제

URL: /api/user/delete

Method: POST

Request Body:

```
json  
코드 복사  
{  
  "firebase_uid": "abc123"  
}
```

Response:

성공 시 (200 OK):

```
json  
코드 복사  
{  
  "message": "사용자 계정이 삭제되었습니다."  
}
```

에러 처리:

- 404 Not Found: 요청한 사용자가 존재하지 않는 경우.

```
json  
코드 복사  
{  
  "error": "User not found"  
}
```

4. 사용자 자동 확인 및 등록

URL: /api/users/{firebase_uid}

Method: GET

Request Parameters:

- firebase_uid (URL parameter, String): Firebase 사용자 고유 ID.

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "message": "기존 사용자입니다.",
  "firebase_uid": "abc123"
}
```

에러 처리:

- 500 Internal Server Error: 서버에서 오류 발생.

```
json
코드 복사
{
  "error": "Failed to retrieve or create user"
}
```

6.2 일기 관리 API

1. 일기 생성

URL: /api/diaries/create

Method: POST

Request Body:

```
json
코드 복사
{
  "firebase_uid": "abc123",
  "content": "오늘은 즐거운 하루였다.",
  "date": "2024-12-01",
  "city": "Seoul"
}
```

Response:

성공 시 (201 Created):

```
json
코드 복사
{
  "diary_id": 1,
  "firebase_uid": "abc123",
  "content": "오늘은 즐거운 하루였다.",
  "date": "2024-12-01",
  "weather": "맑음",
  "emotion": "행복"
}
```

에러 처리:

- 400 Bad Request: 요청 데이터가 부족하거나 잘못된 경우.

```
json
코드 복사
{
  "error": "Invalid request data"
}
```

2. 모든 일기 조회

URL: /api/diaries/check

Method: GET

Request Headers:

- firebase_uid (Header parameter, String): Firebase 사용자 고유 ID.

Response:

성공 시 (200 OK):

```
json
코드 복사
[
  {
    "diary_id": 1,
    "content": "오늘은 즐거운 하루였다.",
    "date": "2024-12-01",
    "weather": "맑음",
    "emotion": "행복"
  },
  {
    "diary_id": 2,
    "content": "비가 내려 우울했던 하루였다.",
    "date": "2024-12-02",
    "weather": "비",
    "emotion": "우울"
  }
]
```

에러 처리:

- 400 Bad Request: firebase_uid가 제공되지 않은 경우.

```
json
코드 복사
{
  "error": "Firebase UID is required"
}
```

3. 특정 일기 조회

URL: /api/diaries/{diary_id}/check

Method: GET

Request Parameters:

- diary_id (URL parameter, INT): 요청할 일기의 ID.

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "diary_id": 1,
  "firebase_uid": "abc123",
  "content": "오늘은 즐거운 하루였다.",
  "date": "2024-12-01",
  "weather": "맑음",
  "emotion": "행복"
}
```

에러 처리:

- 404 Not Found: 요청한 일기가 존재하지 않는 경우.

```
json
코드 복사
{
  "error": "Diary not found"
}
```

4. 일기 수정

URL: /api/diaries/{diary_id}/update

Method: POST

Request Body:

```
json
코드 복사
{
  "content": "오늘은 매우 특별한 하루였다."
}
```

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "message": "일기 수정 성공",
  "updated_diary": {
    "diary_id": 1,
    "content": "오늘은 매우 특별한 하루였다.",
    "date": "2024-12-01",
    "weather": "맑음",
    "emotion": "기쁨"
  }
}
```

에러 처리:

- 400 Bad Request: 요청 데이터가 부족한 경우.

```
json
코드 복사
{
  "error": "Content is required"
}
```

5. 일기 삭제

URL: /api/diaries/{diary_id}/delete

Method: POST

Request Headers:

- firebase_uid (Header parameter, String): Firebase 사용자 고유 ID.

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "message": "다이어리가 삭제되었습니다."
}
```

에러 처리:

- 404 Not Found: 요청한 일기가 존재하지 않거나 권한이 없는 경우.

```
json
코드 복사
{
  "error": "Diary not found or unauthorized"
}
```

6.3 태그 관리 API

1. 태그 추가

URL: /api/tag

Method: POST

Request Body:

```
json
코드 복사
{
  "firebase_uid": "abc123",
  "tags": ["행복", "카페"],
  "diary_id": 1
}
```

Response:

성공 시 (201 Created):

```
json
코드 복사
{
  "message": "태그 추가 성공"
}
```

에러 처리:

- 400 Bad Request: 요청 데이터가 부족한 경우.

```
json
코드 복사
{
  "error": "Invalid request data"
}
```

2. 특정 태그로 일기 검색

URL: /api/tag/search?tagName={tag}

Method: GET

Request Parameters:

- tagName (Query parameter, String): 검색할 태그 이름.

Response:

성공 시 (200 OK):

```
json
코드 복사
[
  {
    "diary_id": 1,
    "content": "오늘은 카페에 갔다.",
    "date": "2024-12-01"
  },
  {
    "diary_id": 2,
    "content": "카페에서 새로운 아이디어를 얻었다.",
    "date": "2024-11-30"
  }
]
```

에러 처리:

- 404 Not Found: 요청한 태그와 관련된 일기가 없는 경우.

```
json
코드 복사
{
  "error": "No diaries found for this tag"
}
```

3. 특정 일기의 태그 조회

URL: /api/tag/{diary_id}

Method: GET

Request Parameters:

- diary_id (URL parameter, INT): 요청할 일기의 ID.

Response:

성공 시 (200 OK):

```
json
코드 복사
[
    "행복",
    "카페"
]
```

에러 처리:

- 404 Not Found: 요청한 일기의 태그가 없는 경우.

```
json
코드 복사
{
    "error": "No tags found for this diary"
}
```

4. 특정 일기에서 태그 삭제

URL: /api/tag/{diary_id}

Method: DELETE

Request Parameters:

- diary_id (URL parameter, INT): 삭제할 태그가 연결된 일기의 ID.

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "message": "태그 삭제 성공. 필요 없는 태그 2개 삭제됨."
}
```

에러 처리:

- 404 Not Found: 요청한 일기의 태그가 없는 경우.

```
json
코드 복사
{
  "error": "No tags found for this diary"
}
```

6.4 사진 관리 API

1. 사진 업로드

URL: /api/photo/upload

Method: POST

Request Body:

- photo (Multipart file): 업로드할 사진 파일.
- diary_id (Form data): 사진이 연결될 일기의 ID.

Response:

성공 시 (201 Created):

```
json
코드 복사
{
  "message": "Photo uploaded and path saved!",
  "photo_id": 1
}
```

에러 처리:

- 400 Bad Request: 파일 또는 diary_id가 없는 경우.

```
json
코드 복사
{
  "error": "Photo and diary_id are required"
}
```

2. 특정 일기의 사진 조회

URL: /api/photo/{diary_id}

Method: GET

Request Parameters:

- diary_id (URL parameter, INT): 조회할 사진이 연결된 일기의 ID.

Response:

성공 시 (200 OK):

```
json
코드 복사
[
  {
    "photo_id": 1,
    "file_path": "/uploads/photos/2024/11/30/photo1.jpg",
    "created_at": "2024-11-30T15:30:00Z"
  }
]
```

에러 처리:

- 404 Not Found: 요청한 일기에 연결된 사진이 없는 경우.

```
json
코드 복사
{
  "error": "No photos found for this diary"
}
```

3. 특정 일기의 사진 삭제

URL: /api/photo/{diary_id}

Method: DELETE

Request Parameters:

- diary_id (URL parameter, INT): 삭제할 사진이 연결된 일기의 ID.

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "message": "All photos for this diary have been deleted"
}
```

에러 처리:

- 404 Not Found: 요청한 일기에 연결된 사진이 없는 경우.

```
json
코드 복사
{
  "error": "No photos found to delete for this diary"
}
```

6.5 날씨 정보 API

1. 실시간 날씨 조회

URL: /api/weather

Method: GET

Request Parameters:

- location (Query parameter, String): 요청할 위치 이름.

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "location": "Seoul",
  "weather": {
    "temperature": 20.5,
    "description": "clear sky",
    "rain": 0
  }
}
```

에러 처리:

- 400 Bad Request: 위치 정보가 제공되지 않은 경우.

```
json
코드 복사
{
  "error": "Location is required"
}
```

- 500 Internal Server Error: 외부 API 호출 중 오류 발생.

```
json
코드 복사
{
  "error": "Failed to fetch weather data"
}
```

6.6 일기 분석 API

1. 일기 내용 분석

URL: /api/openai/analyze

Method: POST

Request Body:

```
json
코드 복사
{
  "content": "오늘 하루는 정말 힘들었다. 하지만 희망을 잃지 않기로 다짐했다."
}
```

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "mood": "희망",
  "hashtags": ["희망", "다짐", "하루"]
}
```

에러 처리:

- 400 Bad Request: 요청 데이터가 없는 경우.

```
json
코드 복사
{
  "error": "Diary content is required"
}
```

- 500 Internal Server Error: OpenAI API 호출 중 오류 발생.

```
json
코드 복사
{
  "error": "Failed to analyze diary content"
}
```

2. 심리 상담 요청

URL: /api/openai/counsel

Method: POST

Request Body:

```
json
코드 복사
{
  "content": "오늘은 너무 우울하고 혼자 있는 기분이 들었다."
}
```

Response:

성공 시 (200 OK):

```
json
코드 복사
{
  "counseling": "당신은 지금 외로움을 느끼고 있는 것 같습니다. 외출하여 친구를
    만나거나 새로운 취미를 시작해보세요."
}
```

에러 처리:

- 400 Bad Request: 요청 데이터가 없는 경우.

```
json
코드 복사
{
  "error": "Diary content is required"
}
```

- 500 Internal Server Error: OpenAI API 호출 중 오류 발생.

```
json
코드 복사
{
  "error": "Failed to provide counseling"
}
```

7. 에러 처리 설계

7.1 공통 에러 처리 방식

- **400 Bad Request:** 클라이언트 요청이 잘못된 경우 (예: 요청 데이터 부족, 잘못된 형식).
- **401 Unauthorized:** 인증이 필요한 요청에서 인증 정보가 없거나 잘못된 경우.
- **403 Forbidden:** 권한이 없는 사용자가 요청을 시도한 경우.
- **404 Not Found:** 요청한 데이터가 존재하지 않는 경우.
- **500 Internal Server Error:** 서버 내부에서 예기치 않은 오류가 발생한 경우.

