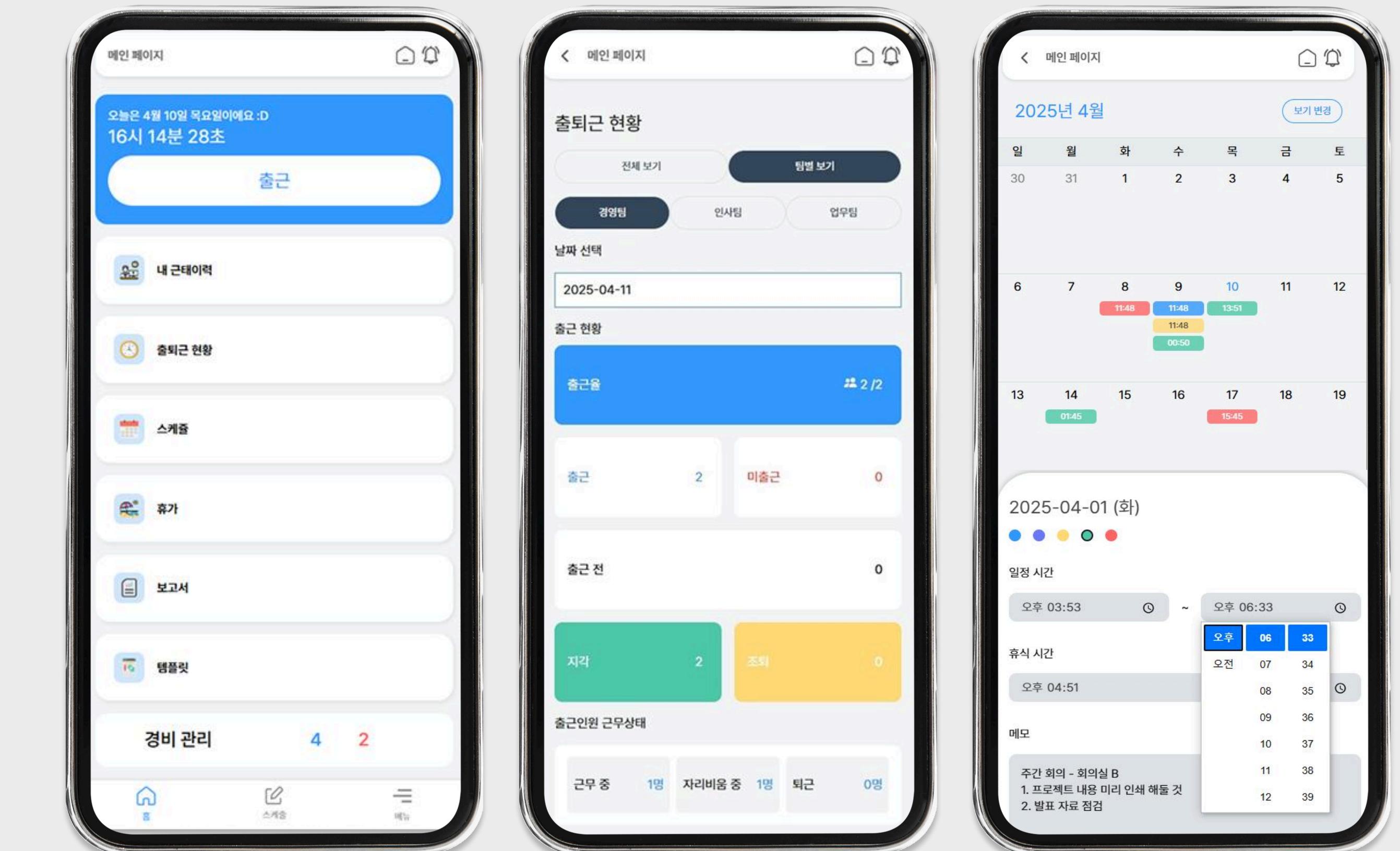


Full Stack Develop



근태 관리 웹/모바일 어플리케이션

김성우 · 최수민 · 박시진
손수용 · 오예준 · 이영현



00. 목차

01. 프로젝트 개요

02. 프로젝트 상세

03. 프로젝트 시연

04. 코드 리뷰

05. 프로젝트 회고

01. 프로젝트 개요

1-1. 기획 의도

1-2. 개발 목표

1-3. 개발 환경

1-4. 프로젝트 구조도

1-5. 팀원소개

01. 프로젝트 개요

1-1. 기획 의도

1-2. 개발 목표

1-3. 프로젝트 구조도

1-4. 프로젝트 개발일정

1-5. 팀원소개

● 기존 수동식 출퇴근 기록의 불편함



- 출결, 근태 관리 설문조사 결과**
- 수기 작성 후 직접 보고하는 직원이 설문조사 결과의 대부분을 차지.
 - 출결 보고 내용이 누락되거나 사실과 다른 경우가 빈번하게 발생함.

- 불투명한 기록 관리**
- 변경 이력이 명확히 기록되지 않거나, 출결 데이터를 위변조하는 것이 상대적으로 용이
 - 투명성 부족과 신뢰 문제가 발생할 수 있으며, 직원 간의 갈등이나 불만을 초래함.

● 기존 수동식 출퇴근 기록의 불편함



관리자의 업무 부담 증가

- 관리자나 인사 담당자는 매일 수많은 직원들의 출퇴근 시간을 확인하고, 이를 종합하여 월별 근태를 계산해야 함.
- 이 과정에서 누락된 기록을 찾아내는데 추가적인 시간이 소요 되며, 비효율적이고 시간 낭비가 될 수 있음.



분산된 기록 관리

- 수동 기록 방식은 서로 다른 형식(종이, 엑셀, 수기 등)으로 출결을 관리하는 경우가 많음.
- 수동 기록 과정에서, 수기와 엑셀의 기록이 상이한 경우가 잦음
- 서로 다른 경로에서 기록된 데이터가 상호 연동되지 않기 때문에 전체적인 출결 현황을 파악하는 데 혼란이 있을 수 있음

01. 프로젝트 개요

1-1. 기획 의도

1-2. 개발 목표

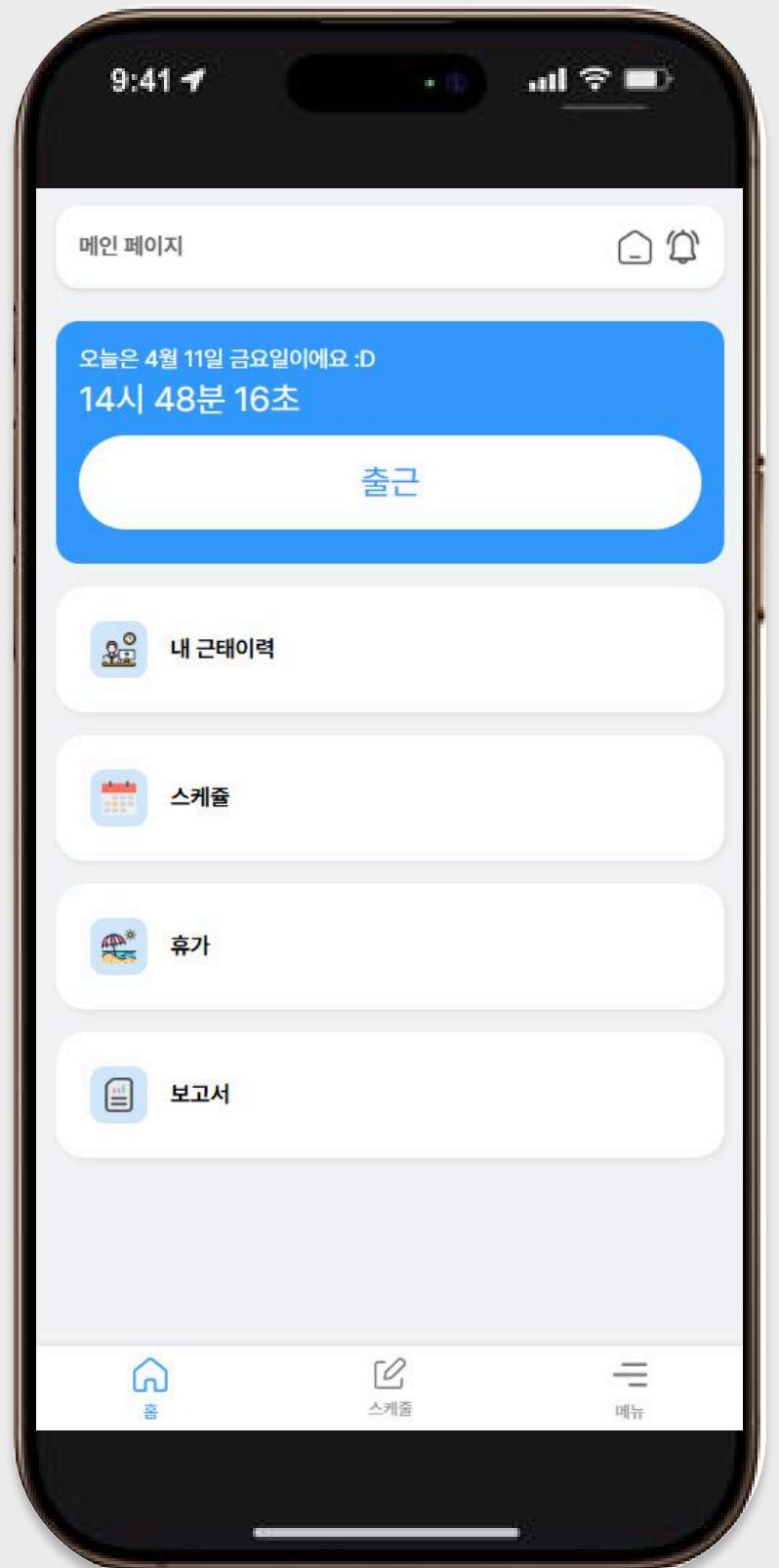
1-3. 프로젝트 구조도

1-4. 프로젝트 개발일정

1-5. 팀원소개

출퇴근과 스케줄 관리, 보고서 작성까지

어디서든 모바일로 편리하게 !



간편한 출퇴근 및 업무 시스템



정확하고 효율적인 출결 관리

- 출퇴근 시간을 실시간으로 체크인
- 관리자와 사용자는 실시간으로 출결 현황을 파악
- 출결 기록 자동화로 인한 통계를 바탕으로 불필요한 행정 업무의 최소화
- 사용자 경험을 고려한 직관적인 UI / UX 디자인



간편한 업무 관리 및 보고

- 보고서 작성과 승인 절차를 모바일에서 손쉽게 통합 관리
- 휴가를 신청하고, 승인 여부를 알림을 통해 빠르게 확인
- 스케줄 관리까지 통합된 올인원 업무 플랫폼

