

SW상세설계서 - 3팀(불타는 감자)

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

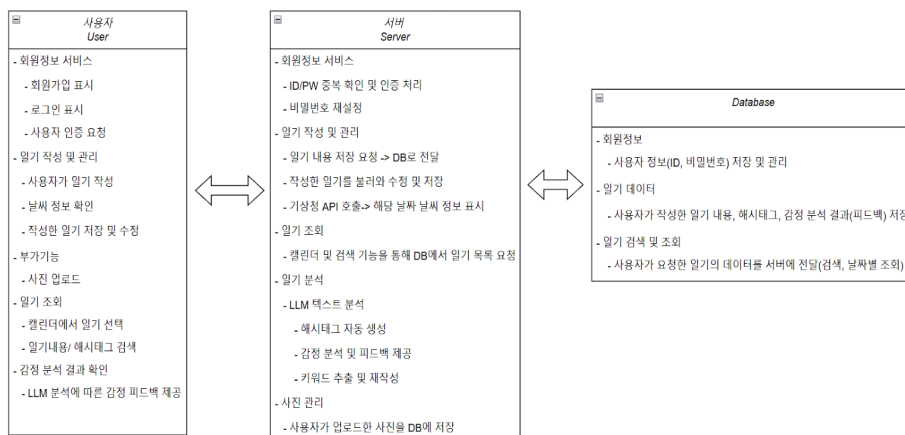
- 기능구조도 문서
- 시스템 구조도 문서
- 각각의 API관련 문서
 - OpenWeatherAPI 문서
 - LLM API문서
 - Firebase 인증 및 데이터베이스 문서
 - React, Node.js, MySQL 개발 문서 및 가이드

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 아키텍처

- 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

기능 설명: 회원가입, 로그인 입력

(Input): 사용자 정보 (전화번호) 출력

(Output): 등록 성공 여부, 로그인 토큰

프로세스 흐름:

- 사용자가 회원 가입 정보를 제출
- 입력된 정보를 firebase auth에 저장
- firebase auth 에서 phone authentication

주요 클래스 및 메서드:

MemberController

- login(String phoneNumber, String code): 로그인 기능
- validateUserCredentials(): 사용자 자격 검증
- save(UserEntity user): 사용자 정보 저장

3.2. 일기 조회 모듈(검색바, 캘린더 클릭)

기능 설명: 일기 검색, 캘린더 날짜 클릭시 일기 조회

(Input): 해시태그(검색바) , 날짜(캘린더)

(Output): 일기 목록, 선택한 일기 상세 정보

프로세스 흐름:

- 사용자가 선택한 날짜의 일기 상세 정보 표시
- 해시태그로 일기 검색 시 해당 일기가 있는 캘린더 날짜 버튼 활성화
 - ex) '카페' 해시태그로 검색하면 캘린더에서 '카페'를 포함한 캘린더 날짜 버튼들만 활성화

주요 클래스 및 메서드:

DiaryController

- getDiaryByDate(LocalDate date): 선택 날짜의 일기 조회
- searchDiaryByTag(String tag): 해시태그로 일기 검색

DiaryService

- getAllDiaries(): 모든 일기 조회
- getDiaryById(int diaryId): 일기 상세 조회

3.3. 일기 작성 및 관리 시스템 모듈

기능 설명: 일기 작성, 일기 정보 수정 및 삭제

(Input): 일기 내용, 날짜, 해시태그

(Output): 일기 작성 성공 여부

프로세스 흐름:

- 사용자가 일기를 작성 입력된 정보를 데이터베이스에 저장

주요 클래스 및 메서드:

DiaryWriteController

- addDiary(DiaryDto diary): 일기 생성
- updateDiary(int diaryId, DiaryDto diary): 일기 수정
- deleteDiary(int diaryId): 일기 삭제

DiaryWriteRepository

- save(DiaryEntity diary): 일기 저장

3.4. 일기 분석 기능 모듈

기능 설명: 일기 텍스트 분석, 감정 분석

(Input): 일기 내용

(Output): 분석 결과

프로세스 흐름:

- 사용자가 작성한 일기를 분석 해시태그 자동 생성
- 감정 피드백 텍스트 생성(줄 글 형식, 챗봇형식 x)
- 분석 결과를 사용자에게 제공

주요 클래스 및 메서드:

AnalysisController

- analyzeDiary(int diaryId): 일기 분석

AnalysisService

- performSentimentAnalysis(String content): 감정 분석

AnalysisRepository

- saveAnalysisResult(AnalysisResultEntity result): 분석 결과 저장

3.5. 부가기능

기능 설명: 사진 업로드, 날씨 정보표시

(Input): 이미지 파일

(Output): 업로드 성공 여부, 날씨 정보

프로세스 흐름:

- 사용자가 사진을 업로드 업로드된 이미지를 서버에 저장

주요 클래스 및 메서드:

PhotoUploadController

- uploadPhoto(PhotoDto photo): 사진 업로드

PhotoUploadService

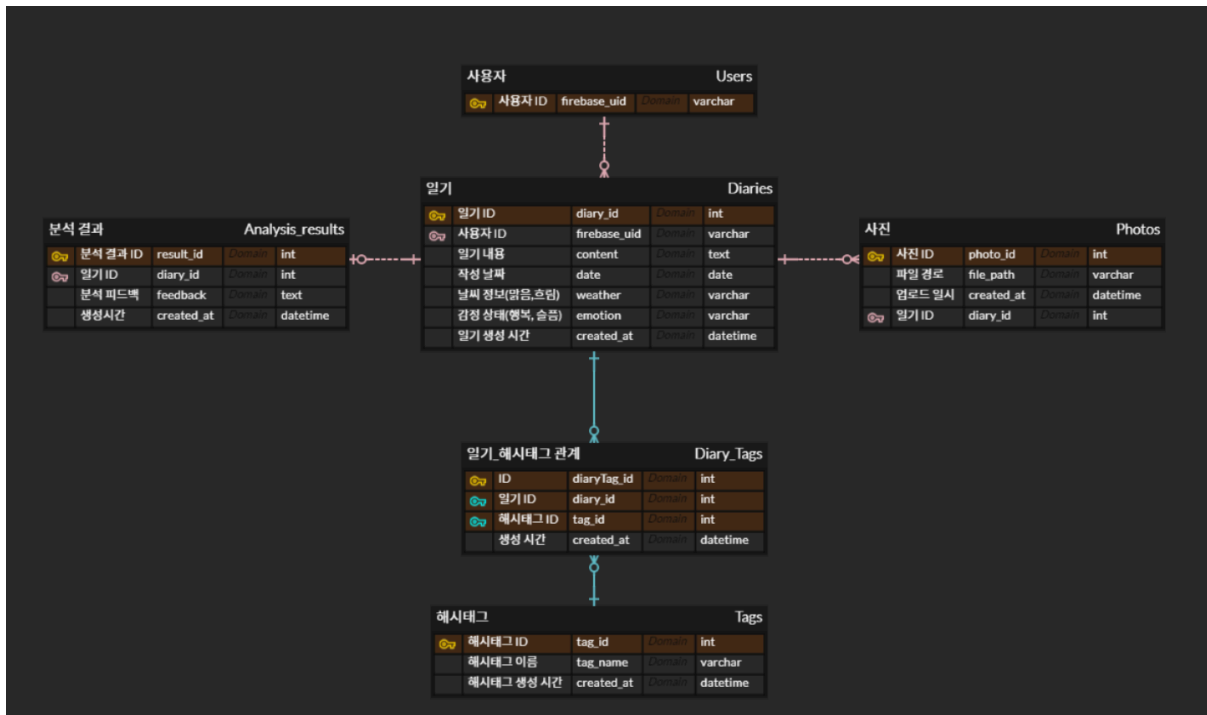
- savePhoto(): 사진 저장

WeatherService

- fetchWeatherData(String location): 날씨 데이터 API 호출
- parseWeatherData(response): API 응답을 파싱하여 필요한 정보 추출

4. 데이터베이스 설계

4.1 ER 다이어그램



4.2 테이블 정의

사용자 테이블: **users**

컬럼:

firebase_uid (VARCHAR) - Firebase 인증을 통한 사용자 고유 ID

일기 테이블 (Diary Table): **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: **date** 컬럼에 인덱스 설정

해시태그 테이블 (Tag Table): **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): **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): **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): **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)