7.2 모듈별 에러 처리

7.2.1 사용자 관리

- 시나리오: 요청한 사용자가 없을 때.
 - 에러 코드: **404 Not Found**
 - 에러 메시지: "User not found"
- 시나리오: Firebase UID가 제공되지 않을 때.
 - 에러 코드: **400 Bad Request**
 - 에러 메시지: "Firebase UID is required"

7.2.2 일기 관리

- 시나리오: 요청한 일기가 존재하지 않을 때.
 - 에러 코드: **404 Not Found**
 - 에러 메시지: "Diary not found"
- 시나리오: 잘못된 형식의 날짜가 제공되었을 때.
 - 에러 코드: **400 Bad Request**
 - 에러 메시지: "Invalid date format"

7.2.3 태그 관리

- 시나리오: 태그가 없는 일기를 조회할 때.
 - 에러 코드: **404 Not Found**
 - 에러 메시지: "No tags found for this diary"

7.2.4 사진 관리

- 시나리오: 사진이 없는 일기를 조회할 때.
 - 에러 코드: **404 Not Found**
 - 에러 메시지: "No photos found for this diary"

7.2.5 날씨 정보

- 시나리오: 위치 정보가 제공되지 않을 때.
 - 에러 코드: **400 Bad Request**
 - 에러 메시지: "Location is required"

7.2.6 일기 분석

- 시나리오: OpenAI API 호출 실패.
 - 에러 코드: 500 Internal Server Error
 - 에러 메시지: "Failed to analyze diary content"

8. 보안설계

8.1 사용자 인증

- Firebase Authentication:
 - 모든 요청에 `firebase_uid`를 포함해 사용자 인증 수행.
 - Firebase에서 발급된 토큰으로 사용자 신원을 검증.

8.2 데이터 접근 권한

- 권한 검증:
 - 사용자의 `firebase_uid`와 요청된 데이터(`diary_id`, `photo_id`, 등)를 매칭해 데이터 접근 권한 확인.
 - 데이터 소유자만 접근/수정/삭제 가능.

8.3 요청 데이터 검증

- 모든 요청 데이터에 대해 유효성 검증 수행:
 - JSON 스키마 검증.
 - 필수 필드 확인 (예: `content`, `diary_id` 등).
 - 잘못된 요청 데이터 차단.

8.4 민감 정보 보호

- 환경 변수:
 - OpenAI API 키, Firebase 구성 정보, OpenWeatherMap API 키 등을 `.env` 파일을 통해 관리.
 - Git 저장소에 포함되지 않도록 `.gitignore`에 추가.
- 비밀번호 및 토큰:
 - 비밀번호는 Firebase에서 관리하며 서버에 저장하지 않음.
 - 요청마다 Firebase 인증 토큰 사용.

8.5 로그 관리

- 모든 중요한 이벤트 (로그인 실패, 데이터 접근 실패 등)를 서버에 기록.
- 민감한 정보는 로그에 저장하지 않음.

9. 테스트 계획

단위 테스트 (Unit Test)

- 각 기능별로 독립적으로 실행.
- 주요 컨트롤러와 서비스로직 검증.
- 예: Firebase UID가 없는 경우 400 Bad Request를 반환하는지 확인.

통합 테스트 (Integration Test)

- 여러 모듈 간의 데이터 흐름 및 상호작용 확인.
- 예: 일기 삭제 시 관련 태그와 사진도 함께 삭제되는지 확인.

시스템 테스트 (System Test)

- 전체 시스템이 통합된 상태에서 요구사항이 충족되는지 확인.
- 예: 일기 생성 → 내용 분석 → 태그 추가까지의 작업 흐름 테스트.

부하 테스트 (Load Test)

- 동시 다수의 사용자가 API를 호출할 경우 시스템이 안정적으로 작동하는지 확인.
- 예: 초당 100건 이상의 요청이 들어왔을 때 성능 유지 여부.

보안 테스트 (Security Test)

- 인증되지 않은 사용자의 요청이 차단되는지 확인.
- 예: Firebase UID가 없는 경우 데이터를 조회할 수 없는지 확인.

10. 배포 계획

배포 환경

- 프론트엔드: AWS S3 (정적 웹사이트 호스팅)
- 백엔드: AWS EC2 (Node.js 기반 서버)
- 데이터베이스: AWS RDS (MySQL)

최종발표자료

☀ 마음의 날씨 :
감정기록 플랫폼

3팀 불타는 감자 : 김동현 남궁민 이민주 임하늘

목차

1. 개요

- 프로젝트 배경
- 기능별 설명

2. 기술 아키텍처

- 기술 스택
- 프로젝트 아키텍처

3. 역할 분담

4. 기능 구현

5. UI 화면

6. 발견된 문제 및 해결과정

마음의 날씨 : 감정 기록 플랫폼

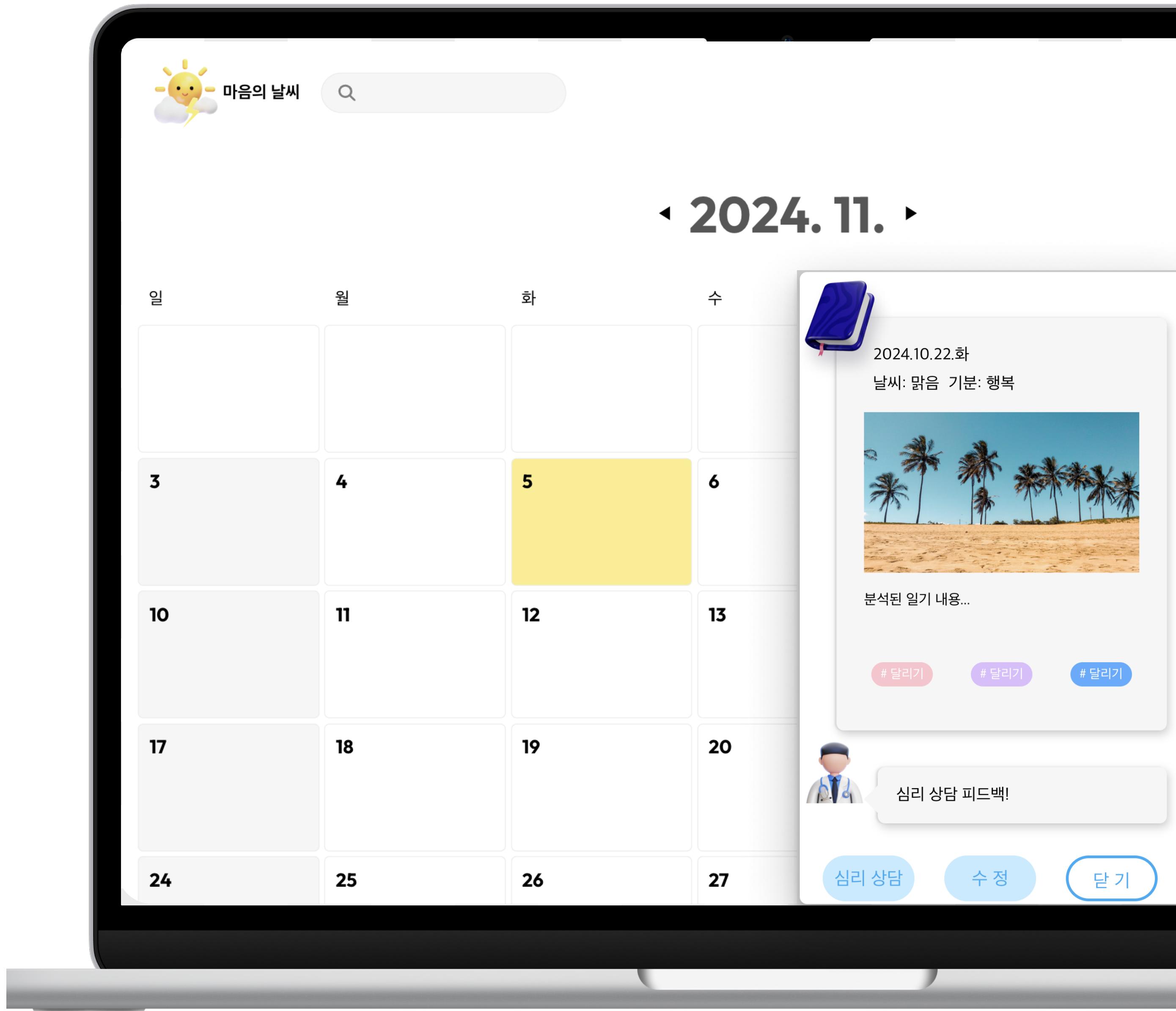
프로젝트 배경

이 프로젝트는 기존 일기 앱의 한계를 극복하기 위해 LLM을 활용한 감정 분석 및 심리 상담 기능을 도입합니다.