01. 프로젝트 개요

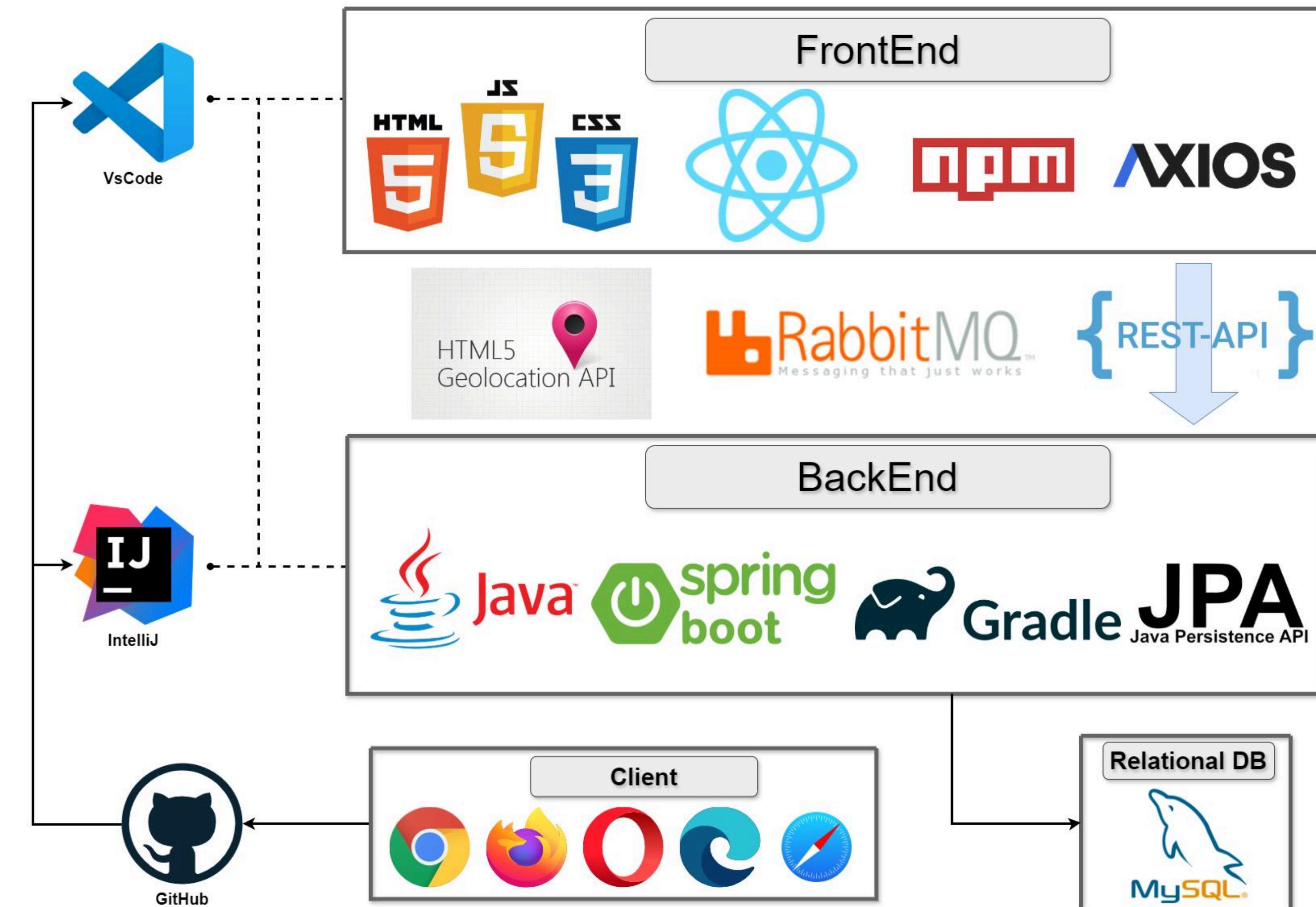
1-1. 기획 의도

1-2. 개발 목표

1-3. 프로젝트 구조도

1-4. 프로젝트 개발일정

1-5. 팀원소개



01. 프로젝트 개요

1-1. 기획 의도

1-2. 개발 목표

1-3. 프로젝트 구조도

1-4. 프로젝트 개발일정

1-5. 팀원소개

단계	작업	이름	번호	세부항목 및 산출물	시작일	종료일	작업기간
기획	현행업무 분석	공통	1	레퍼런스 서칭	3.18	3.18	2
		공통	2	현행 어플리케이션 분석			
		공통	3	기능별 업무 분담			
설계	요구사항 분석	파트별 담당자	4	파트별 요구사항 분석 후 전체 취합	3.21	3.23	3
	아키텍처 설계	공통	5	SW 아키텍처 설계서 작성			
	화면 설계	파트별 담당자	6	기능별 화면 정의 및 와이어프레임 작성			
		파트별 담당자	7	전체 UI 흐름도 구성 및 공통 디자인 패턴 정리			
	프로그램 설계	공통	8	SW 개발 명세서 작성(REST API 명세서, 테이블 명세서, UML)			
	데이터베이스 설계	파트별 담당자	9	파트별 데이터베이스 설계 및 취합			
개발 구현	Backend	김성우	10	스케줄 추가/삭제 기능 개발	3.24	3.28	5
		오예준	11	휴가 신청 및 조회 삭제 구현	3.24	3.28	5
		손수용	12	보고서 관련 엔티티(Entity) 설계 및 데이터베이스 연동	3.24	3.28	5
		이영현	13	출퇴근, 내 근태 이력, 출퇴근 현황 백엔드 기능 개발	3.24	3.31	7
		박시진	14	경비관리페이지 개발	3.24	3.31	7
		손수용	15	REST API 개발 (보고서 CRUD)	3.28	3.31	4
		최수민	16	로그인, 회원가입 기능 개발, 시큐리티 설정 및 Jwt 관련 기능 개발	3.24	4.02	10
		오예준	17	메시지 큐, 웹 소켓 활용 실시간 알림 서비스 로직 구현	3.31	4.04	5
		최수민	18	관리자 기능 개발, 마이페이지 기능 개발	3.31	4.04	5
		김성우	19	스케줄 페이지 개발	3.31	4.11	11
개발 구현	Frontend	손수용	20	보고서 관련 UI 페이지 구현	4.01	4.03	3
		박시진	21	경비관리페이지 개발	4.01	4.11	9
		이영현	22	출퇴근, 내 근태 이력, 출퇴근 현황 프론트엔드 기능 개발	4.01	4.11	9
		최수민	23	로그인, 회원가입 페이지 개발	4.04	4.08	5
		오예준	24	휴가 신청, 조회 페이지 구현	4.04	4.08	5
		손수용	25	REST API 연동 (보고서 CRUD 기능)	4.04	4.11	8
		최수민	26	관리자 페이지, 마이페이지 개발	4.08	4.11	4
		오예준	27	알림 버튼 활성화 및 알림 페이지 구현	4.08	4.11	4
테스트 및 보완	애플리케이션 테스트	공통	28	통합 테스트 및 오류 수정 및 보완	4.14	4.16	3
보고서 작성 및 애플리케이션 공유	최종결과보고서 작성 및 애플리케이션 공유	공통	29	최종 결과보고서 PPT 작성 및 최종 프로젝트 파일 공유	4.16	4.16	1

01. 프로젝트 개요

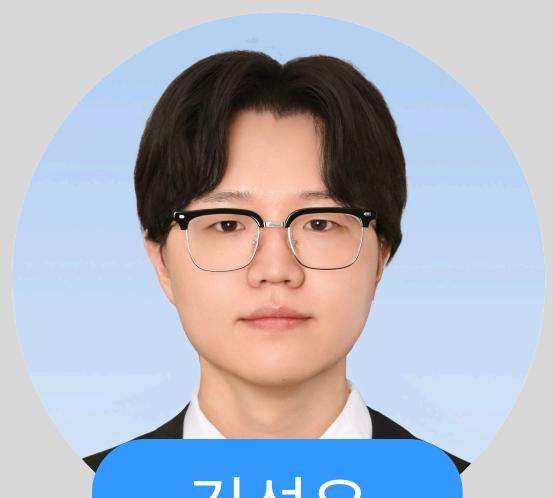
1-1. 기획 의도

1-2. 개발 목표

1-3. 프로젝트 구조도

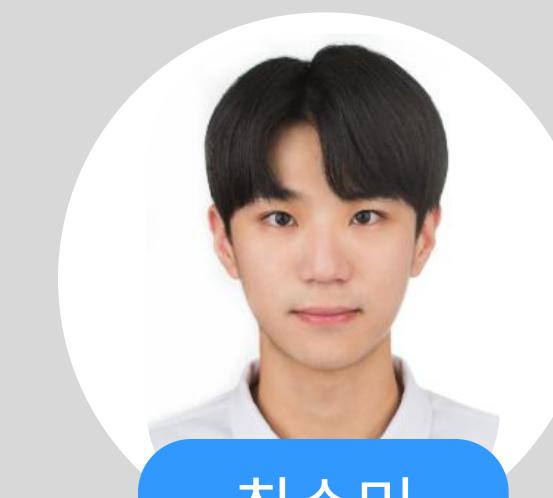
1-4. 프로젝트 개발일정

1-5. 팀원소개



프론트 / 백엔드 개발
스케줄 파트 담당
공통 스타일 CSS 제작
공용 캘린더 개발
캘린더 시스템 내 CRUD 기능 개발

김성우



프론트 / 백엔드 개발
멤버 파트 담당
JWT를 이용한 인증 시스템 개발
회원가입 신청 및 승인 기능 개발

최수민



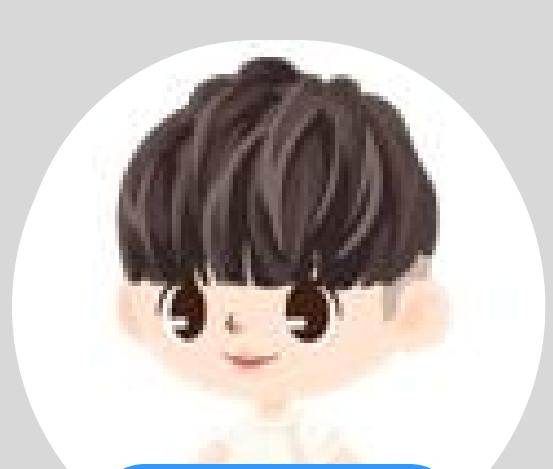
프론트 / 백엔드 개발
보고서 파트 담당
템플릿 작성 기능 개발
보고서, 댓글 CRUD 개발
드래그 앤 드롭 기능 구현

손수용



프론트 / 백엔드 개발
비용처리 파트 담당
비용처리 관련 CRUD 개발
헤더 / 푸터 제작
제미니 AI 연동 챗봇 기능 구현

박시진



프론트 / 백엔드 개발
휴가 및 알림 파트 담당
RabbitMQ 를 이용한 비동기적 실시간 알림
시스템 구현
휴가 신청 CRUD 개발

오예준



프론트 / 백엔드 개발
출결 및 근태 현황 파트 담당
Geolocation api를 이용한 GPS 기반 출퇴근
처리 기능 개발
출퇴근 현황 페이지 개발

이영현

02. 프로젝트 상세

2-1. 기능 정의서

2-2. 요구사항 정의서

2-3. UML

2-4. ERD

02. 프로젝트 상세

2-1. 기능 정의서

2-2. 요구사항 정의서

2-3. UML

2-4. ERD

출퇴근 (직원)

No	Location						상태	Page Name	Detail	
	1Depth	2Depth	3Depth	4Depth	5Depth	6Depth			Content	Description
직원										
1	홈	출퇴근 배너					상태 배너(기본)	배너	홈 > 기본 배너 '오늘은 {0월 0일 0요일} 이에요' / '[현재 시간]' / '출근' 버튼.	
2							상태 배너(근무 중)	배너	홈 > 기본 배너 > (출근 성공 시) > 근무 중 배너 '근무 중 {출근 요청 시간}~' / '자리비움' 버튼 / '퇴근' 버튼.	
3							상태 배너(자리비움 중)	배너	홈 > 근무 중 배너 > '자리비움' 버튼 클릭 > '식사' / '휴식' / '기타' 팝업 선택 시 > 자리비움 중 배너 '식사' > '자리비움 중 {식사 요청 시간}~' / '복귀' 버튼 / '퇴근' 버튼. '휴식' > '자리비움 중 {휴식 요청 시간}~' / '복귀' 버튼 / '퇴근' 버튼. '기타' > '자리비움 중 {요청 시간}~' / '복귀' 버튼 / '퇴근' 버튼.	
4							출근	버튼	클릭 시 > GPS로 현재 위치의 경도와 위도 정보를 로드. Haversine Formula로 회사 위치 기준 50m 이내면 출근 가능. (아니면 ALERT). 지정된 출근시간 이후일 시 지각한 시간 계산.	
5							자리비움	버튼	클릭 시 > '휴식', '식사', '기타' 사유 선택창 팝업. 사유 선택 시 자리비움 가능.	
6							복귀	버튼	클릭 시 > GPS로 현재 위치의 경도와 위도 정보를 로드. Haversine Formula로 회사 위치 기준 50m 이내면 복귀 가능. (아니면 ALERT)	
7							퇴근	버튼	클릭 시 > 지정된 퇴근시간 이전일 시 조회한 시간 계산 후 퇴근 가능.	
1	홈	내 근태 이력	일간 출결 기록				내 근태이력 - 일간 출결 기록	페이지	헤더 페이지 제목 '일간 출결 기록' 버튼 / '주간 통계' 버튼 / '월간 통계' 버튼 DatePicker 달력으로 날짜 선택 (최대 선택 가능 일자는 현재 날짜까지만) 해당 일자의 출결기록이 없을 시 > '출결 기록이 없습니다.' 해당 일자의 출결기록이 있을 시 > 출결 기록 탑입과 시간을 카드로 표시.	
2			주간 통계				내 근태 이력 - 주간 통계	페이지	헤더 페이지 제목 '일간 출결 기록' 버튼 / '주간 통계' 버튼 / '월간 통계' 버튼 DatePicker 달력으로 날짜 선택하면 해당 주의 데이터를 반환. (최대 선택 가능 일자는 현재 날짜까지만) 출근한 일 수 / 해당 주의 주말을 제외한 평일을 표시하는 '출근율 배너', 출근한 일 수를 표시하는 '출근 배너', 미출근한 일 수를 표시하는 '미출근 배너', 지각한 총 분 수를 표시하는 '지각 배너', 조퇴한 총 분 수를 표시하는 '조퇴 배너'.	
3			월간 통계				내 근태 이력 - 월간 통계	페이지	헤더 페이지 제목 '일간 출결 기록' 버튼 / '주간 통계' 버튼 / '월간 통계' 버튼 DatePicker 달력으로 월을 선택하면 해당 월의 데이터를 반환. (최대 선택 가능 월은 현재 월까지만) 출근한 일 수 / 해당 월의 주말을 제외한 평일을 표시하는 '출근율 배너', 출근한 일 수를 표시하는 '출근 배너', 미출근한 일 수를 표시하는 '미출근 배너', 지각한 총 분 수를 표시하는 '지각 배너', 조퇴한 총 분 수를 표시하는 '조퇴 배너'.	

출퇴근 (관리자)

리더									
번호	페이지	제작일	제작인	제작내용	제작내용	제작내용	제작내용	제작내용	
1	홈	출퇴근 현황	전체 보기	출퇴근 현황 - 전체 보기	페이지	(token 검사하여 ADMIN 아닌 경우 Alert > 홈으로 이동) 헤더 페이지 제목 '전체 보기' 버튼 / '팀별 보기' 버튼 출근 한 직원 수 / 전체 직원 수를 표시하는 '출근을 배너'. 출근한 직원 수를 표시하는 '출근 배너', 미출근(휴가자+결근자)한 직원 수를 표시하는 '미출근 배너', 출근 전인 직원 수를 표시하는 '출근 전 배너', 지각한 직원 수를 표시하는 '지각 배너', 조퇴한 직원 수를 표시하는 '조퇴 배너', 출근한 직원의 상태를 '근무 중' / '자리비움 중' / '퇴근' 으로 표시하는 '출근인원 근무상태 배너'.			
2			팀별 보기	출퇴근 현황 - 팀별 보기	페이지	(token 검사하여 ADMIN 아닌 경우 Alert > 홈으로 이동) 헤더 페이지 제목 '전체 보기' 버튼 / '팀별 보기' 버튼 '경영팀' / '인사팀' / '업무팀' 버튼 출근 한 해당 팀 직원 수 / 해당 팀 전체 직원 수를 표시하는 '출근을 배너'. 출근한 해당 팀 직원 수를 표시하는 '출근 배너', 미출근(휴가자+결근자)한 해당 팀 직원 수를 표시하는 '미출근 배너', 출근 전인 해당 팀 직원 수를 표시하는 '출근 전 배너', 지각한 해당 팀 직원 수를 표시하는 '지각 배너', 조퇴한 해당 팀 직원 수를 표시하는 '조퇴 배너', 출근한 해당 팀 직원의 상태를 '근무 중' / '자리비움 중' / '퇴근' 으로 표시하는 '출근인원 근무상태 배너'.			
3				출퇴근 현황 페이지 (오늘 일자 조회)	페이지	'출근 전 배너' / '출근인원 근무상태 배너' 표시. 입력된 출근시간(09시) 이전에 조회 시 모든 직원이 출근 전으로 카운팅. 09시 이전 출근하면 '출근' 으로 카운팅, 09시 이후 출근하지 않았을 시 '미출근'으로 카운팅.			
4				출퇴근 현황 페이지 (이전 일자 조회)	페이지	'출근 전 배너' / '출근인원 근무상태 배너' 표시 안됨. 자정까지 출근하지 않아서 날짜가 바뀌면 해당 직원은 출근 전 > 미출근 상태로 변경.			