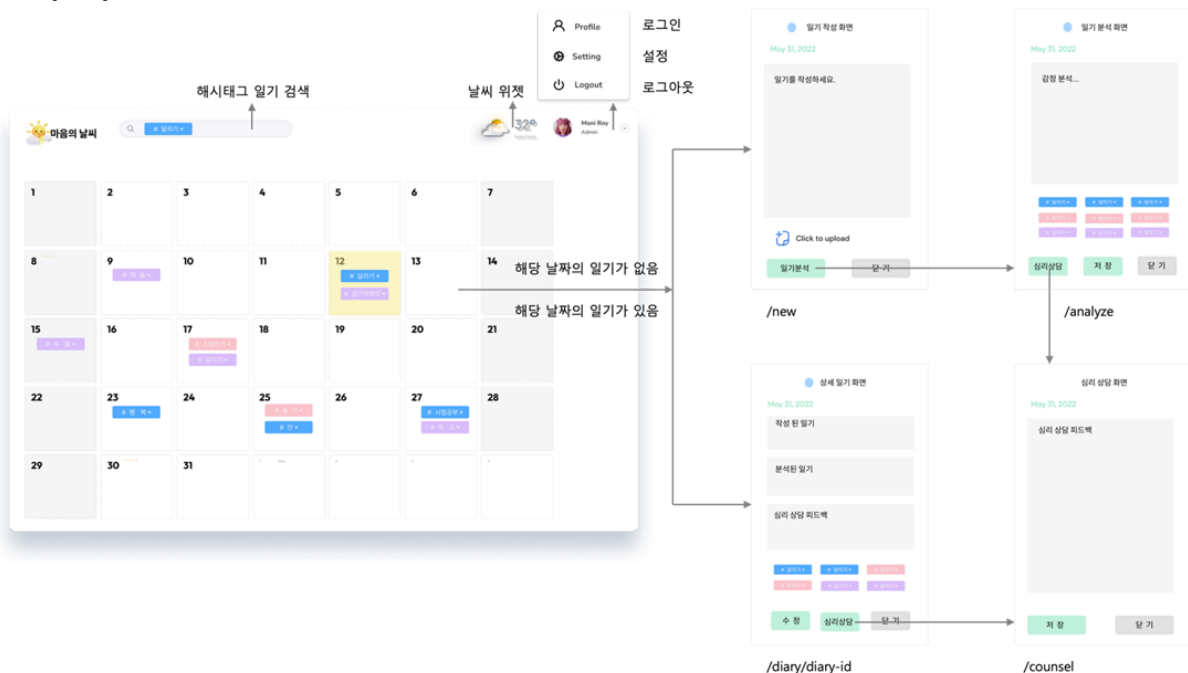
4.3 테이블간 관계 정리

- 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 있다.
 - 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 있다.
 - 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 있다.
 - 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)

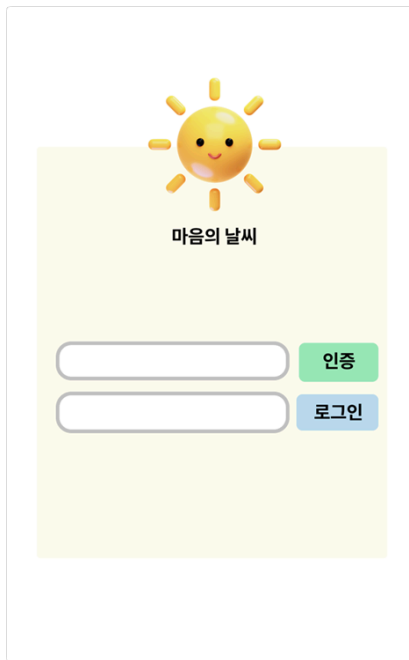
5. 사용자 인터페이스

5.1 화면구조도



- React 라우터를 사용한 SPA(Single Page Application)으로 클라이언트 사이트 렌더링이 되도록 그림과 같이 페이지 구성
- 현재 URL 경로에 맞는 Route 컴포넌트를 페이지에 렌더링하며 메인 화면(캘린더), 로그인 화면, 날씨 정보 화면, 일기 작성 화면, 일기 분석 화면, 상세 일기 화면, 심리 상담 화면으로 구성