사용자는 감정 상태를 반영한 일기를 쉽게 작성할 수 있으며, 자동으로 날짜, 날씨, 감정을 입력할 수 있는 기능을 제공합니다.

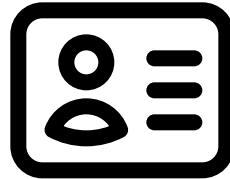
또한, 캘린더와 해시태그 생성 기능을 통해 과거 일기를 손쉽게 찾아볼 수 있어 사용자 경험을 향상시킵니다.

이를 통해 일기 작성으로 감정 기록 뿐 아니라 심리 문제 해결 방법도 제시하는 목표를 가지고 있습니다.



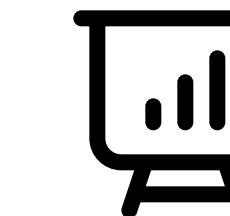
마음의 날씨 : 감정 기록 플랫폼

기능별 설명



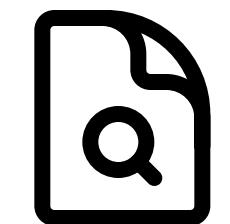
회원정보 서비스

- Firebase Authentication의 전화번호 인증 가능



일기 분석 기능: OpenAI API

- 해시태그 자동 생성 : 텍스트 분석 후 관련된 해시태그 6개 생성
- 심리 분석 : 텍스트 분석을 통해 심리 상태 피드백 제공
- 키워드 추출 및 재작성 : 내용 분석 후 정리



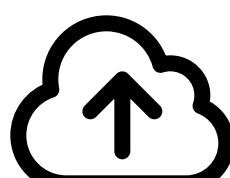
일기조회 서비스

- 캘린더 : 각 일자 별 해시태그 보기 제공
- 일기 검색바 : 해시태그 검색시 관련 일기 자동 하이라이트 처리



일기 작성 및 관리 시스템

- 일기 작성창 : 형식에 상관없이 자유롭게 작성
- 날씨 정보 : OpenWeatherMap API를 통해 날씨 표시
- 일기 저장 및 수정 : 작성한 내용 저장 및 수정 가능



부가 기능

- 사진 업로드 : 사진 추가 시 AWS EC2 인스턴스의 로컬 디스크 저장 가능

마음의 날씨 : 감정 기록 플랫폼

기술스택 프로젝트 아키텍처

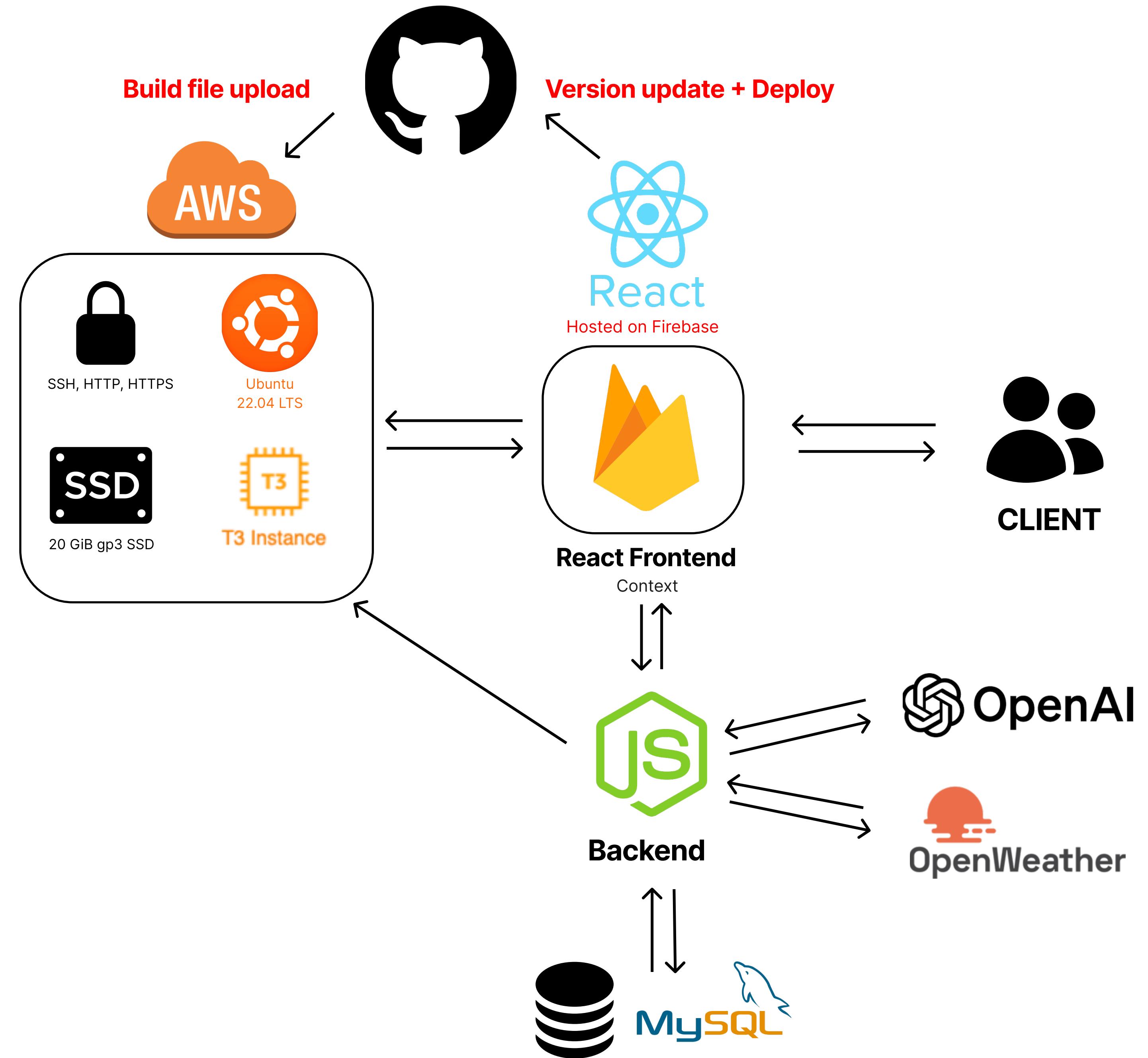
Language
JavaScript

Library & FrameWork
React, Node.js

Database
MySQL, Firebase

Tool
AWS, GitHub

etc. Figma Notion



마음의 날씨 : 감정 기록 플랫폼

역할 분담



임하늘

<백엔드 1>

- Node.js 서버 구축
- 사용자 정보 DB 연동
- 일기 작성 및 수정 API 개발
- LLM API 연동 및 텍스트 분석 처리
- 일정 관리 기능 API 개발
- AWS 서버 배포



남궁민

<백엔드 2>

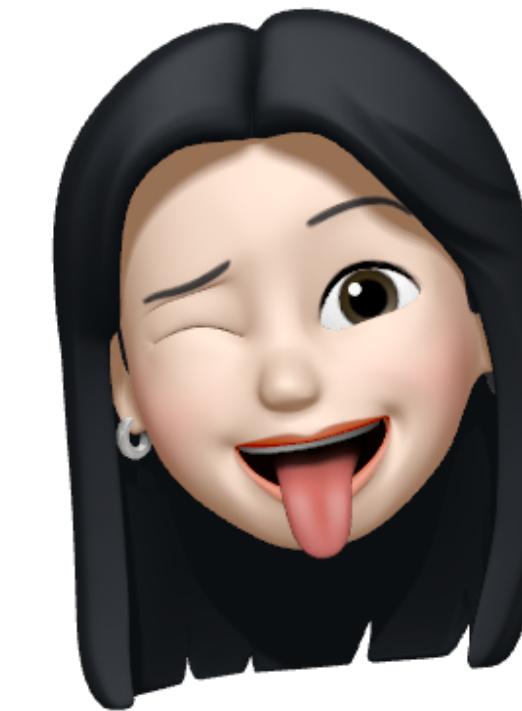
- MySQL DB 설계 및 Firebase 연동 준비
- 날씨 API 연동 및 DB 저장처리
- 사진 업로드 처리 및 저장
- 검색 API 개발
(해시태그 및 텍스트 검색)



김동현

<프론트엔드 1>

- 프로젝트 구조 설정, 초기 컴포넌트 구성
- 회원가입 및 로그인 UI 개발
- 일기 작성 UI 개발 (날씨 표시 포함)
- 텍스트 분석 결과 UI 개발 (감정 분석 및 해시태그 생성 표시)
- 일정관리 캘린더 UI 및 기능 개발
- 버그 수정 및 성능 최적화