경비관리

No	Location						상태	Page Name	Detail	
	1Depth	2Depth	3Depth	4Depth	5Depth	6Depth			Content	Description
경비관리										
1	경비관리 목록						경비관리 목록페이지	경비관리 목록	경비관리 게시물을 데이터 조회 (리더는 전체 조회, 직원은 작성한 게시물만 조회) 작성 페이지 이동, 상세 페이지 이동, 검색창	
2		경비관리 작성					경비관리 작성페이지	경비관리 작성	경비관리 게시물을 작성 DTO를 통해 게시물 저장	
3		경비관리 상세					경비관리 상세페이지	경비관리 상세 조회	경비관리 id로 상세 내용 불러오기, 경비관리 수정 페이지 이동, 경비관리 삭제(관리자만), 경비관리 목록 페이지 이동	
5		경비관리 수정					경비관리 수정페이지	경비관리 수정	id로 상세내용 불러오기, 경비관리 내용 수정	
6		경비관리 삭제					경비관리 삭제		일치하는 Id 데이터 삭제, 관리자만 삭제 가능	
		검색조회					경비관리 목록페이지	경비관리 검색리스트	검색 키워드를 가지고 있는 정보 조회	
7		승인 경비내역					승인 경비내역 페이지	승인경비 내역	승인된 경비 내역 조회	
메인										
1	메인 (경비관리)						메인 페이지	경비관리 메인버튼	비수락, 거절 개수 표시 클릭시 경비관리 목록 페이지로 이동 메인페이지에서 리더만 버튼이 보임	
메뉴										
1	메뉴 (경비관리)						메뉴 페이지	경비관리 메뉴 버튼	클릭시 경비관리 목록페이지 이동	

보고서 및 템플릿

No	Location						상태	Page Name	Detail	
	1Depth	2Depth	3Depth	4Depth	5Depth	6Depth			Content	Description
유저										
1	보고서	보고서 리스트	보고서 상세조회				보고서 리스트 페이지		각 보고서를 순차적으로 조회함.	
2							보고서 상세조회 페이지		템플릿 이름 / 작성자 이름 및 프로필 사진 / 작성 시간 / 댓글 버튼 보고서 상세 내용	
3				자세히	내용 수정		내용 수정 기능		수정 가능하게 입력창이 있는 수정페이지로 이동 저장 버튼 하단에 생성	
4					삭제		삭제 기능		버튼 누를 시 '보고서를 삭제하시겠습니까?' 나타나며 보고서 삭제 가능함	
5		보고서 작성					보고서 작성 페이지		템플릿 선택 후 선택한 템플릿 이름 및 각 템플릿에 맞는 입력칸 나타남 하단 제출 버튼	
6	댓글	댓글 리스트 조회					댓글 리스트 조회 기능		프로필 사진 / 이름 / 댓글 내용 / 작성일자 자신이 작성한 댓글의 경우 수정 가능	
7		댓글 작성					댓글 작성 기능		입력창에 댓글 내용을 작성할 수 있으며 체크표시 아이콘을 누르면 작성 완료된다.	
8		댓글 수정					댓글 수정 기능		댓글 내용을 수정 가능하다. 수정 버튼을 누르면 입력칸이 나타나고 저장하면 수정 이 된다.	
관리자										
1	템플릿 설정						템플릿 설정 페이지		각 탭에서 템플릿 리스트를 조회할 수 있다. 각 템플릿 목록을 클릭하면 템플릿 상세페이지로 이동된다. 템플릿 목록에서 템플릿 삭제가 가능하다.	
2		템플릿 살세 설정					템플릿 상세설정 페이지		템플릿 설정을 변경 할 수 있으며 항목은 변경 불가하다. 설정 칸을 이용하여 테마 색상, 아이콘을 지정 가능 고급 설정을 이용하여 누가 작성하는지, 작성해야 하는 날이 있는지 설정	
3	새 템플릿 만들기						새 템플릿 만들기 페이지		템플릿 이름을 작성할 수 있다. 설정 칸을 이용하여 테마 색상, 아이콘을 지정 가능	
4		보고서 작성 항목 추가					보고서 작성 항목 추가 기능		보고서 작성 항목 문항을 만들 수 있다. 문항 이름을 공통적으로 적을 수 있다. 어떠한 항목을 만들지 목록에서 고를 수 있으며 고른 항목에 따라 작성 내용이 바뀐다.	

휴가

No	Location							상태	Page Name	Detail	
		1Depth	2Depth	3Depth	4Depth	5Depth	6Depth			Content	Description
1	홈	휴가 페이지						휴가	휴가조회	본인의 남은 연차 개수와 최근 휴가 신청 내역 5개 확인 및 전체 내역 조회와 신청 페이지로 이동 가능	
2			휴가 신청					휴가 신청	휴가신청	시작 날짜와 종료 날짜를 선택할 수 있고 선택한 기간에 맞춰 총 일수가 계산되어 표시된다. 휴가 타입(연차, 반차, 반반차)을 선택할 수 있고 휴가 사유를 작성 후 신청	
3			전체 내역					전체 조회	리스트	본인의 휴가 신청 전체 내역을 조회 할 수 있는 페이지	
4			상세조회					상세 조회	상세	본인이 신청한 휴가에 대한 상세 내역을 나타낸 날짜, 총 일수, 신청날짜, 사유 및 관리자의 승인 상태를 표시, 직원은 취소 버튼이 활성화 되고 관리자는 해당 페이지에 승인 및 반려 버튼 활성화	

No	Location							상태	Page Name	Detail	
		1Depth	2Depth	3Depth	4Depth	5Depth	6Depth			Content	Description
1	홈	알림 메시지						알림 메시지	메시지	직원이 신청한 내역에 해당하는 메시지가 화면 상단 중앙에 toast message로 구현하여 표시, 컴포넌트 위치를 최상위로 옮겨 전체 라우터 내에 표시 됨, 메시지 클릭 시 알림 메시지 리스트로 이동	
2			리스트					알림함	리스트	전체 알림 메시지 리스트를 표시하고 읽은 메시지 내역은 0.5의 opacity 적용 메시지 클릭 시 해당 메시지와 관련된 내역의 상세 페이지로 이동.	

로그인 및 회원가입

No	Location				Page ID	Page Name	Detail	
	1Depth	2Depth	3Depth	4Depth		Content	Description	
1	로그인				MEM_01_01	로그인 페이지	이메일, 비밀번호 입력	이메일로 로그인
2		회원가입			MEM_01_02	회원가입 페이지	이메일, 이름, 휴대폰번호, 비밀번호 입력	직원은 리더의 가입 승인을 받아야 가입이 가능.
3	메뉴	가입승인			MEM_01_03	가입승인 페이지	가입을 신청한 직원의 이름, 이메일 표시	직원이 가입 신청하면 가입 승인 페이지에 노출
4	마이페이지	사진 편집			MP_01_01	프로필 사진 변경	프로필 사진 변경	
5		재직 정보			MP_01_02	재직 정보	직급(리더, 직원), 사번 표시	직급은 리더, 직원으로 나뉘며 리더가 지정하는 항목
6					MP_01_03	부서	각 직원 부서 표시 (경영, 인사, 업무)	리더가 지정하는 항목

스케줄

스케줄 기능 정의						
No	1 Depth	2 Depth	3 Depth	Page Name	Detail	
					Content	Description
1	메인페이지	스케줄		스케줄	스케줄 캘린더	스케줄 조회, 편집용 달력을 보여준다. 기본값은 현재 월로 보여준다. 달력은 현재 월 기준 앞, 뒤로 최대 1년까지 조회 가능하다.
2	메인페이지	스케줄	스케줄 수정	스케줄 수정	스케줄 수정 페이지	스케줄 수정 버튼 클릭/터치 시 수정 화면으로 전환 된다. 스케줄 수정은 근무, 휴가 탭으로 나뉜다. 근무 탭에는 날짜, 근무시간, 휴게시간, 메모를 입력할 수 있다. 휴가탭을 클릭/터치 하면 휴가를 신청할 수 있다. 수정완료 버튼 클릭 시 해당 스케줄을 관리자에게 발송하고 승인 대기 상태가 된다.

02. 프로젝트 상세

2-1. 기능 정의서

2-2. 요구사항 정의서

2-3. UML

2-4. ERD

Office + Time = Offime

본 시스템은 직원들의 근무 출퇴근 및 스케줄 관리, 휴가 신청, 경비 처리 및 비용, 보고서 관리, 로그인 및 멤버 관리 기능을 제공하여 운영 효율성을 극대화하는 것을 목표로 한다.

Offime 요구사항 정의서

1. 개요

Office + Time = Offime

본 시스템은 직원들의 근무 출퇴근 및 스케줄 관리, 휴가 신청, 경비 처리 및 비용, 보고서 관리, 로그인 및 멤버 관리 기능을 제공하여 운영 효율성을 극대화하는 것을 목표로 한다.

2. 주요 기능

2.1. 로그인 및 멤버 관리

로그인 기능

- 사용자는 이메일과 비밀번호를 입력하여 로그인 가능
- 가입을 신청한 직원의 정보를 리더가 확인 후 승인하여 앱 이용을 허가
- 비밀번호 재설정 기능 제공 및 프로필 사진 추가, 변경 가능
- 로그아웃 여부 탑재, 확인 시 로그아웃, 로그인 페이지로 이동

회원 관리

- 회원가입 기능 제공 (관리자 승인 필요)
- 직급 및 부서, 사번이 표기
- 직급, 부서는 리더가 직접 지정관리자 권한 부여 및 철회 가능
- 해당 직원과 리더만이 탈퇴, 혹은 삭제가 가능하고 탈퇴 시 비활성화 처리

2.2. 스케줄 관리

- 캘린더에서 해당 날짜를 클릭하여 스케줄 추가 가능
- 스케줄 추가 시 근무시간, 휴게시간, 메모 입력 가능
- 스케줄 추가 후 승인 대기 상태로 표시됨
- 관리자가 승인 또는 반려 가능
- 승인 여부에 따라 캘린더에 반영됨
- 실시간 변경 사항 알림 제공
- 과거 캘린더 조회 가능
- 휴가 신청 시 자동으로 캘린더에 반영됨

2.3. 경비 및 비용 처리

비용 목록 관리 (관리자 / 직원)

- 관리자는 모든 비용 계사를 조회 가능
- 직원은 자신이 작성한 비용 계사를만 조회 가능
- 계사를 목록에 번호, 제목, 작성자, 작성 날짜, 카테고리(식비, 교통, 숙박, 경조사, 기타), 금액 표시
- 계사를 상세 조회 기능 제공
- 날짜, 카테고리, 작성자 등으로 검색 가능

비용 작성 기능

- 날짜, 카테고리, 제목, 사진(최대 10 개), 금액, 총금액, 내용 입력 가능
- 작성 완료 시 목록 페이지로 이동
- 작성 시 알림 발송
- 사진 삭제 기능 제공

비용 상세 조회 기능

- 작성자, 날짜, 카테고리, 제목, 사진(다운로드 가능), 금액, 내용 표시
- 수정 및 삭제 기능 제공 (관리자만 삭제 가능)
- 목록으로 이동 버튼 제공

비용 수정 기능

- 제목, 사진, 금액, 내용 수정 가능
- 수정 완료 시 상세 페이지 이동
- 수정 알림(관리자에게 발송)
- 작성자 및 관리자만 수정 가능

비용 승인 (관리자 전용)

- 비용 요청 발생 시 관리자 알림 제공
- 작성자, 제목, 내용, 금액, 간략 정보 제공
- 승인/반려 버튼 제공

비용 처리 메인 화면

- 비용 전용 칸 (관리자 전용)
- 상세 페이지 이동 기능 제공

2.3. 보고서 및 템플릿

보고서 목록 (직원)

- 템플릿에 따른 보고서 아이콘, 템플릿 이름 표시
- 작성자 이름, 프로필 사진, 작성 날짜 및 시간 표시
- 댓글 아이콘 클릭 시 댓글 페이지 이동 기능 제공
- 작성 버튼 클릭 시 작성페이지 이동 기능 제공

보고서 작성 기능 (직원)

- 설정된 템플릿 작성 가능
- 템플릿 선택 후 작성 기능 제공

보고서 상세 조회 기능 (직원)

- 템플릿 이름, 작성자 이름, 프로필 사진, 작성 시간 표시
- 댓글 페이지 이동 기능 제공
- 보고서 삭제 가능

보고서 댓글 기능 (직원)

- 댓글 작성 및 수정 기능 제공
- 댓글 리스트 조회 기능 제공

템플릿 설정 기능 (관리자 전용)

- 템플릿 리스트 조회 기능 제공
- 템플릿 클릭 시 템플릿 상세페이지 이동 기능 제공
- 템플릿 삭제 기능 제공
- 템플릿 설정 변경 기능 제공 – 테마색상, 아이콘 지정

새템플릿 작성 기능 (관리자 전용)

- 템플릿 이름, 테마 색상, 아이콘 등을 작성 가능
- 보고서 작성 항목 문항 제공
- 문항 이름 공통적으로 작성 가능
- 항목 선택에 따른 작성 내용 변경 기능 제공

2.5. 휴가 신청

- 직원은 휴가 신청 페이지에서 휴가 신청 가능
- 연차 및 사용한 연차 정보 제공
- 휴가 유형(종일/반차) 선택 가능
- 휴가 시작일 및 종료일 입력 가능
- 결재 상태(대기, 승인, 반려) 확인 가능
- 휴가 사유 작성 기능 제공
- 결재자(리더) 지정 가능
- 신청 일자 포함
- 신청 후 승인 대기 상태로 표시됨
- 관리자는 휴가 신청 내역을 확인 후 승인/반려 가능
- 휴가 신청 내역은 데이터베이스의 휴가 테이블에 저장됨

2.6. 출퇴근 관리

출퇴근 배너

- 현재시간 및 출근 가능 제공
- 출근 성공 시 근무중으로 변경 및 자리비움, 퇴근 기능 제공
- 자리비움 시 식사, 휴식, 기타 선택 후 복귀 및 퇴근 기능 제공
- GPS 기능 제공
- 퇴근 시 조퇴한 시간 계산 후 퇴근 기능 제공

본인 근태 이력 목록

- 근태 이력 일간, 주간, 월간 통계 조회 가능
- 달력에 나와있는 날짜를 선택하여 기록정보 조회 가능

출퇴근 현황 (관리자)

- 출퇴근 현황 전체 조회 가능
- 출퇴근 현황 템플릿 조회 가능
- 출퇴근 오늘 일자 및 이전 일자 조회 가능

3. 결론

본 시스템을 통해 직원들의 출퇴근 및 근무 스케줄, 휴가 신청 및 비용 처리 과정을 보다 체계적으로 관리할 수 있으며, 관리자와 직원 간의 원활한 소통 및 효율적인 업무 처리가 가능하도록 설계되었다.