5.2 주요화면 설계

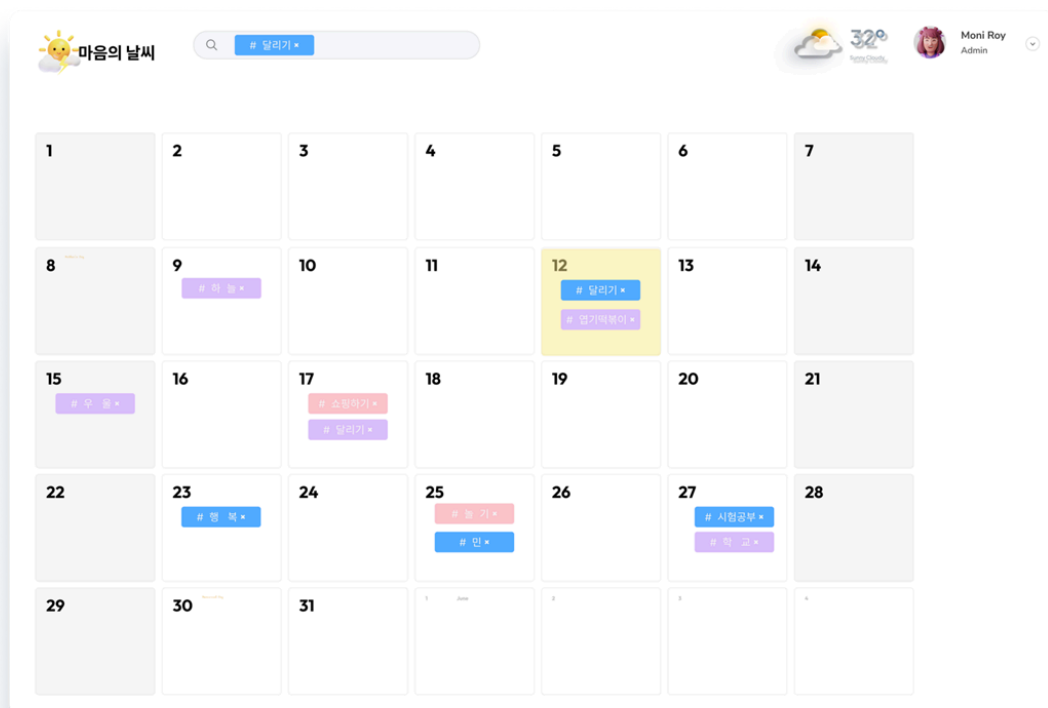


5.2.1 로그인 화면

[전화번호 필드]: 사용자 전화번호 입력(하이픈 제외)

[인증번호 필드]: 인증번호 6자리 입력

[로그인 버튼]: 로그인



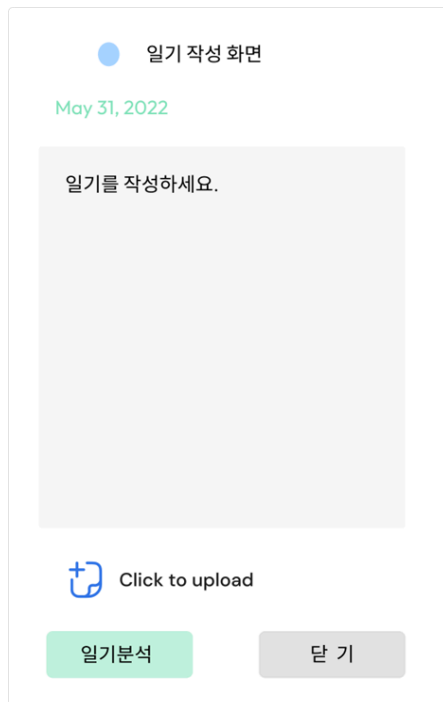
5.2.2 메인 화면(캘린더)