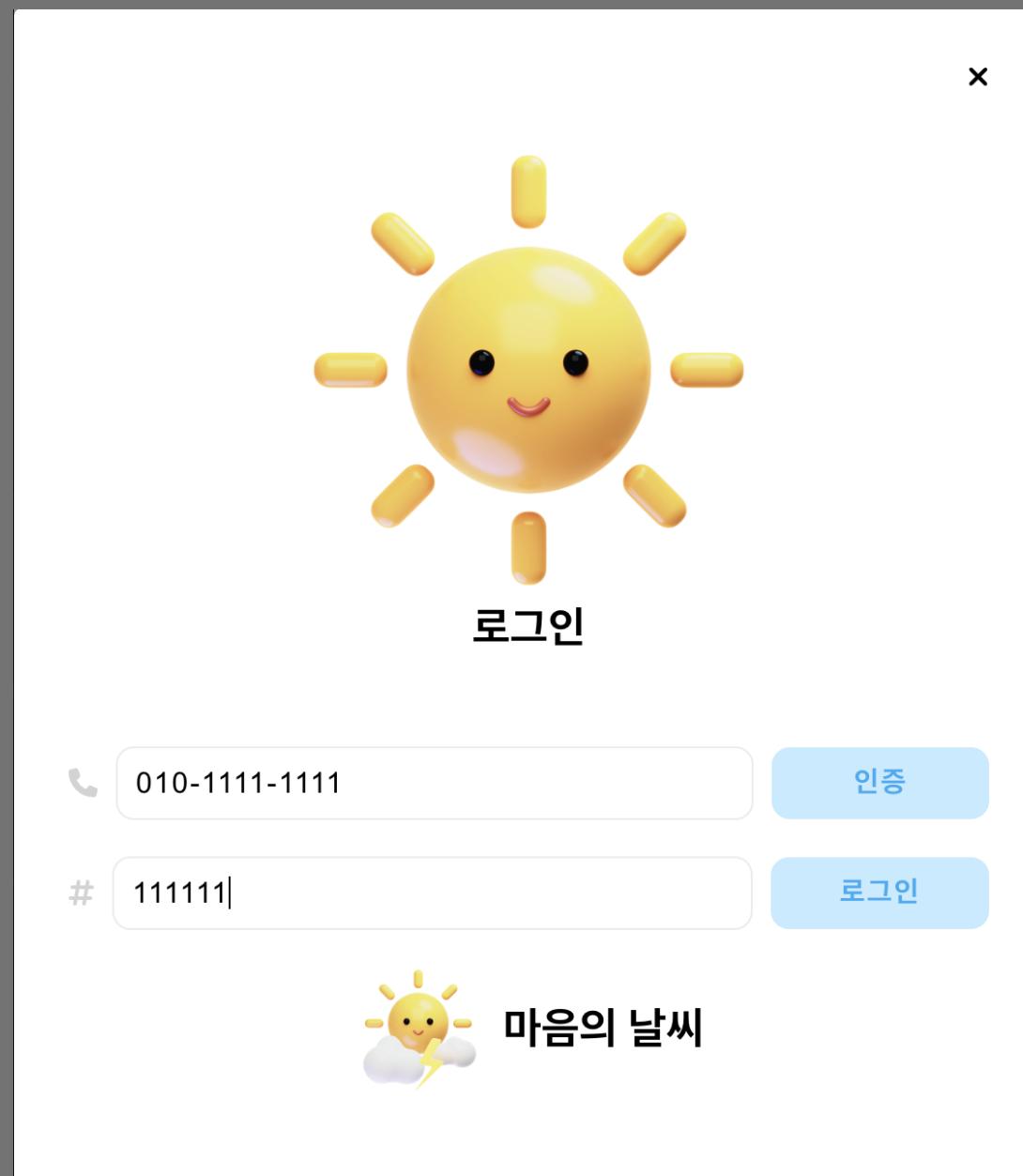
이민주

<프론트엔드 2>

- 스타일 및 레이아웃 설정
- 사용자 인증 및 권한 처리
- 캘린더 UI 및 일기 조회 화면 개발
- 사진 업로드 UI 및 기능 개발
- 검색 바 UI 및 검색 결과 화면 구현

마음의 날씨 : 감정 기록 플랫폼

기능 구현



마음의 날씨

달리기 *

32°
Sunny Cloudy

Moni Roy
Admin

Profile
Setting
Logout

◀ 2024 10 ▶

1	2	3	4	5	6	7
8	9 # 하늘 *	10	11	12 # 달리기 * # 앱기억복이 *	13	14
15 # 우울 *	16	17 # 쇼핑하기 * # 달리기 *	18	19	20	21
22	23 # 행복 *	24	25 # 놀기 * # 민 *	26	27 # 시험공부 * # 학교 *	28
29	30 # 힐링 *	31	1 June	2	3	4

2024.10.22, 화

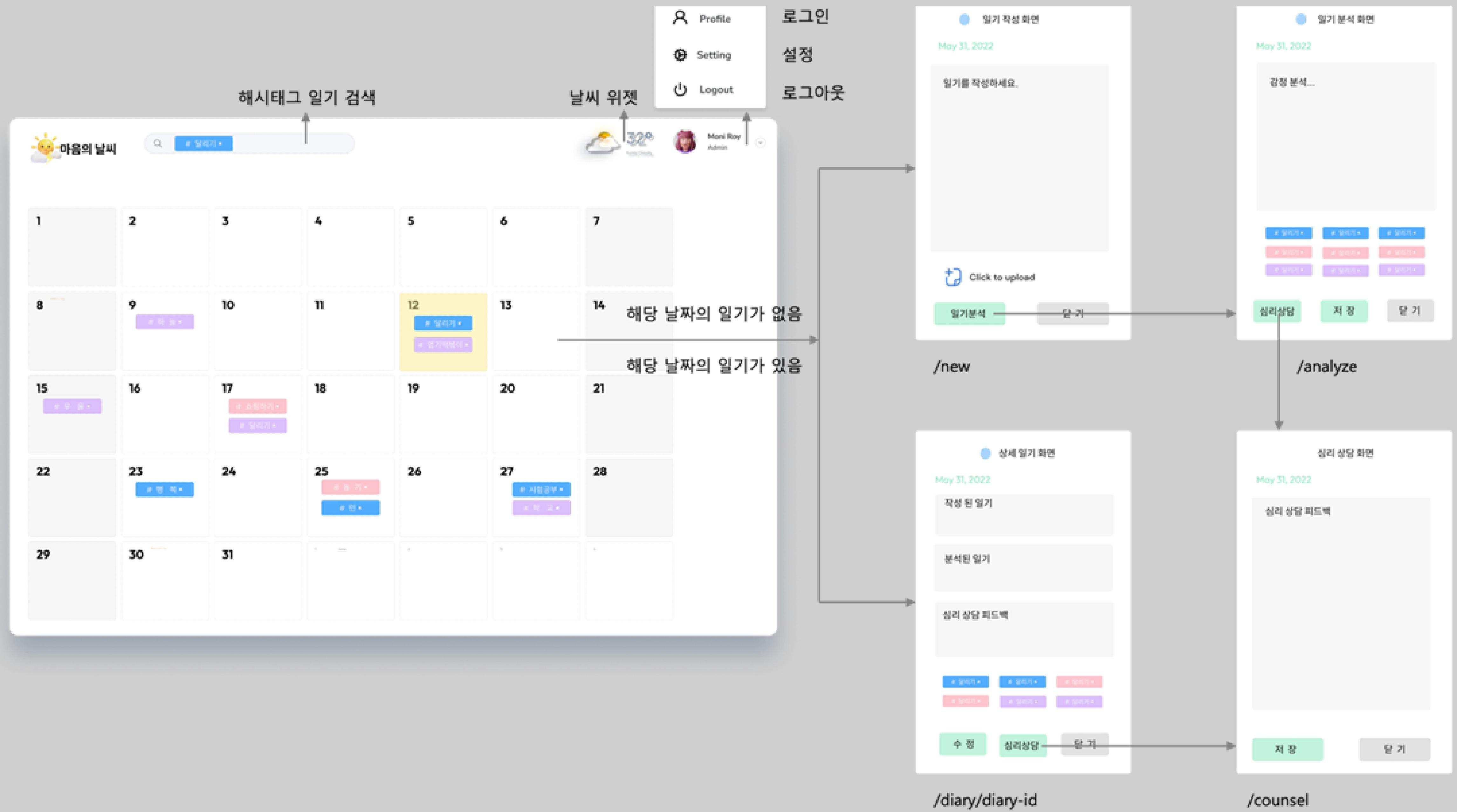
심리 상담 피드백!