02. 프로젝트 상세

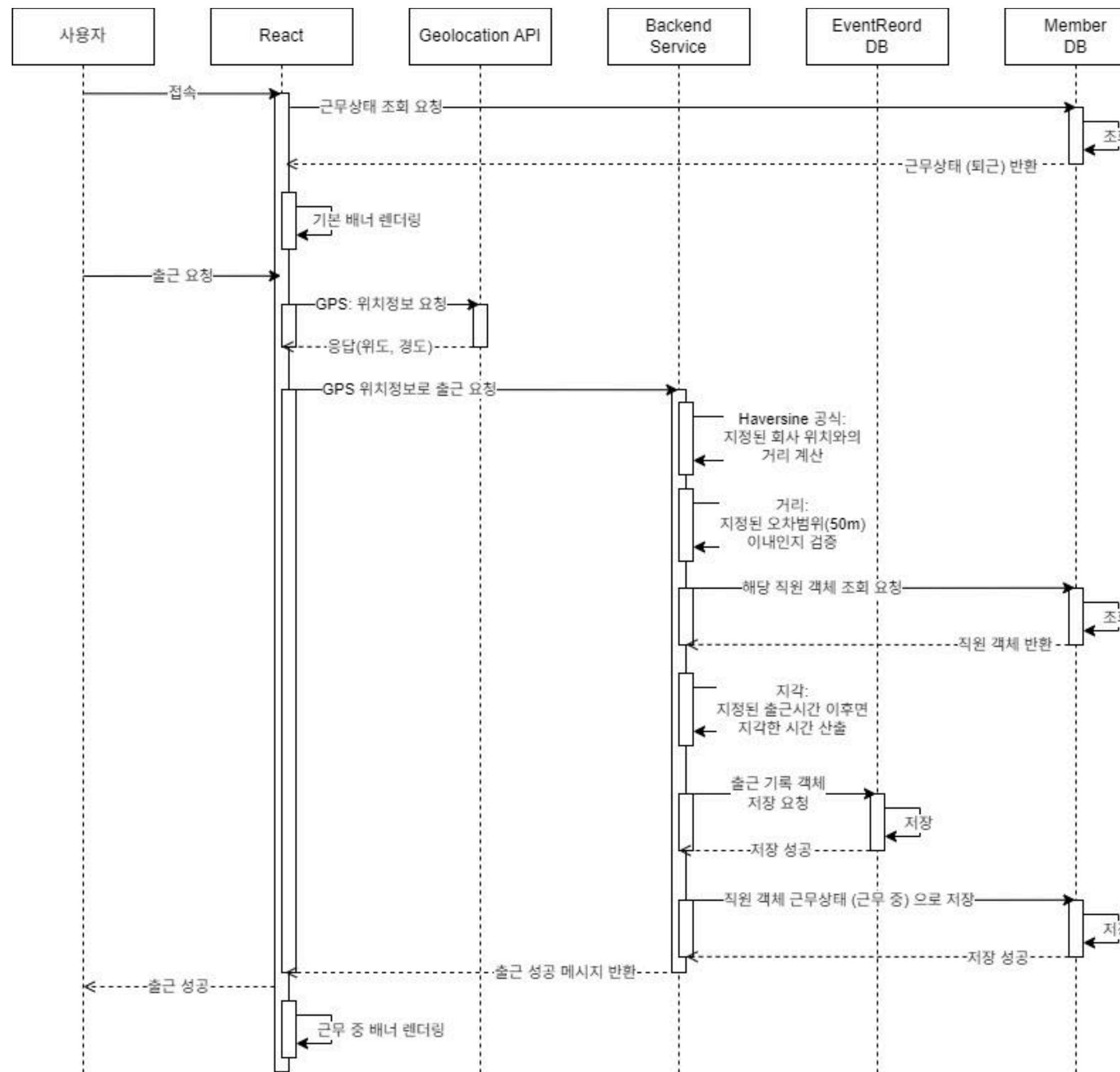
2-1. 기능 정의서

2-2. 요구사항 정의서

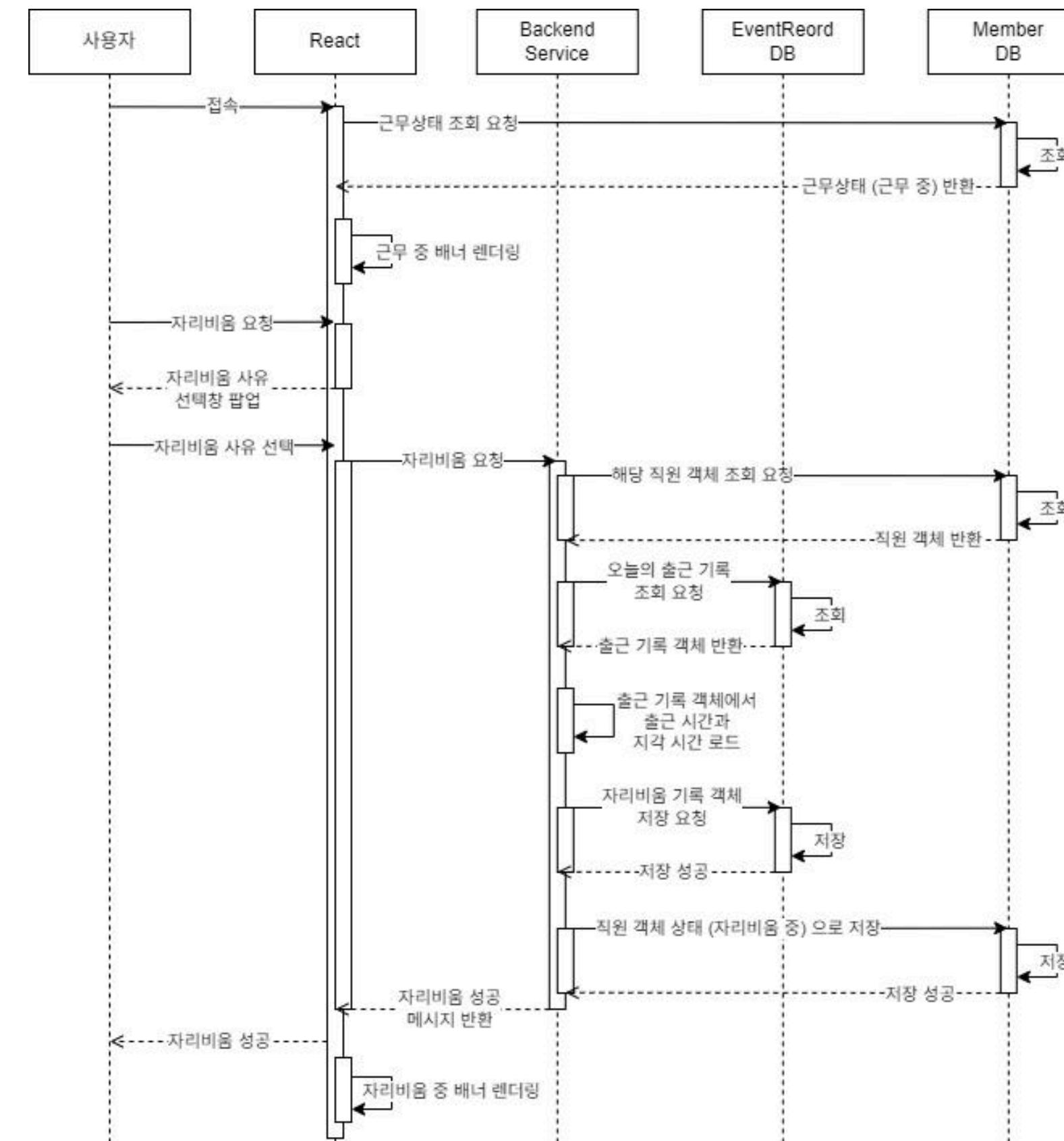
2-3. UML

2-4. ERD

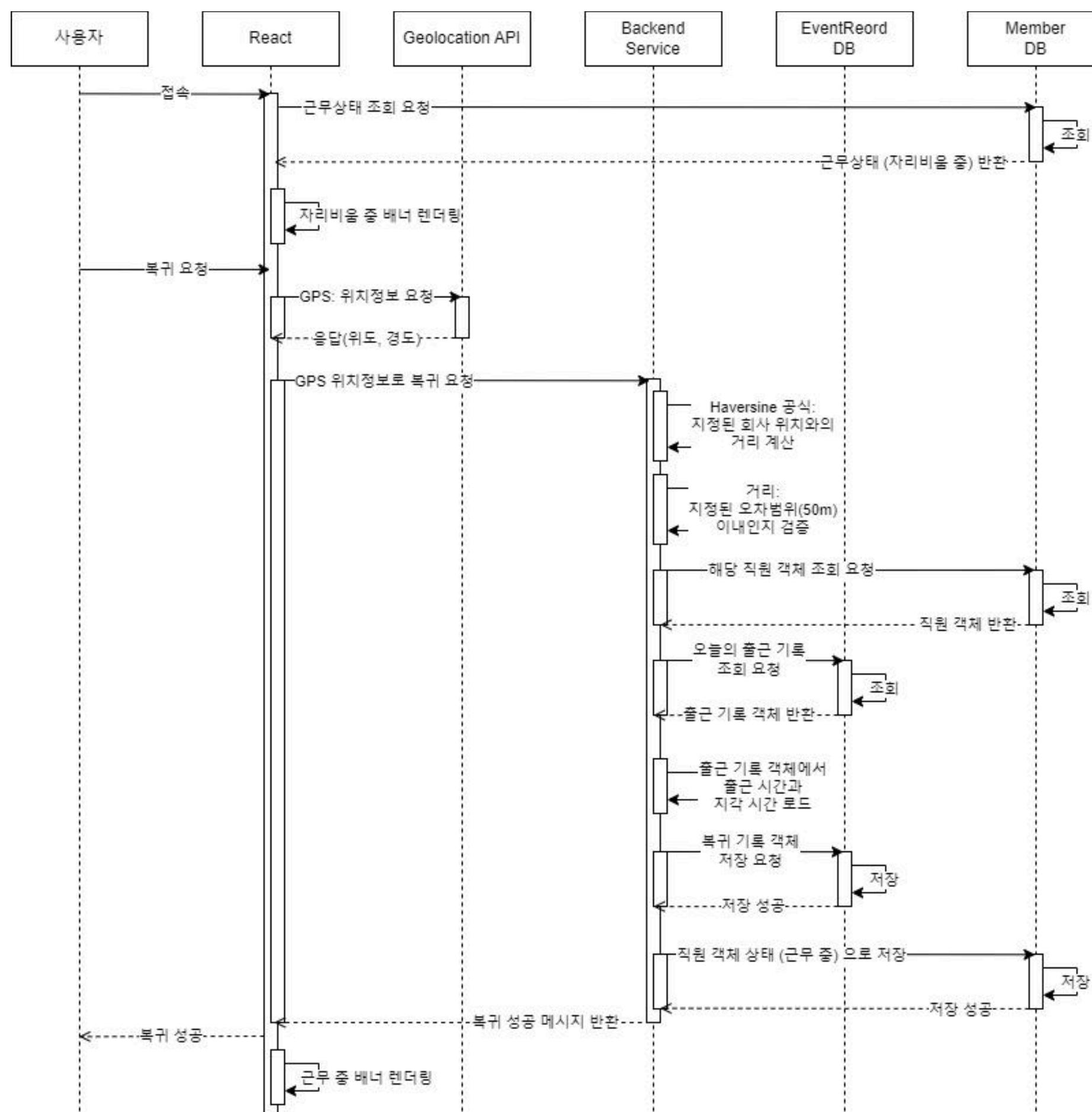
Sequence Diagram - 출근



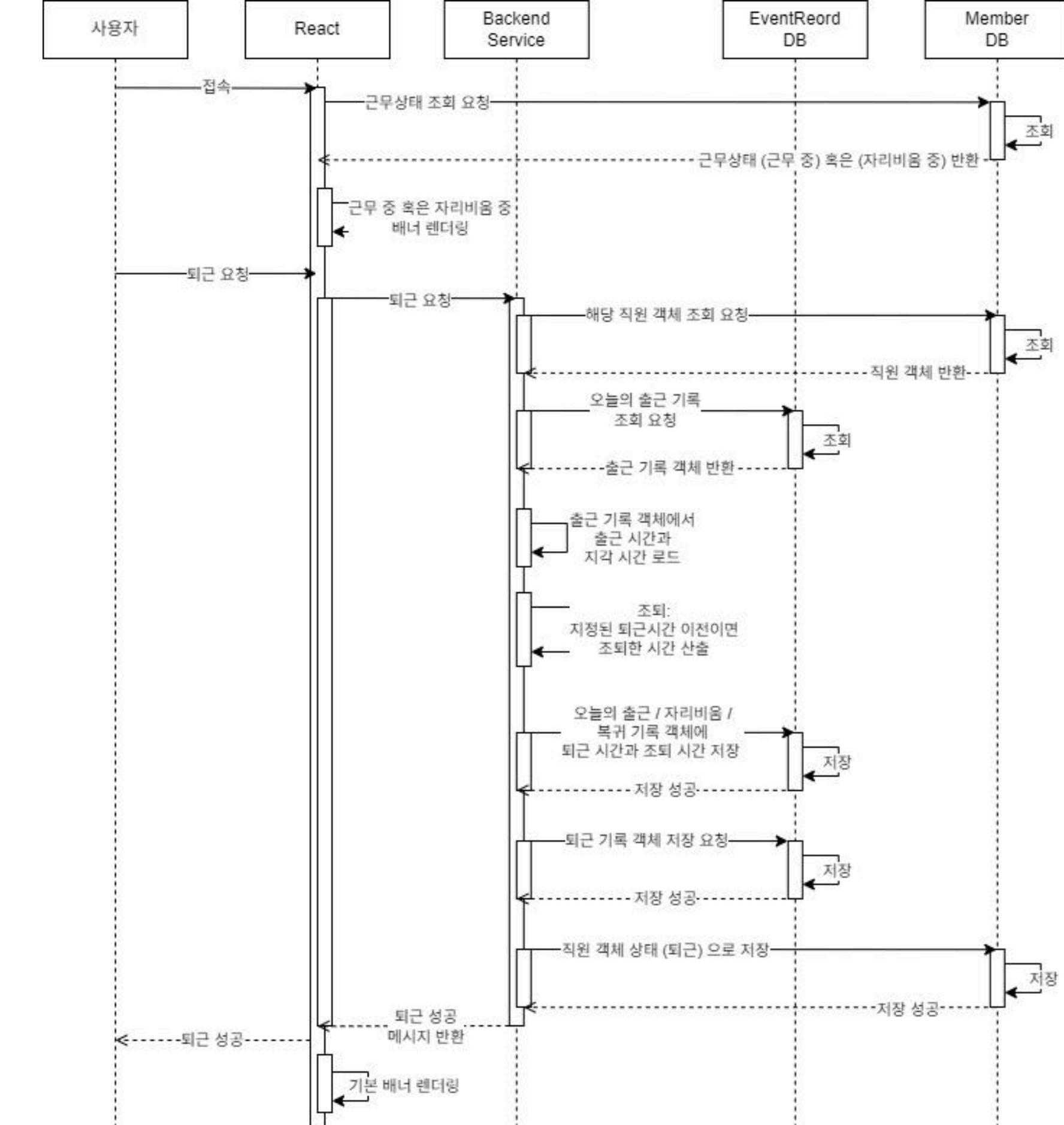
Sequence Diagram - 자리비움



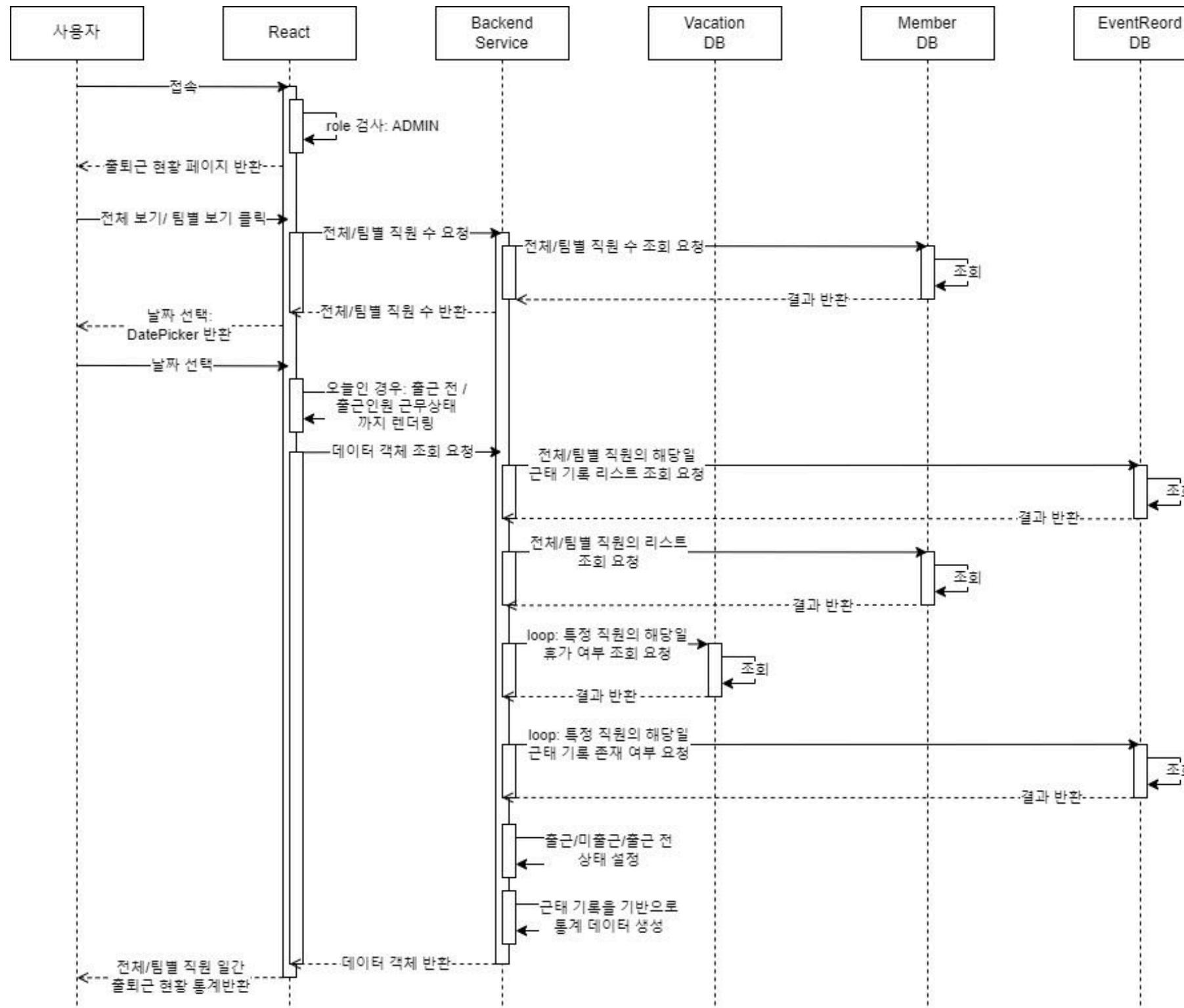
Sequence Diagram - 복귀



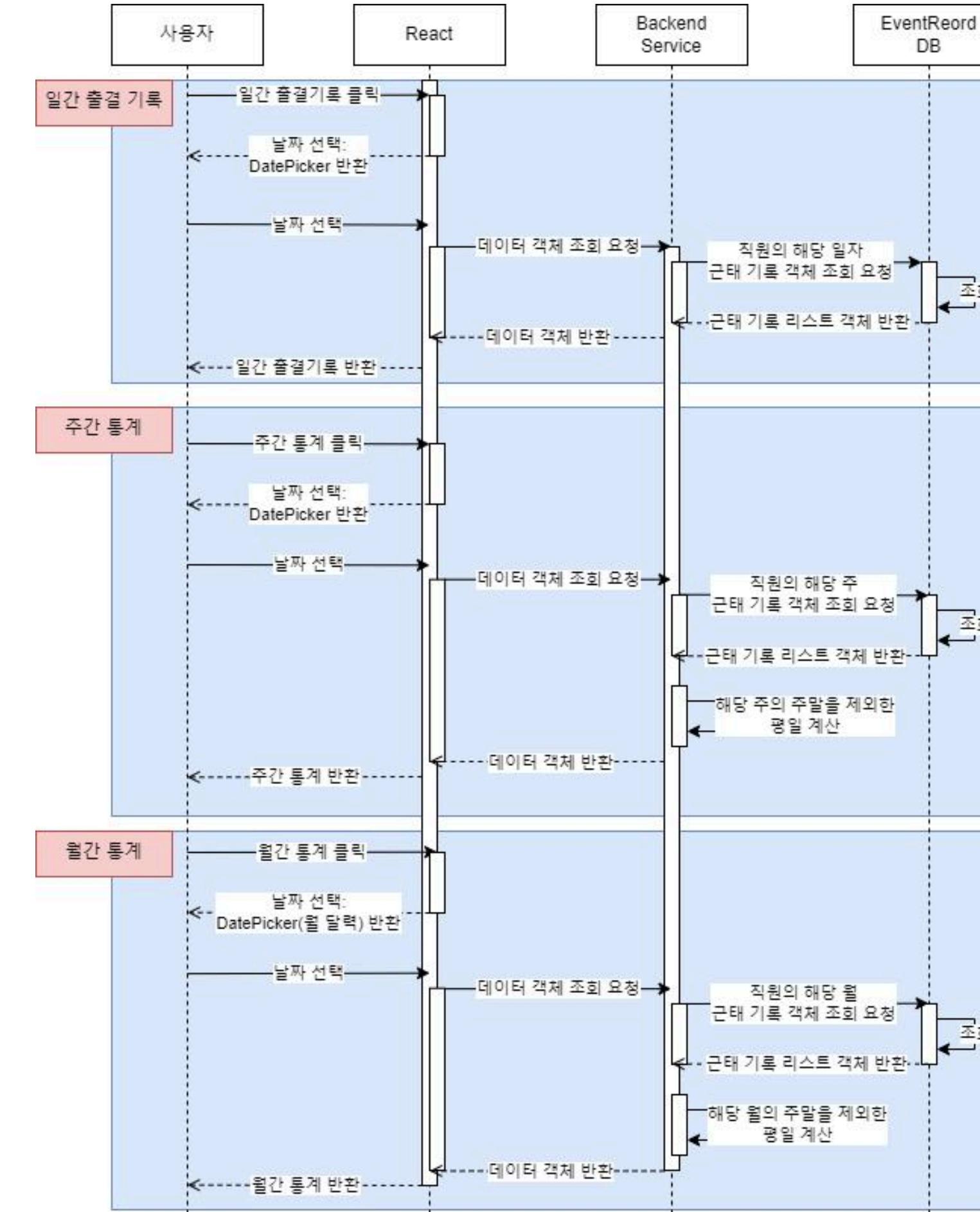
Sequence Diagram - 퇴근