[검색 바]: 해시 태그로 일기를 검색할 수 있으며 해당되는 일기가 캘린더에서 하이라이트 처리됨

[날씨 위젯]: 실시간 날씨 정보 요약 표시

[프로필 메뉴]: 현재 로그인한 사용자 정보를 표시하며 로그인 메뉴 제공

[캘린더]: 작성한 일기를 해시태그로 요약해서 보여줌



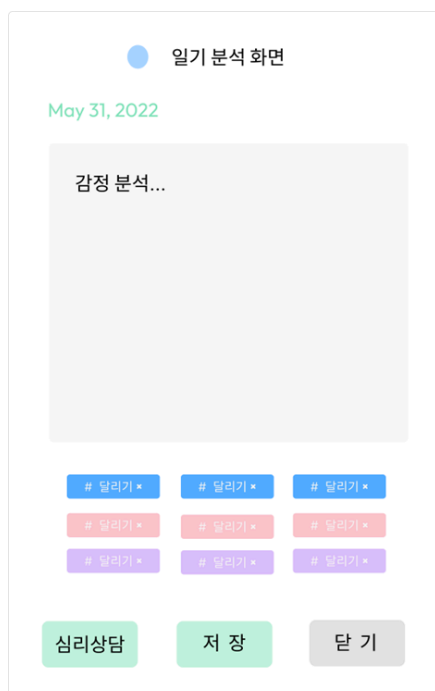
5.2.4 일기 작성 화면

[입력 필드]: 사용자가 일기를 자유롭게 작성

[사진 업로드 버튼]: 사용자가 일기에 첨부할 사진을 가져옴

[일기 분석 버튼]: 작성한 일기를 LLM이 분석하여 일기 분석 화면으로 데이터를 전달하고 이동

[닫기 버튼]: 이전 페이지로 돌아갈지 물어보고 돌아감



5.2.5 일기 분석 화면

[감정 분석 결과]: 사용자가 작성한 일기에서 감정 분석 결과를 보여줌

[해시태그]: 사용자가 작성한 일기에서 주요 해시태그를 추출해서 보여줌

[심리 상담 버튼]: 해당 일기에서 LLM이 심리를 분석하여 심리 상담 화면으로 데이터를 전달하고 이동

[저장 버튼]: 해당 일기 데이터를 저장하고 캘린더 화면으로 돌아옴

[닫기 버튼]: 이전 페이지로 돌아갈지 물어보고 돌아감

심리 상담 화면

May 31, 2022

심리 상담 피드백

저 장 닫 기

5.2.6 심리 상담 화면

[심리 상담 피드백]: 사용자 감정이 분석된 일기에서 생성된 심리 상담 내용 표시

[저장 버튼]: 해당 심리 상담 데이터를 일기 데이터에 추가하여 저장

[닫기 버튼]: 이전 페이지로 돌아올지 물어보고 돌아감

● 상세 일기 화면

May 31, 2022

작성된 일기

분석된 일기

심리 상담 피드백

달리기 * # 달리기 * # 달리기 *

달리기 * # 달리기 * # 달리기 *

수 정 심리상담 닫 기

5.2.7 일기 상세 화면

[일기 내용]: 작성된 일기 텍스트를 표시

[분석 내용]: 감정 분석 결과를 표시

[심리 상담 피드백]: 심리 상담 내용을 표시

[수정 버튼]: 일기를 수정할 수 있는 페이지로 이동

[심리 상담 버튼]: 해당 일기에서 LLM이 심리를 분석하여 심리 상담 화면으로 데이터를 전달하고 이동

[닫기 버튼]: 이전 페이지로 돌아옴

5.3 시퀀스 다이어그램

5.3.1 사용자 등록 시퀀스

- 사용자 -> 로그인 폼: 입력 후 제출
- 로그인 양식 -> MemberController: 폼 데이터 전송
- MemberController -> MemberService: 비즈니스 로직 처리 요청

- MemberService -> MemberRepository: 사용자 데이터 저장 요청
- MemberRepository -> 데이터베이스: 사용자 정보 저장
- 데이터베이스 -> MemberRepository: 저장 완료 응답
- MemberRepository -> MemberService: 성공 메시지 반환
- MemberService -> MemberController: 성공 메시지 반환
- MemberController -> 회원가입 양식: 성공 메시지 표시

5.3.2 일기 작성 시퀀스

- 사용자 -> 일기 작성 화면: 일기 내용 입력 후 저장 버튼 클릭
- 일기 작성 화면 -> DiaryWriteController: 일기 내용 전송
- DiaryWriteController -> DiaryWriteService: 일기 저장 요청
- DiaryWriteService -> DiaryWriteRepository: 일기 데이터 저장 요청
- DiaryWriteRepository -> 데이터베이스: 일기 정보 저장
- 데이터베이스 -> DiaryWriteRepository: 저장 완료 응답
- DiaryWriteRepository -> DiaryWriteService: 성공 메시지 반환
- DiaryWriteService -> DiaryWriteController: 성공 메시지 반환
- DiaryWriteController -> 일기 작성 화면: 성공 메시지 표시

5.3.3 일기 분석 시퀀스

- 사용자 -> 일기 분석 화면: 분석 버튼 클릭
- 일기 분석 화면 -> AnalysisService: 일기 내용 분석 요청
- AnalysisService -> LLM: 분석 요청
- LLM -> AnalysisService: 분석 결과 반환
- AnalysisService -> 일기 분석 화면: 분석 결과 전달
- 일기 분석 화면 -> 사용자: 분석 결과 표시