날씨 : 맑음 기분 : 행복

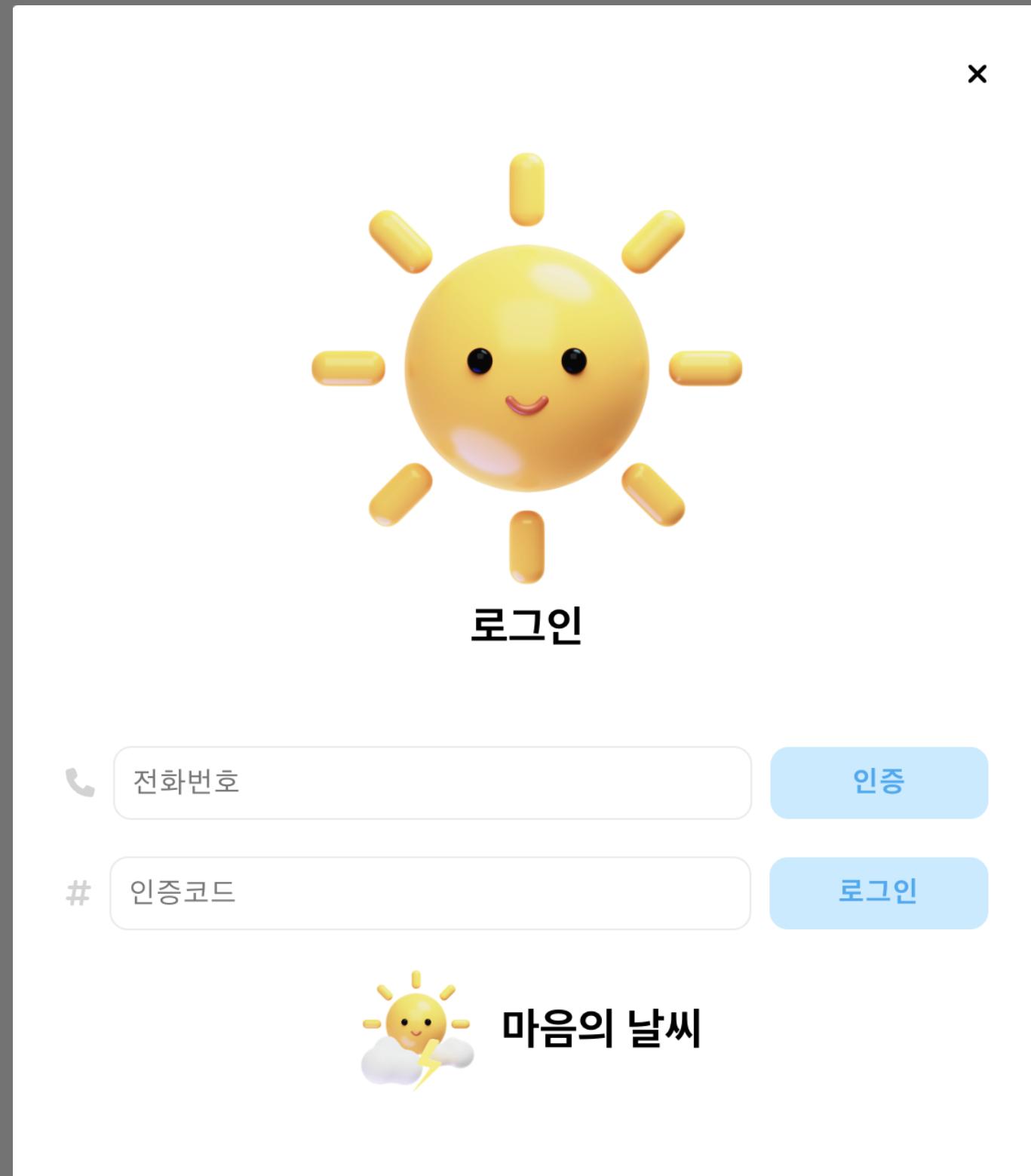
분석된 일기 내용...