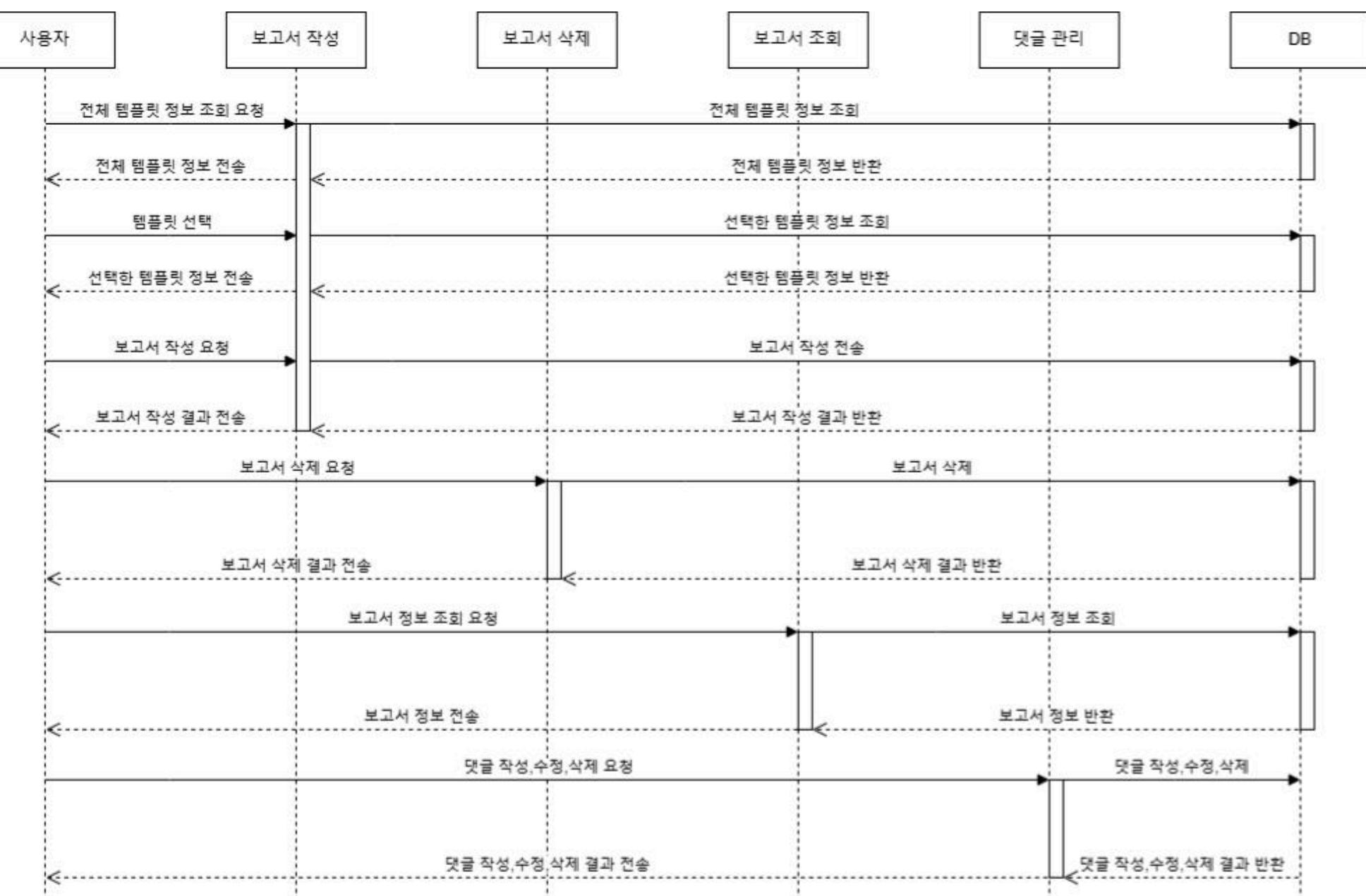
Sequence Diagram - (리더) 출퇴근 현황



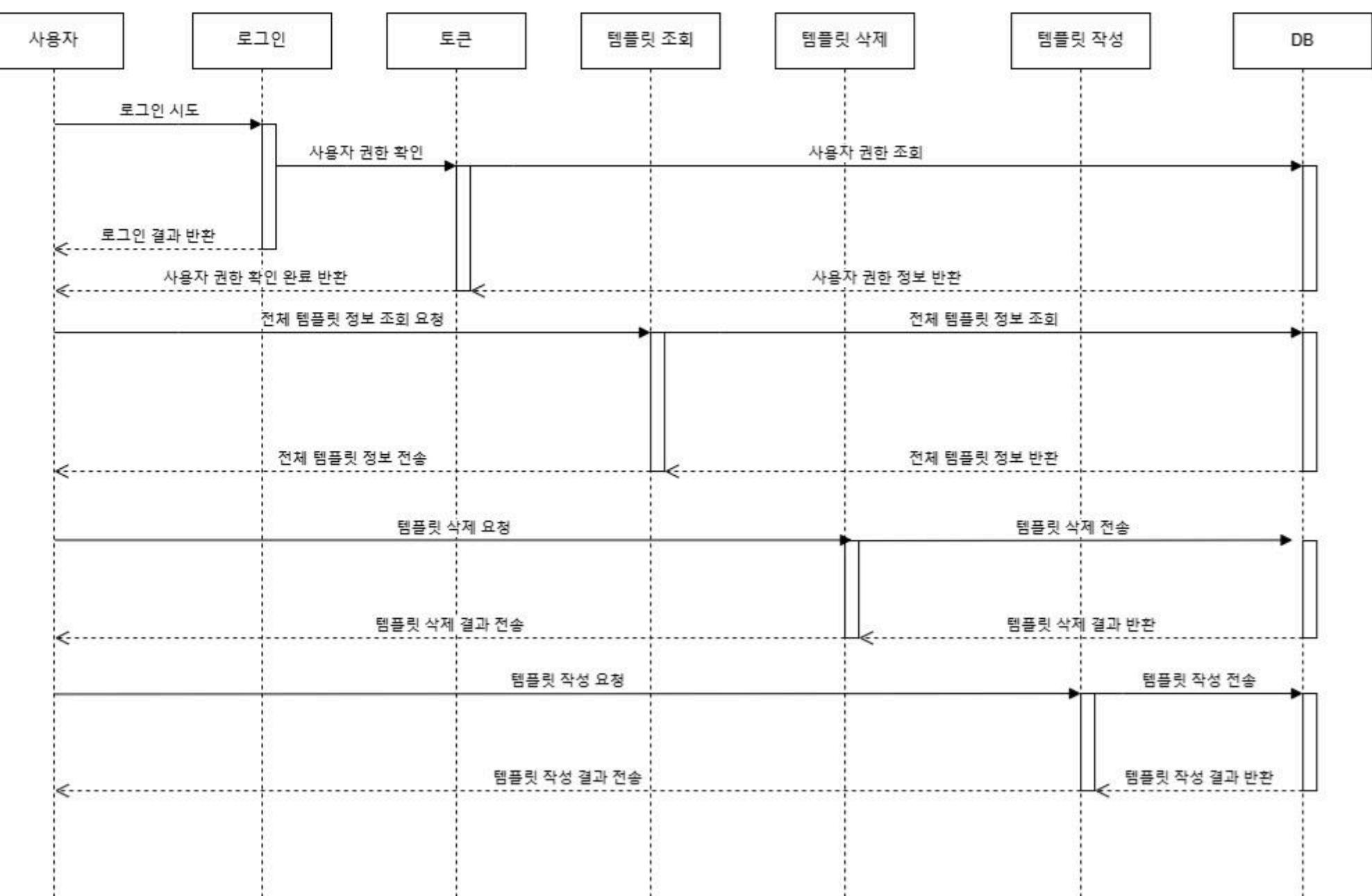
Sequence Diagram - (직원) 내 근태이력



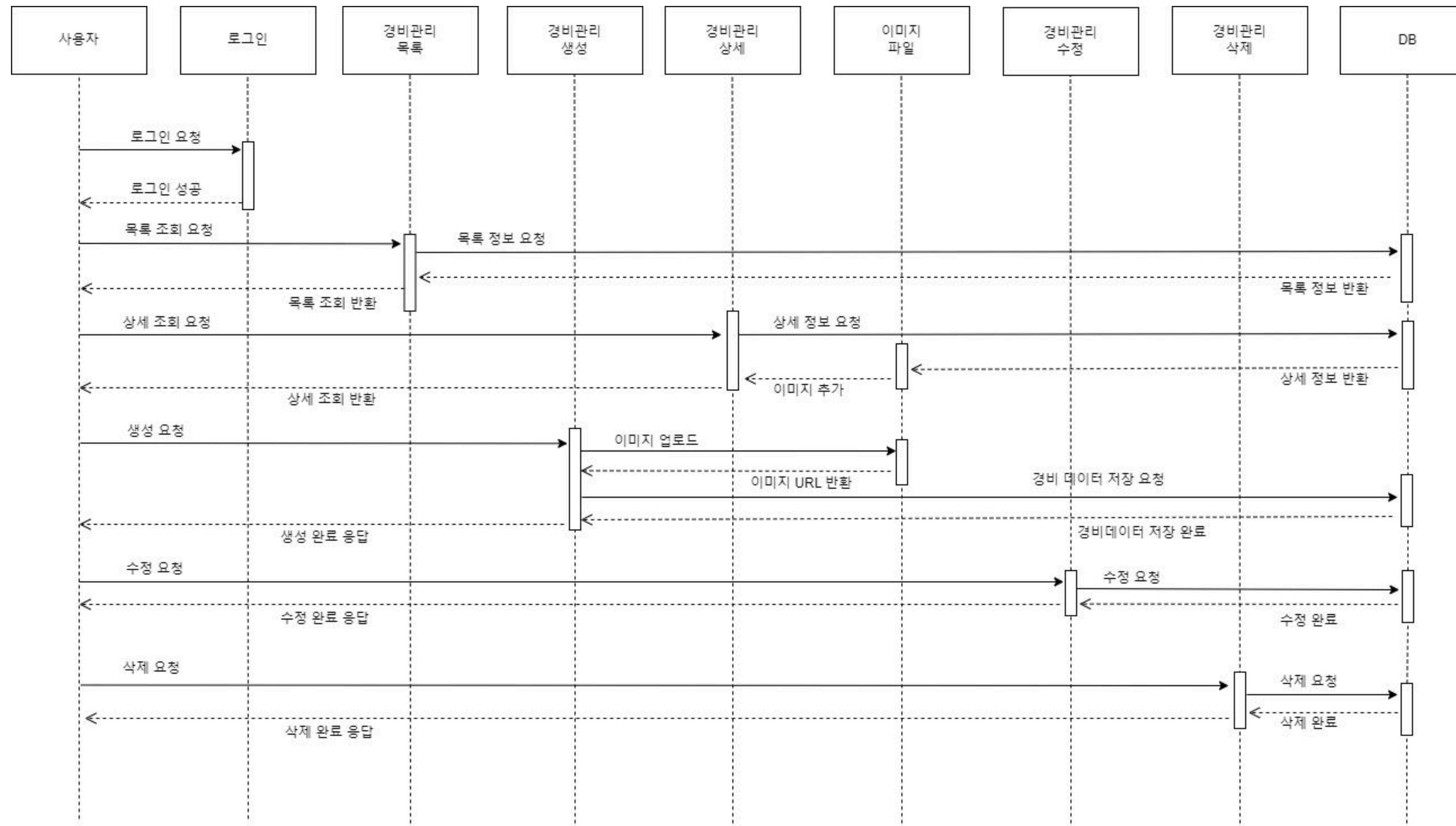
Sequence Diagram - 보고서



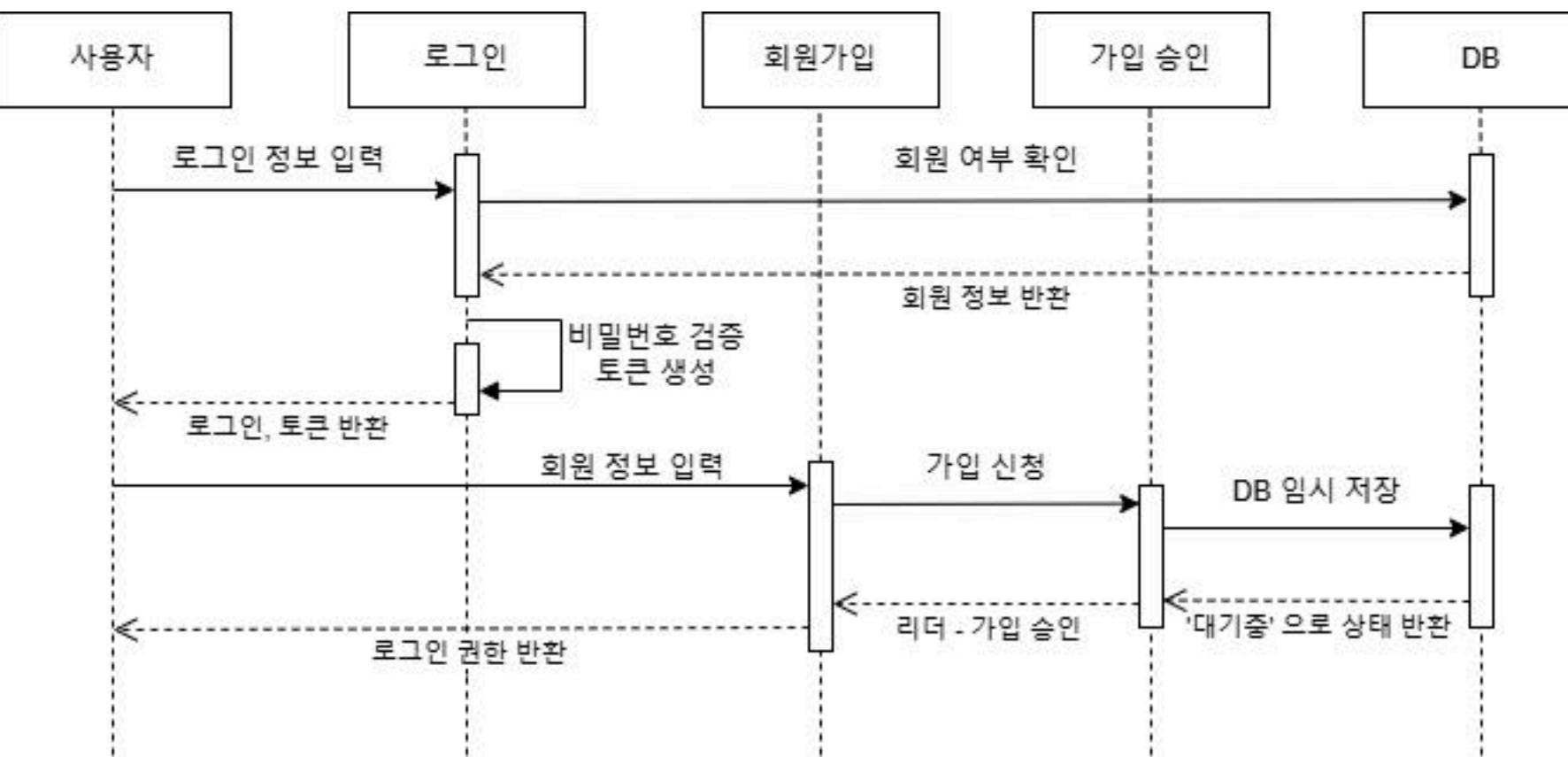
Sequence Diagram - 템플릿



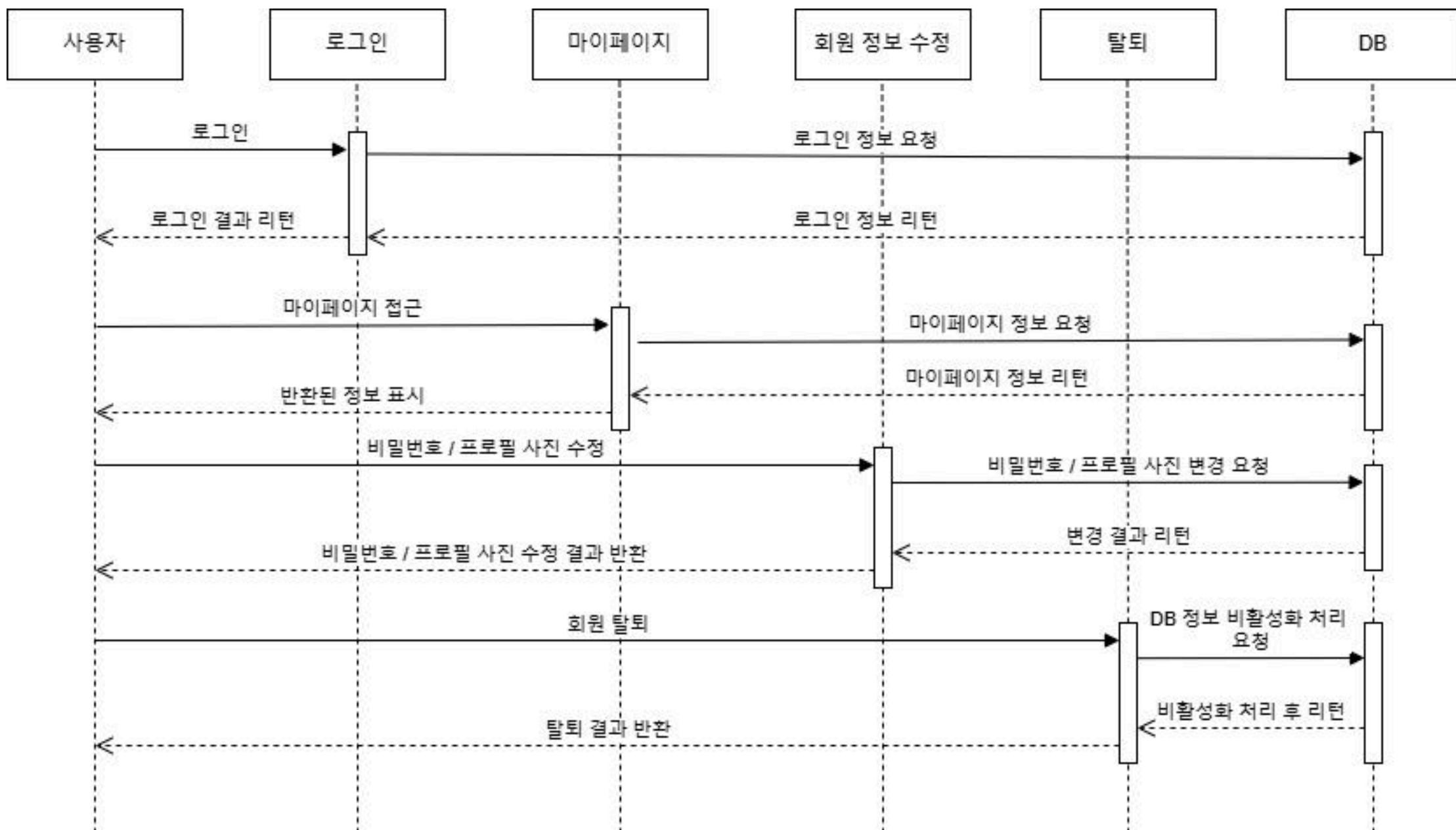
Sequence Diagram - 경비관리



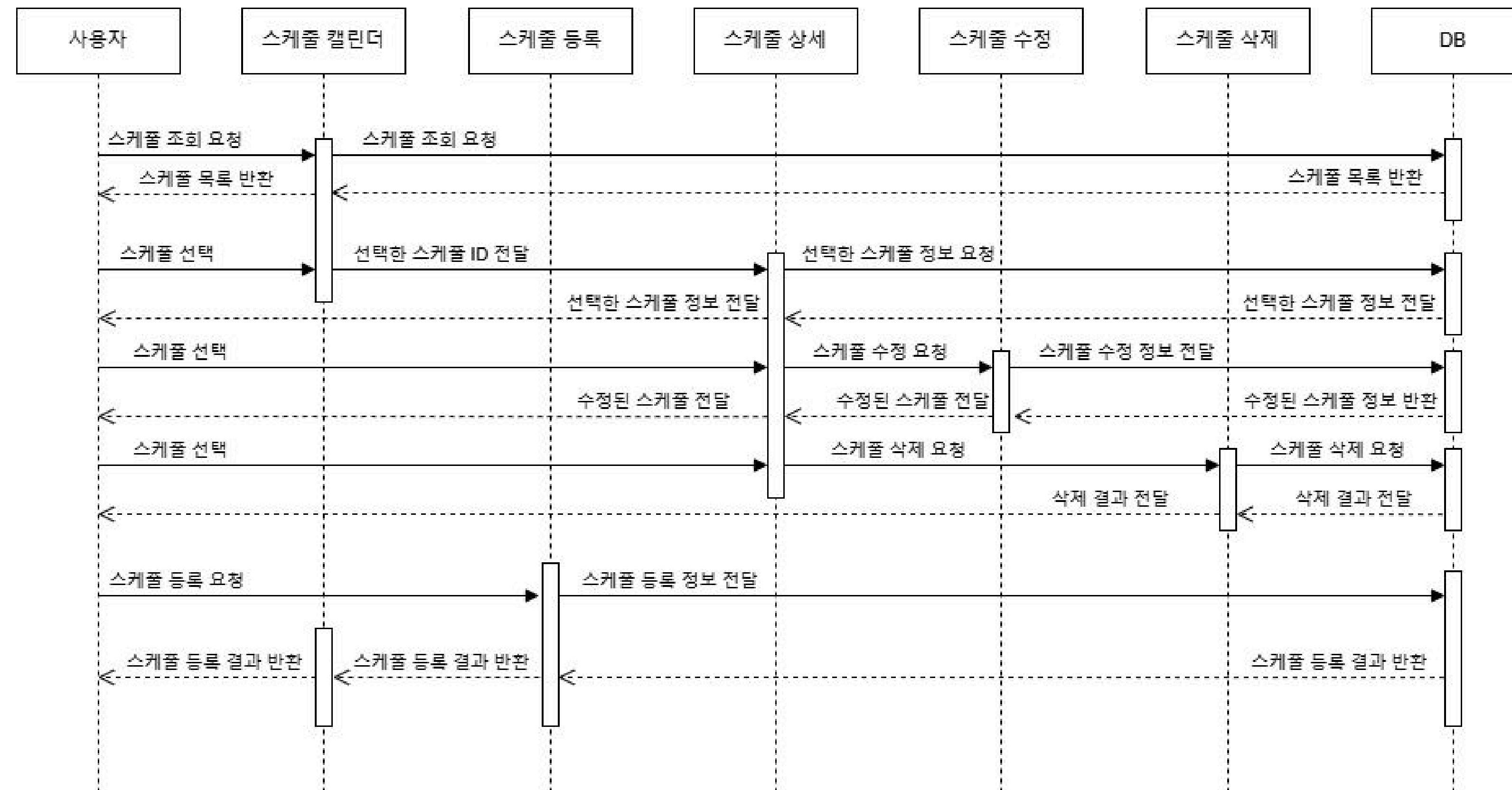