5.3.4 해시 태그 저장 시퀀스

- 사용자 -> 일기 분석 화면: 해시태그 선택 후 저장 버튼 클릭
- 일기 분석 화면 -> DiaryService: 해시태그 저장 요청
- DiaryService -> DiaryWriteRepository: 해시태그 데이터 저장 요청
- DiaryWriteRepository -> 데이터베이스: 해시태그 정보 저장
- 데이터베이스 -> DiaryWriteRepository: 저장 완료 응답
- DiaryWriteRepository -> DiaryService: 성공 메시지 반환
- DiaryService -> 일기 분석 화면: 성공 메시지 반환
- 일기 분석 화면 -> 사용자: 성공 메시지 표시

5.3.5 심리 상담 시퀀스

- 사용자 -> 심리 상담 화면: 심리 상담 버튼 클릭
- 심리 상담 화면 -> AnalysisService: 분석된 일기를 기반으로 상담 내용 요청
- AnalysisService -> LLM: 상담 요청
- LLM -> AnalysisService: 상담 내용 반환
- AnalysisService -> 심리 상담 화면: 상담 내용 전달
- 심리 상담 화면 -> 사용자: 상담 내용 표시
- 사용자 -> 심리 상담 화면: 저장 버튼 클릭
- 심리 상담 화면 -> DiaryService: 상담 내용 저장 요청
- DiaryService -> DiaryWriteRepository: 상담 내용 저장 요청
- DiaryWriteRepository -> 데이터베이스: 상담 정보 저장
- 데이터베이스 -> DiaryWriteRepository: 저장 완료 응답
- DiaryWriteRepository -> DiaryService: 성공 메시지 반환
- DiaryService -> 심리 상담 화면: 성공 메시지 반환

- 심리 상담 화면 -> 사용자: 성공 메시지 표시

5.3.6 해시태그 검색 시퀀스

- 사용자 -> 검색바: 해시태그 입력
- 검색바 -> CalendarService: 해시태그 검색 요청
- CalendarService -> DiaryRepository: 해시태그에 해당하는 날짜 요청
- DiaryRepository -> 데이터베이스: 해당 해시태그 검색
- 데이터베이스 -> DiaryRepository: 검색 결과 반환
- DiaryRepository -> CalendarService: 결과 반환
- CalendarService -> 검색바: 하이라이팅된 날짜 응답
- 검색바 -> 캘린더 화면: 하이라이팅된 날짜 표시

6. API 설계 및 권한 관리

Firebase: 실시간 데이터 처리와 인증, 그리고 서버리스 작업에 활용.

- 인증 관리: Firebase Authentication을 사용해 로그인/회원 가입을 처리할 수 있습니다.
- 실시간 데이터: 감정 분석 결과나 키워드 추출과 같은 데이터는 즉시 반영되어야 하므로, Firebase에서 처리.
- Cloud Functions: LLM API 호출, 감정 분석, 키워드 추출과 같은 기능을 트리거하여 서버리스 방식으로 실행.

Firebase Authentication 사용 : Firebase를 사용해 회원 가입과 로그인을 처리

<회원 가입 코드>

```
import { createUserWithEmailAndPassword } from "firebase/auth";
import { auth } from "../firebase";

const signUp = (email, password) => {
  createUserWithEmailAndPassword(auth, email, password)
    .then((userCredential) => {
      const user = userCredential.user;
      console.log('회원가입 성공:', user);
    })
    .catch((error) => {
      console.error('회원가입 오류:', error.message);
    });
};
```

<로그인 코드>

```
import { signInWithEmailAndPassword } from "firebase/auth";
import { auth } from "../firebase";

const login = (email, password) => {
  signInWithEmailAndPassword(auth, email, password)
```

```

.then((userCredential) => {
  const user = userCredential.user;
  console.log('로그인 성공:', user);
})
.catch((error) => {
  console.error('로그인 오류:', error.message);
});
};

```

<로그아웃 코드>

```

firebase.auth().signOut().then(() => {
  console.log("로그아웃 성공");
}).catch((error) => {
  console.error("로그아웃 오류:", error);
});

```

React 와 OpenWeather API 연결

<위치 정보 가져오기>

```

const getCurrentLocation=()=>{
  navigator.geolocation.getCurrentPosition((position)=>{
    let lat = position.coords.latitude
    let lon = position.coords.longitude
    getWeatherByCurrentLocation(lat, lon)
  })
}