달리기 * # 달리기 * # 달리기 *

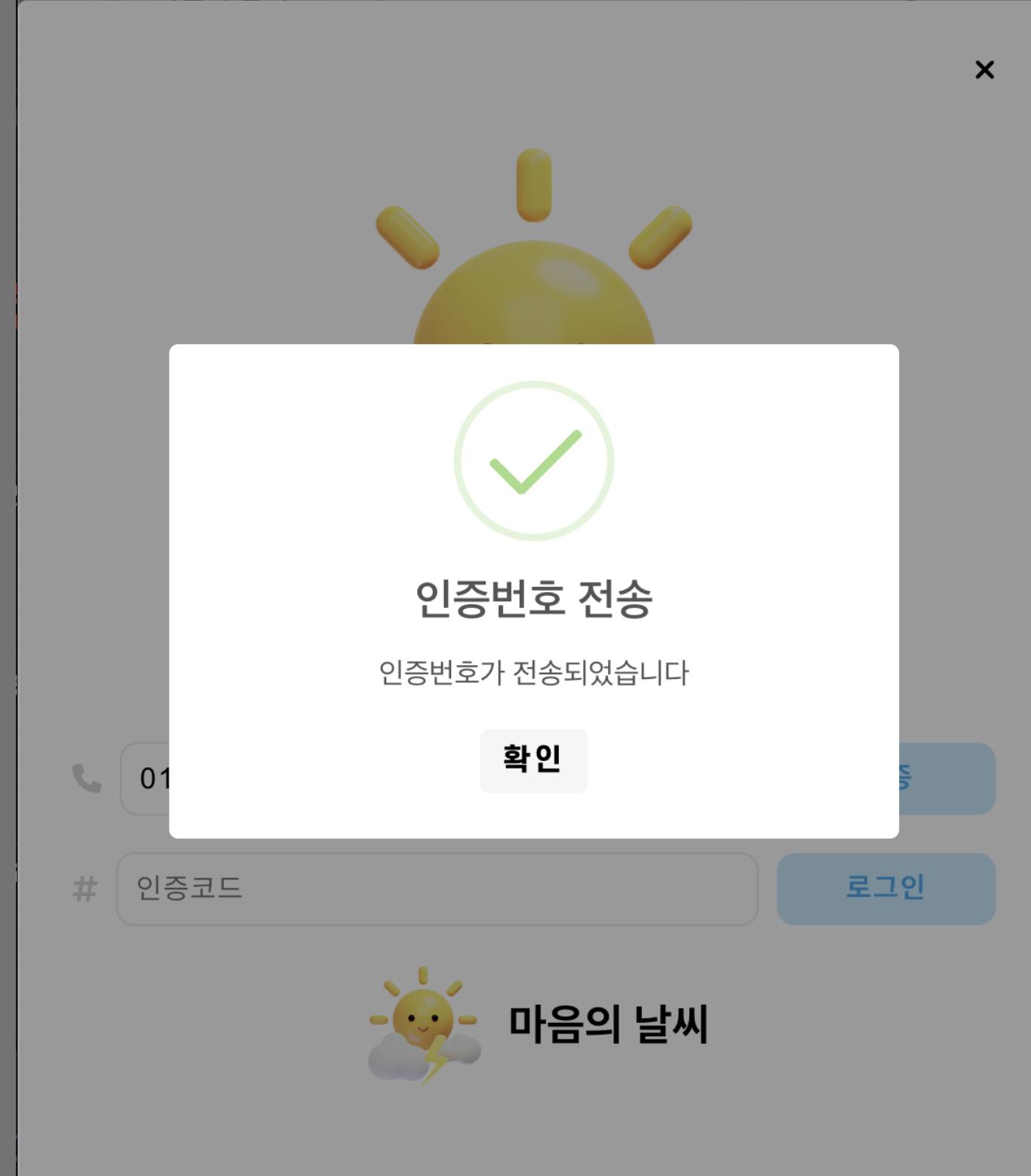
심리상담 보기 수정 닫기



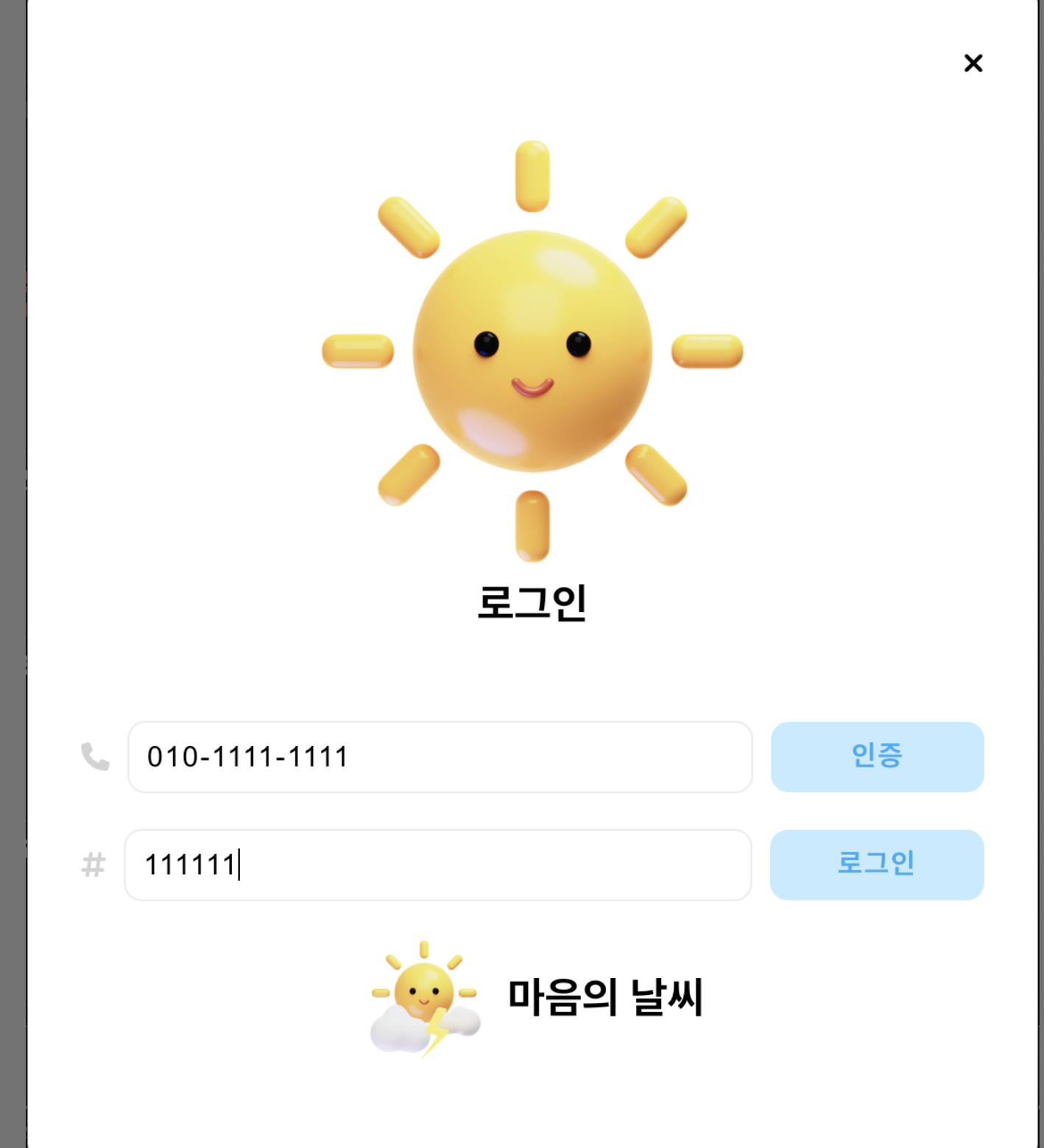
프론트엔드 로그인 창