Sequence Diagram - 회원가입



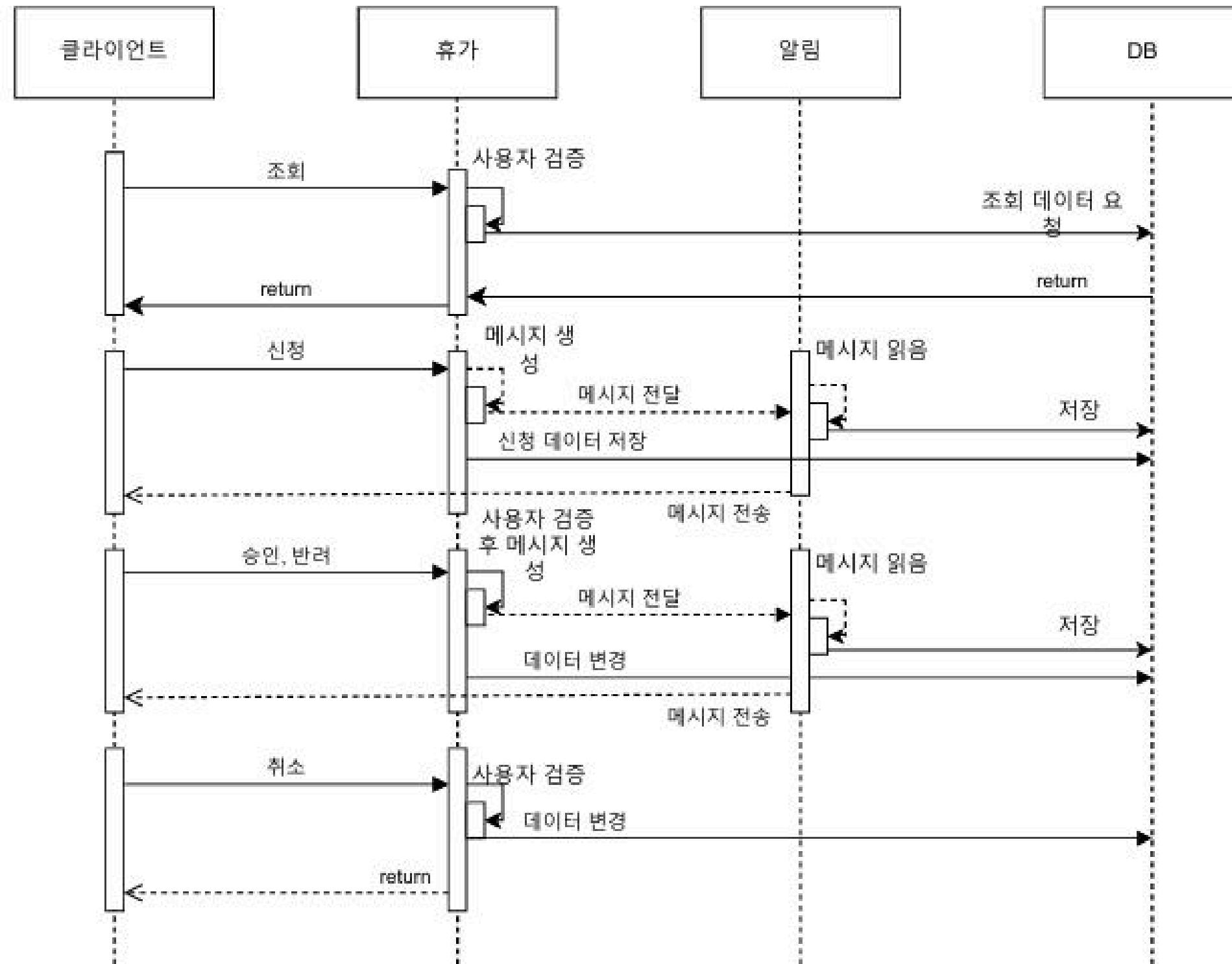
Sequence Diagram - 회원 정보



Sequence Diagram - 스케줄



Sequence Diagram - 휴가, 알림



Use Case Diagram

직원

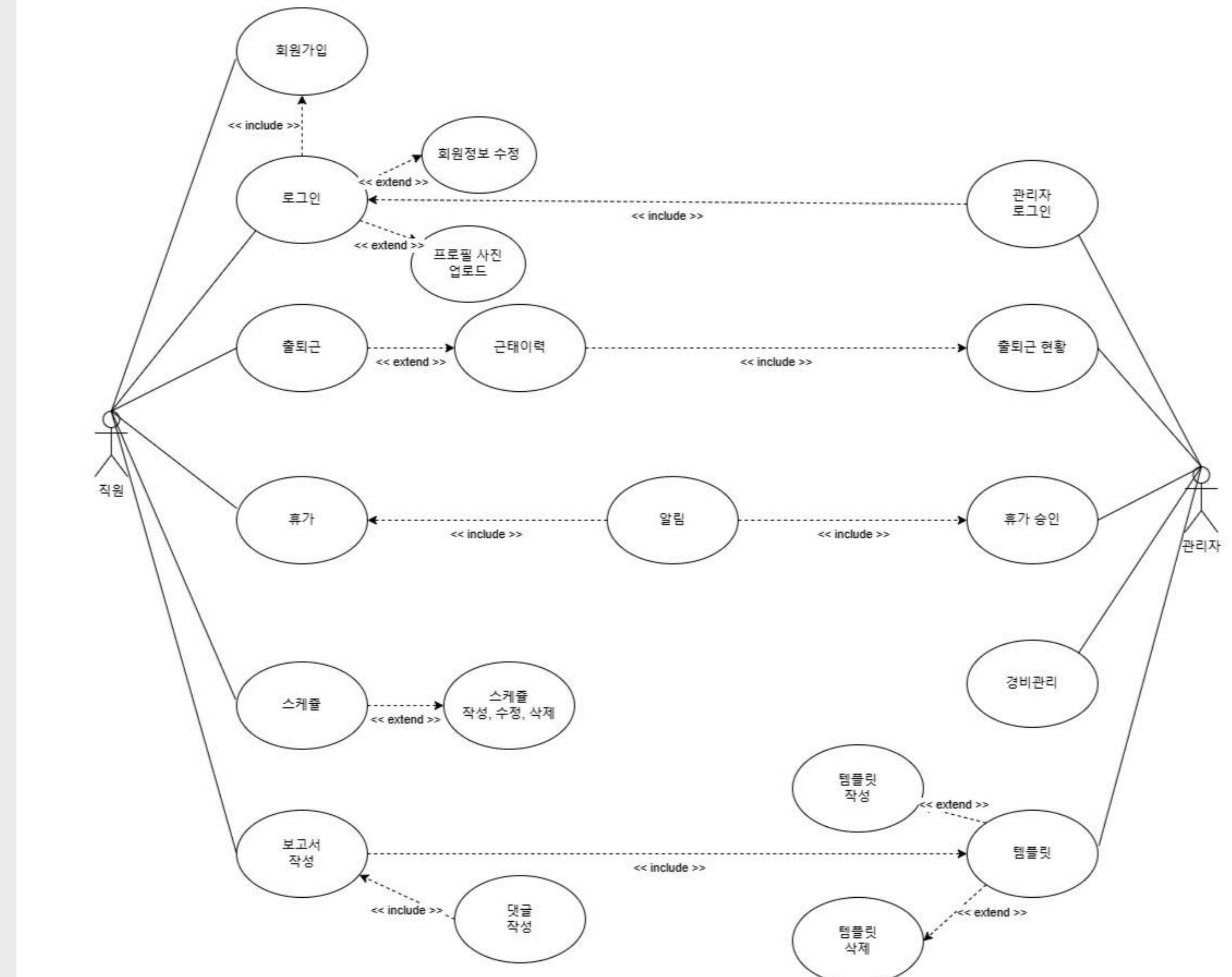
- 출근 및 퇴근 기록
- 근무시간 조회
- 휴가 신청

관리자

- 전체 직원의 근태 관리
- 휴가 승인/거부
- 월별/연별 근태 보고서 생성
- 시스템 설정 및 관리

주요 유스케이스

- 출근/퇴근 기록: 직원이 출퇴근 시간을 기록하고 관리자가 이를 확인
- 근무시간 조회: 직원이 자신의 근무 시간을 조회할 수 있음
- 휴가 신청: 직원이 휴가를 신청하고 팀장이 이를 승인/거부