```

<OpenWeather API 키 발급받은 후 위치와 연동>

```

const getWeatherByCurrentLocation= async (lat, lon)=>{
  let url =
`https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lon}&appid=본인API키`
  let response = await fetch(url)
  let data = await response.json();
  console.log(data);
}

```

7.에러 처리 및 로깅

Firebase에서 발생할 수 있는 일반적인 에러

- 인증 오류: 잘못된 로그인 정보나 사용자 계정이 없는 경우.
- 권한 오류: 사용자가 요청한 데이터에 접근할 권한이 없을 때.
- 데이터베이스 오류: 쿼리 실패, 데이터 구조 불일치.

- 네트워크 오류: 안정적이지 않은 인터넷 연결로 인해 발생하는 오류.

<인증 오류 처리 예시>

```
firebase.auth().signInWithEmailAndPassword(email, password)
  .then((userCredential) => {
    // 로그인 성공
  })
  .catch((error) => {
    const errorCode = error.code;
    const errorMessage = error.message;
    console.error('Authentication error:', errorCode, errorMessage);
    // 사용자에게 에러 메시지 표시
    alert("로그인 오류: " + errorMessage);
  });
```

<데이터베이스 오류 처리 예시>

```
const db = firebase.firestore();

db.collection("users").doc("user_id").get()
  .then((doc) => {
    if (doc.exists) {
      // 데이터 처리
    } else {
      console.error("No such document!");
    }
  })
  .catch((error) => {
    console.error("Error getting document:", error);
    // 사용자에게 에러 메시지 표시
    alert("데이터베이스 오류: " + error.message);
  });
```

로깅 시스템: Firebase Crashlytics 사용

1. Firebase Console에서 Crashlytics 활성화.
2. 앱에 Firebase SDK 설치 및 초기화.
3. **log** 메서드를 사용해 사용자 정의 메시지 기록.

```
import { getCrashlytics } from 'firebase/crashlytics';

const crashlytics = getCrashlytics(app);

// 에러 발생 시 로그 남기기
crashlytics.log('Custom log message');

// 사용자 정의 에러 처리
```

```
crashlytics.recordError(new Error('Test error'));
```

OpenWeather API 사용시 에러처리 경우

- 위치정보 에러 처리: 에러 콜백 함수를 추가하여 위치 정보를 가져오지 못할 경우 에러 메시지를 표시합니다.
- API 에러 처리: API 호출 중 HTTP 에러가 발생하거나 fetch에 실패할 경우 에러 메시지를 표시합니다

```
try {
  setLoading(true); // API 호출 시작 시 로딩 상태 표시
  ....

  if (!response.ok) {
    throw new Error(`HTTP error! status: ${response.status}`);
  }
  ....
} catch (error) {
  setError(`Failed to fetch weather data: ${error.message}`);
} finally {
  ....
}
};
```

로깅시스템

```
console.error('Weather API Error:', error); // 에러를 콘솔에 로깅
```

8. 보안설계

- 인증 및 권한 관리: Firebase Authentication을 사용한 사용자 인증. 이메일/비밀번호, Google 및 소셜 로그인 등 다양한 인증 방식을 제공. 권한 관리는 Firebase Firestore의 사용자 권한 필드를 이용하여 구현.
- 암호화: Firebase Authentication에서 제공하는 암호화 방식(예: 비밀번호 해시 처리)을 그대로 사용하며, 추가적으로 중요한 데이터는 Firebase에서 제공하는 보안 규칙(Firebase Security Rules)을 활용해 보호.

9. 테스트 계획

- 유닛 테스트: React 컴포넌트 및 Firebase Authentication 연동에 대한 유닛 테스트 계획. Jest와 React Testing Library로 UI 및 비즈니스 로직 테스트.
- 통합 테스트: Cypress로 Firebase와의 통합 테스트 계획, E2E 테스트로 인증 흐름과 데이터베이스 연동 확인.
- 테스트 도구: Jest, React Testing Library로 유닛 테스트, Cypress로 E2E 테스트 수행.

10. 배포계획

- 배포 환경: Firebase Hosting을 사용하여 웹 애플리케이션 배포. AWS EC2와 같은 외부 서버 환경이 필요한 경우 병행 사용 가능.
- 배포 자동화: GitHub Actions와 Firebase CLI로 CI/CD 설정, Firebase Hosting에 자동 배포.