전화번호 로그인

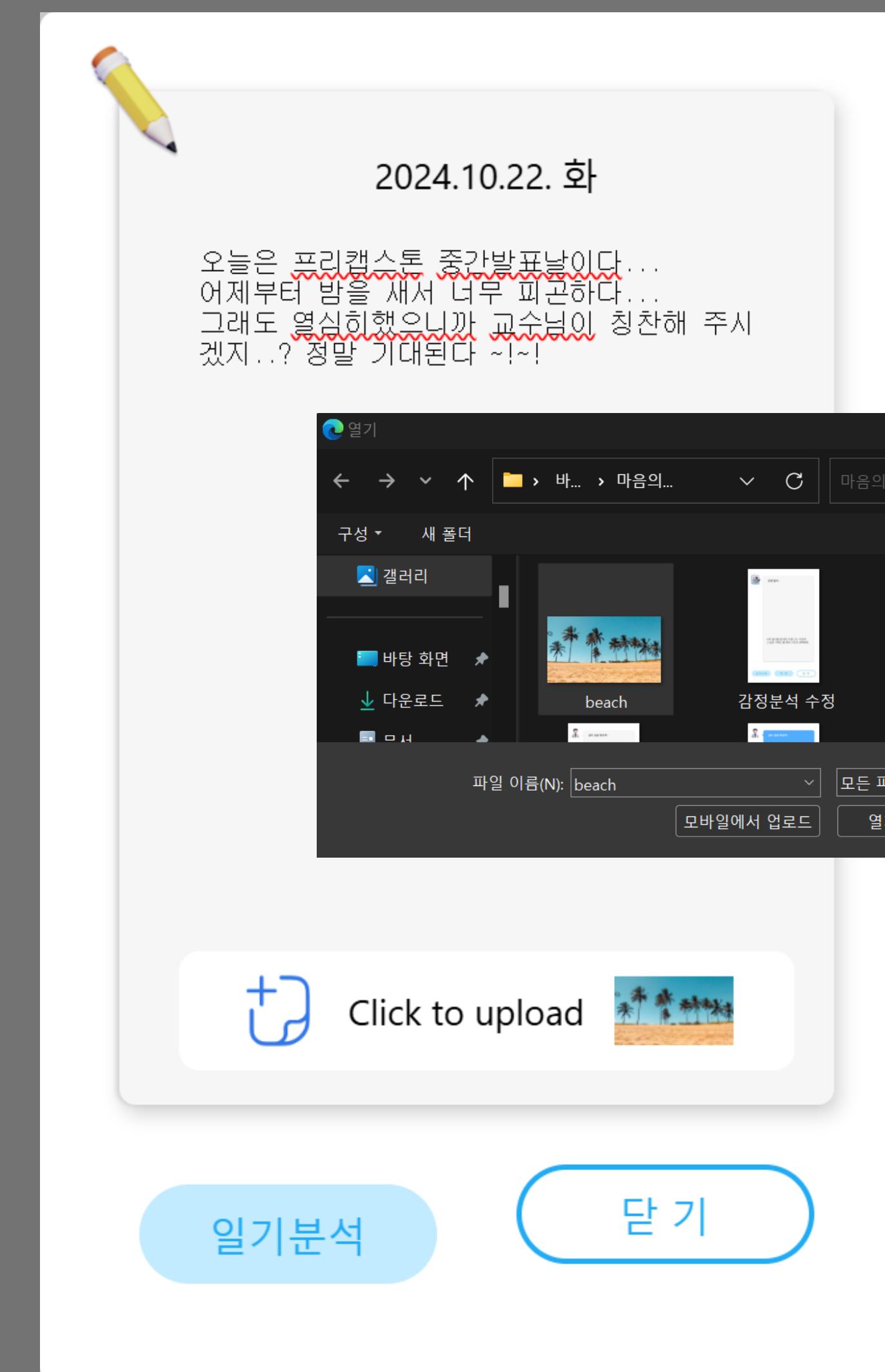
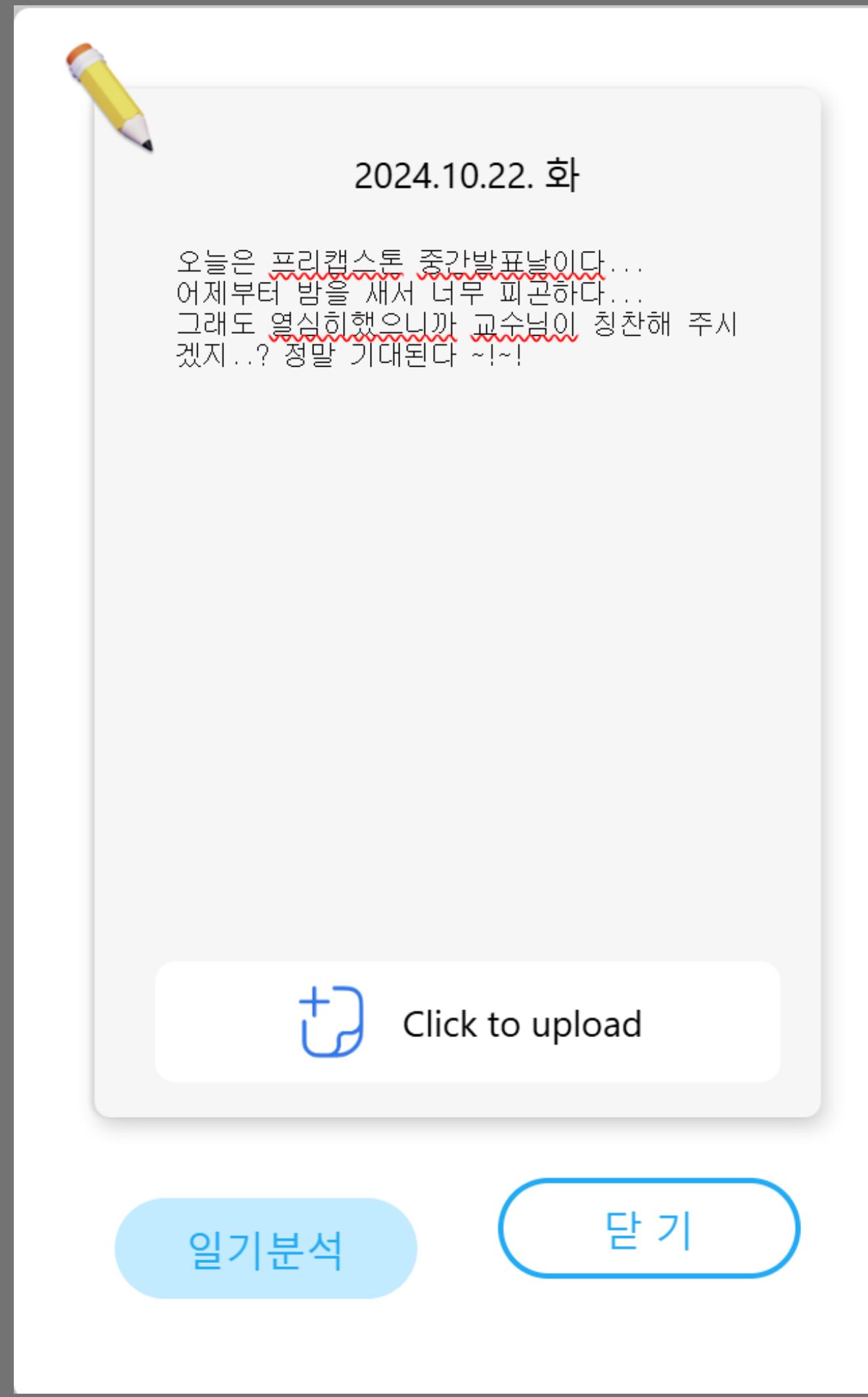
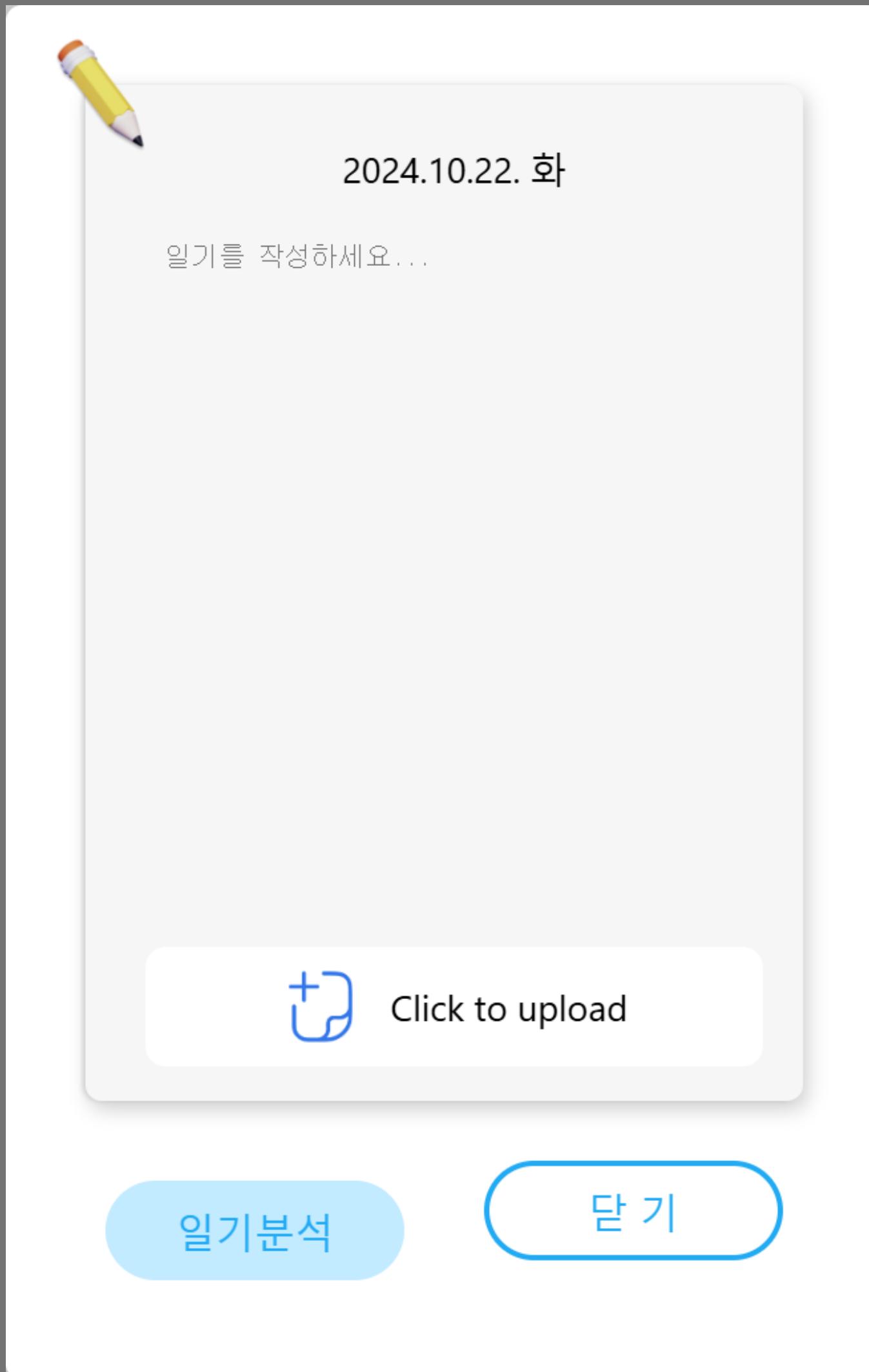