- 근태 보고서: 관리자 또는 팀장이 직원들의 근태를 월별/연별로 보고서 형태로 조회



Class Diagram

클래스 다이어그램 부분 발췌

Member (회원)

- 시스템의 사용자 (이름, 이메일, 부서, 역할 등 포함)
 - **Schedule**, **Vacation**, **EventRecord** 등과 연관됨

Schedule (일정)

- 출근 및 퇴근 시간 관리
 - Member와 연관

Vacation (휴가)

- 휴가 유형, 상태, 시작/종료일 등
 - Member와 연관

EventRecord (근무 기록)

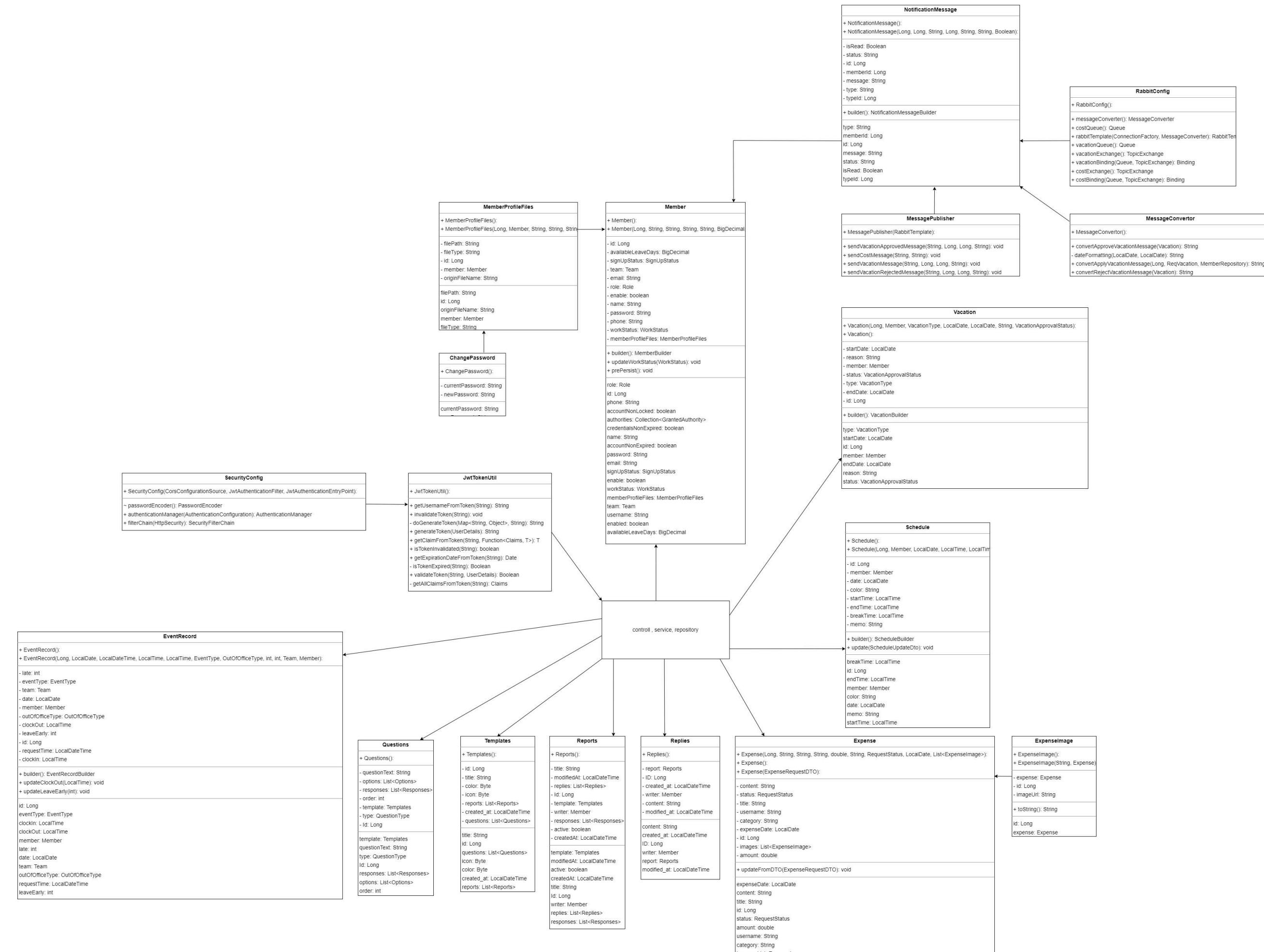
- 출근/퇴근 이벤트 저장
 - 이벤트 타입: **IN, OUT, BREAK** 등