인증번호 전송



인증번호 입력 및 로그인

일기생성 창

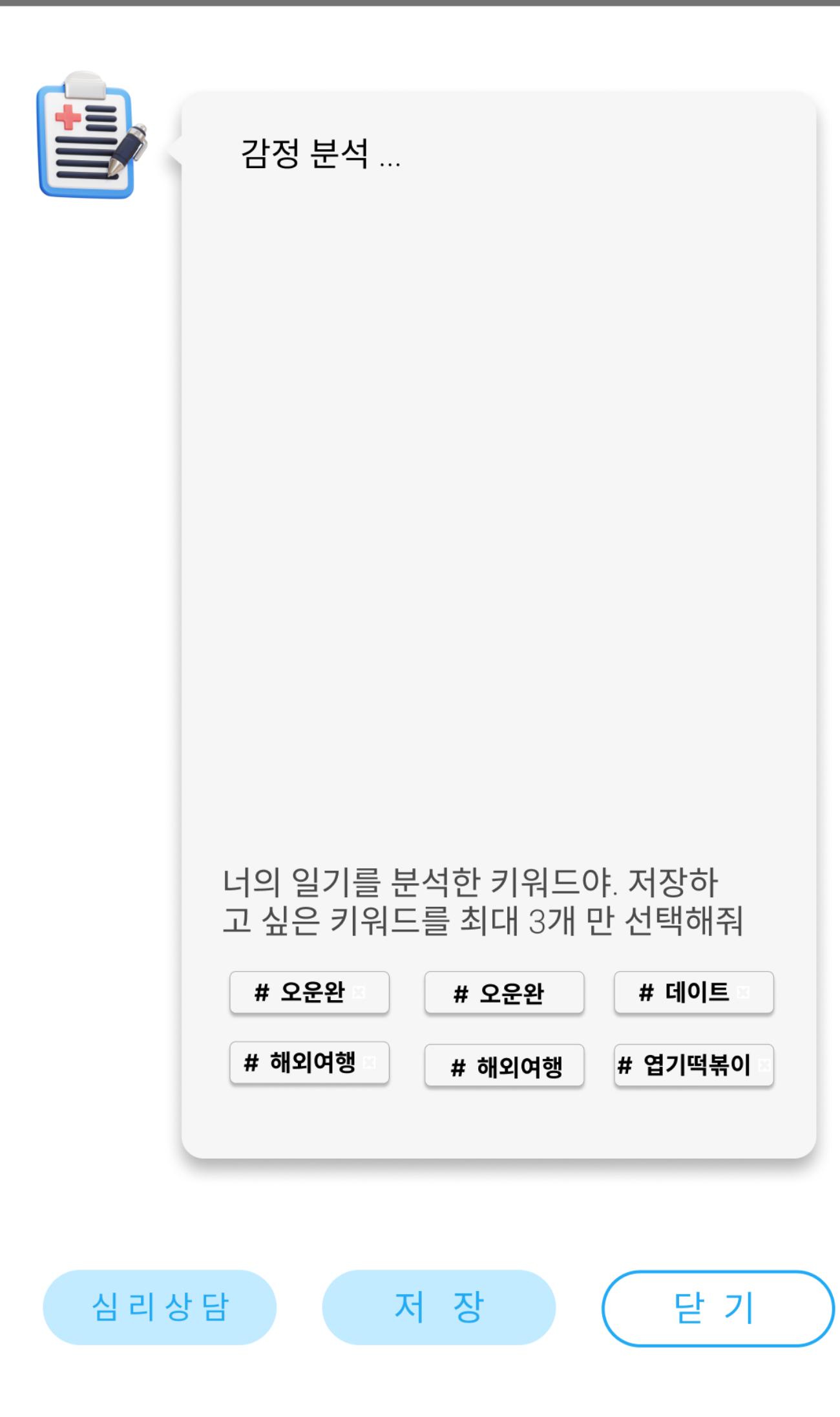


일기생성 창

텍스트 작성

사진 업로드

일기분석 창

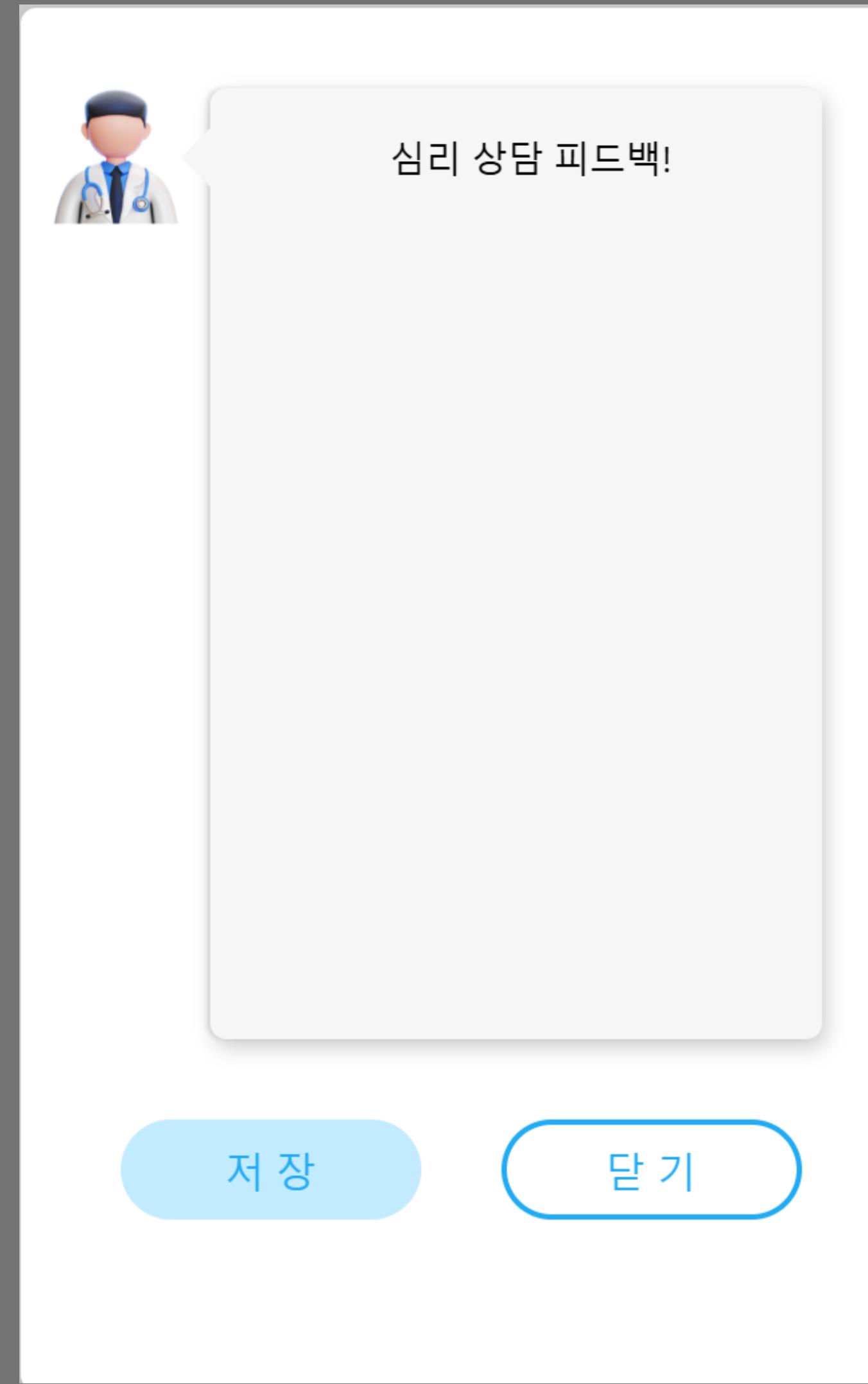


일기로 부터 감정 분석

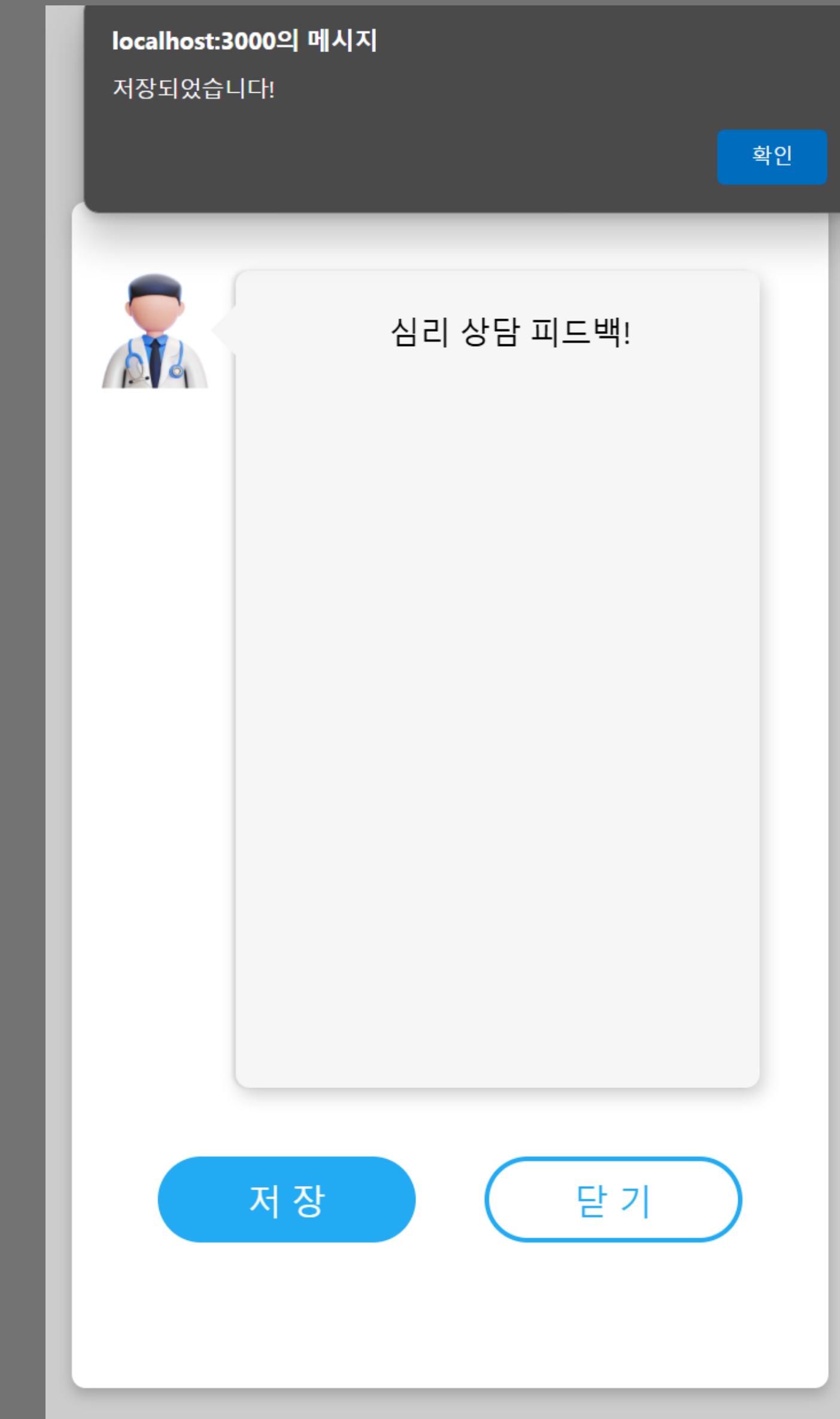


해시태그 선택

일기상담 창



일기 상담 창

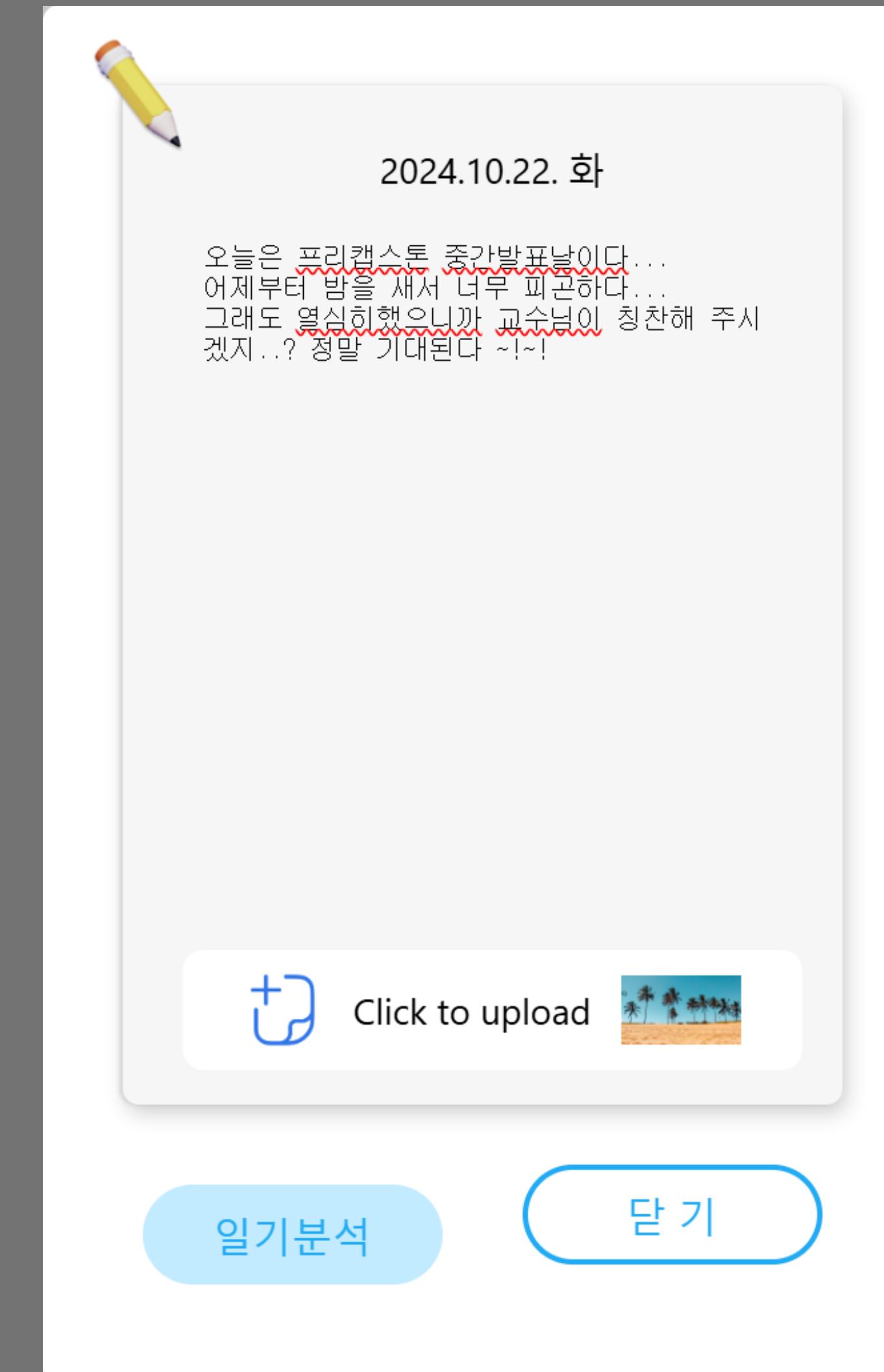


저장 메시지

상세일기 창



텍스트 작성



일기 생성 창에서 수정

마음의 날씨 : 감정 기록 플랫폼

발견된 문제 및 해결 과정

```
▶ {message: '기존 사용자입니다.', firebase_uid: 'kga5uJMTKeP6pVHT89vl7ofLzLv1'}
▶ {location: 'seoul', weather: {...}, message: 'seoul의 날씨를 성공적으로 조회했습니다.'}
▶ {message: '기존 사용자입니다.', firebase_uid: 'kga5uJMTKeP6pVHT89vl7ofLzLv1'}
▶ {location: 'seoul', weather: {...}, message: 'seoul의 날씨를 성공적으로 조회했습니다.'}
일기 내용: asdasdsadasd
업로드된 파일: null
asdasdsadasd 2024-11-26 seoul
▶ POST http://localhost:3000/api/diaries/create 500 (Internal Server Error)
▶ Failed to create diary:
▶ AxiosError {message: 'Request failed with status code 500', name: 'AxiosError',
MLHttpRequest, ...}
▶ {}
▶ POST http://localhost:3000/api/openai/analyze 400 (Bad Request)
```

너의 일기를 분석한 키워드야. 저장하고 싶은 키워드를 최대 3개만 선택해줘.



일기 분석 오류

일기 분석 중 오류가 발생했습니다

확인

Thank you

발표 _ 김동현
PPT _ 남궁민, 이민주, 임하늘