Expense / Expenses Image

- 비용 처리 관련 정보 저장
 - **ExpenseImage**는 영수증 이미지 등

NoticeBoardMessage, MessageComment

- 공지사항 및 댓글 기능 제공
 - Member와 연관



02. 프로젝트 상세

2-1. 기능 정의서

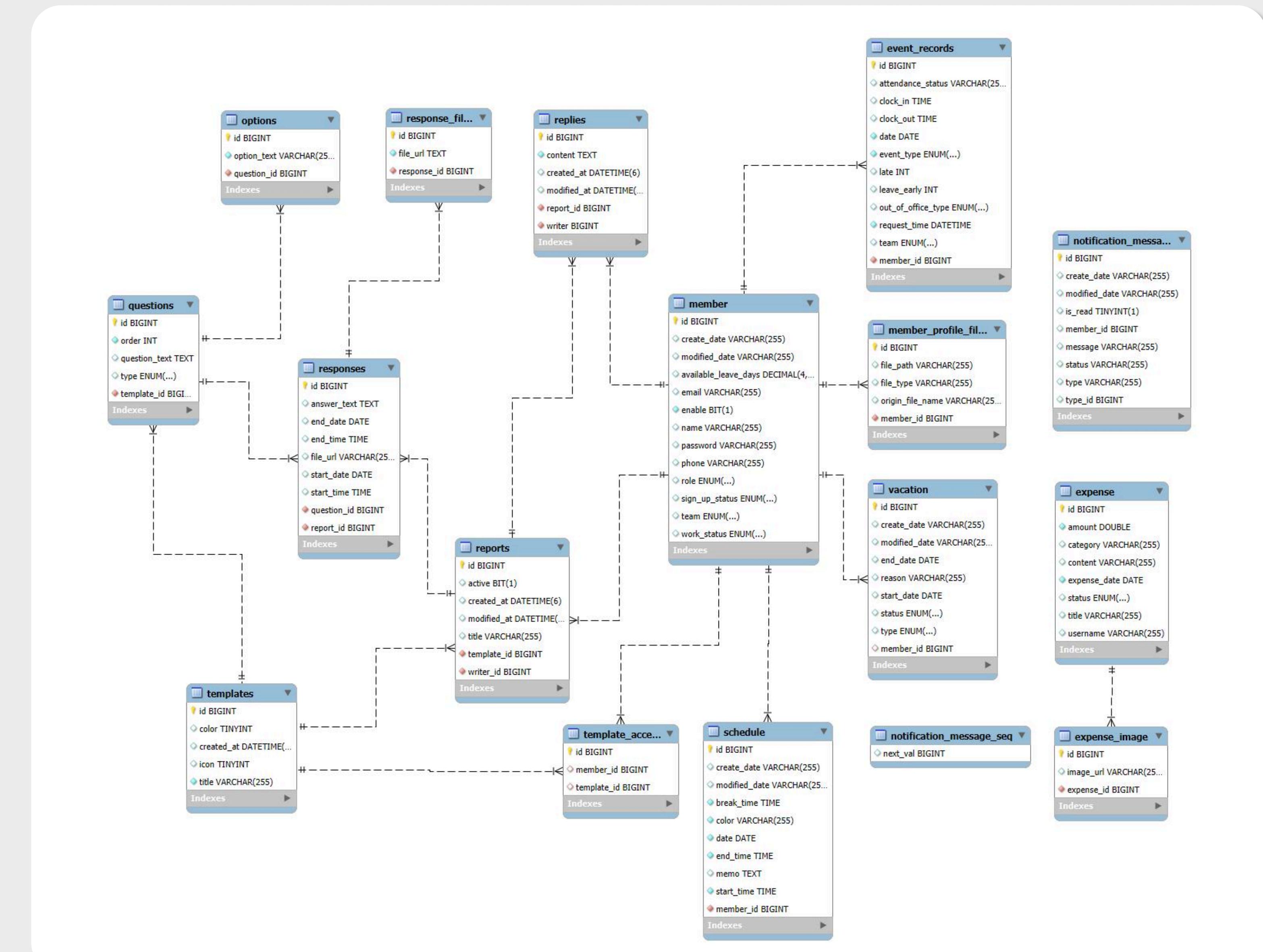
2-2. 요구사항 정의서

2-3. UML

2-4. ERD

OFFICE DATABASE

테이블 이름	간단 설명
MEMBER	직원 기본 정보 저장
MEMBER_PROFILE_FILES	직원 프로필 파일 저장
VACATION	직원 휴가 신청 기록
NOTIFICATIONS	직원 알림 정보
SCHEDULE	직원 근무 스케줄 관리
TEMPLATES	보고서 템플릿 정의
REPORTS	작성된 보고서 내용 저장
QUESTIONS	보고서 템플릿 내 질문 정의
OPTIONS	객관식 질문 선택지 저장
RESPONSES	보고서 질문에 대한 답변 저장
REPLIES	보고서 댓글 저장
EVENT_RECORD	직원 출퇴근 및 근무 상태 기록
EXPENSE	직원 비용 신청 정보
EXPENSE_IMAGE	비용 신청 관련 이미지 저장



03. 프로젝트 시연

3-1. 화면 설계 - 로그인 및 회원 관리

3-2. 화면 설계 - 출퇴근

3-3. 화면 설계 - 스케줄

3-4. 화면 설계 - 경비관리

3-5. 화면 설계 - 템플릿 및 보고서

3-6. 화면 설계 - 휴가 및 알림

담당자 : 최수민

03. 프로젝트 시연

3-1. 화면 설계 - 로그인 및 회원 관리

3-2. 화면 설계 - 출퇴근

3-3. 화면 설계 - 스케줄

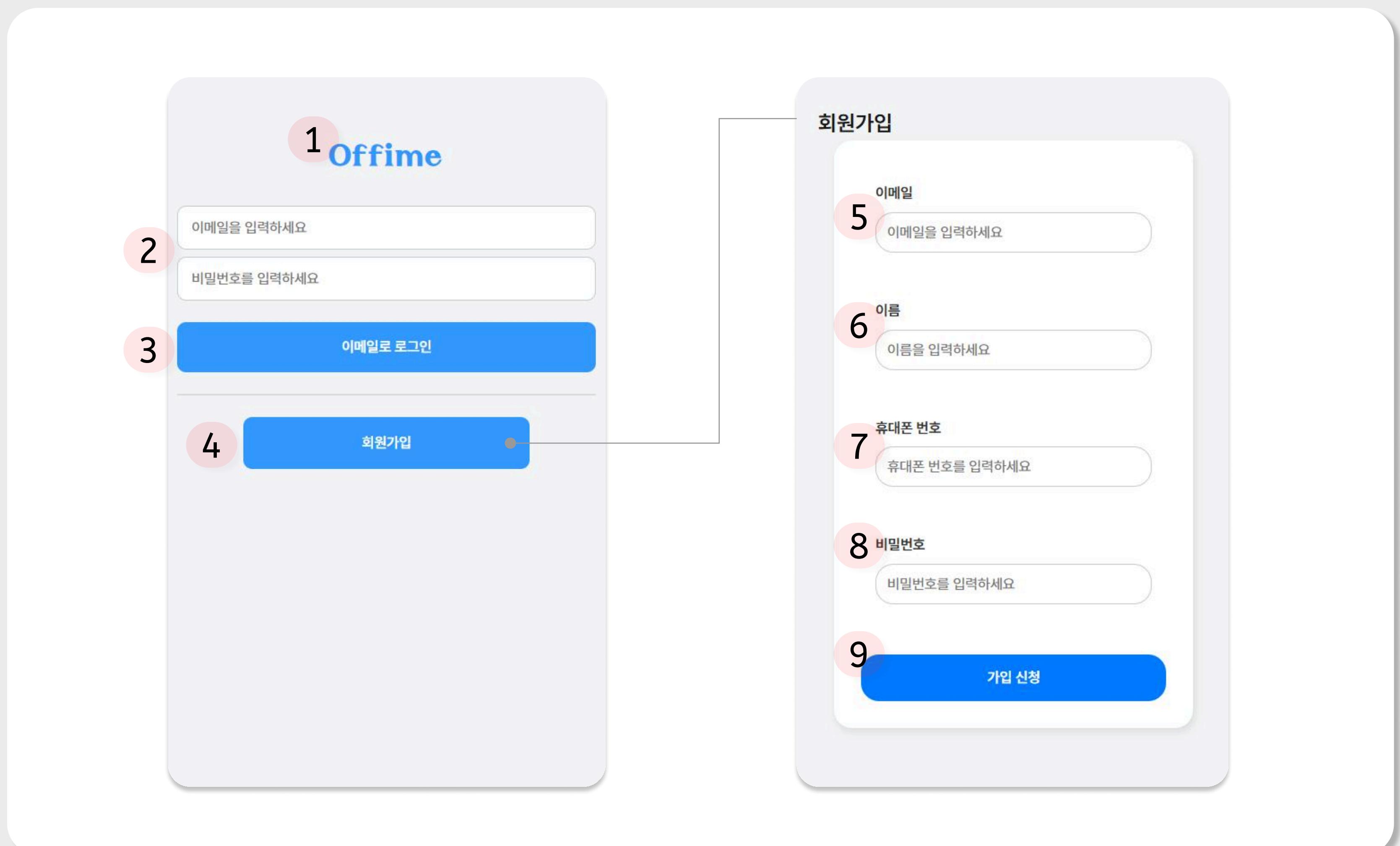
3-4. 화면 설계 - 경비관리

3-5. 화면 설계 - 템플릿 및 보고서

3-6. 화면 설계 - 휴가 및 알림

화면 제목

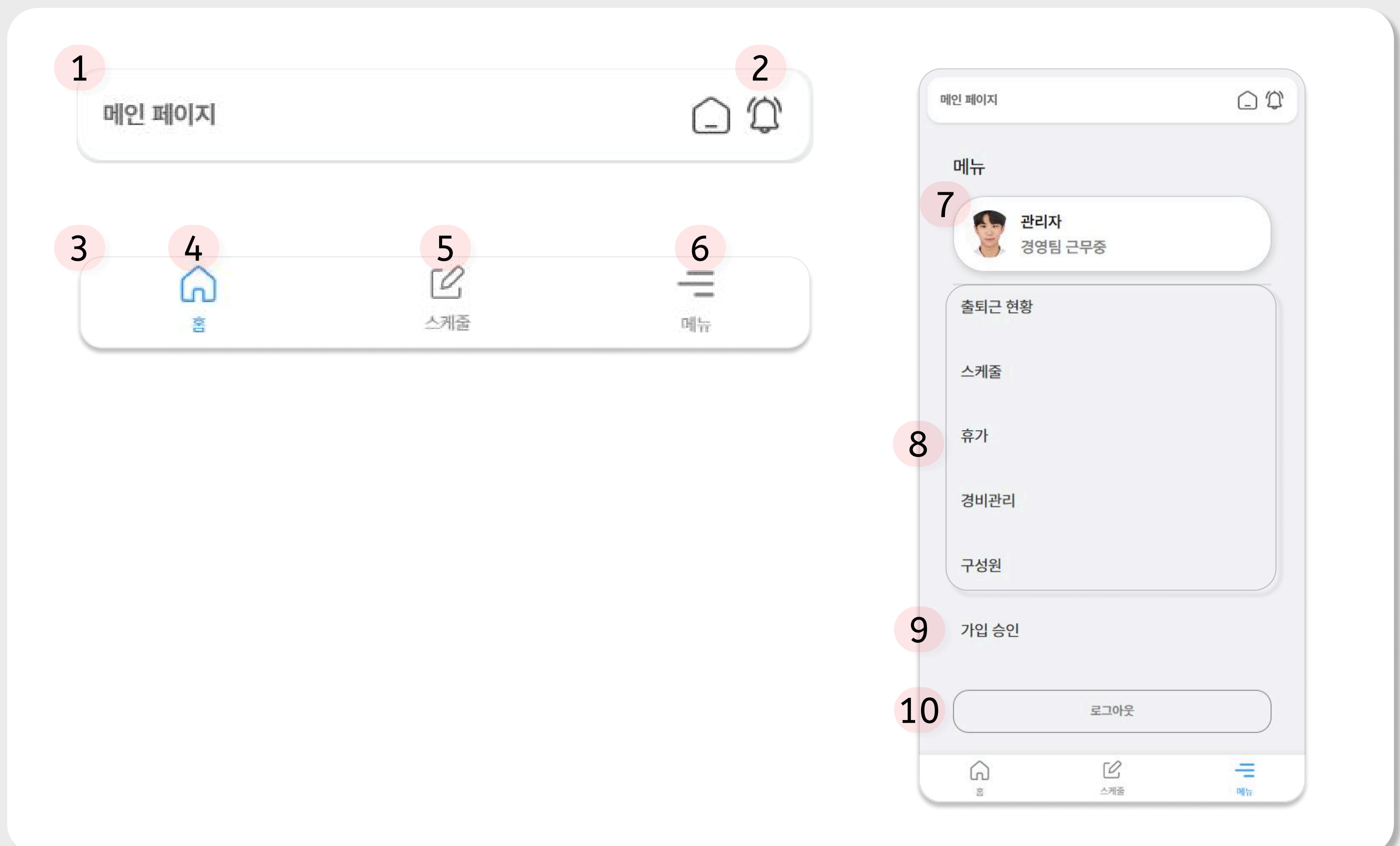
담당자 : 최수민



1. 프로젝트 로고
Offime = Office + Time
2. 이메일, 비밀번호 입력 폼
3. 가입된 회원은 이메일, 비밀번호 작성 후
로그인 가능
4. 앱을 이용하기 위해선 회원가입이 필수이며
회원가입 페이지로 이동
5. 로그인 시 이용할 이메일 주소 작성 폼
6. 사용자의 이름 작성 폼
7. 사용자의 휴대폰 번호 작성 폼
8. 로그인 시 이용할 비밀번호 설정 폼
9. 가입 시 필요한 항목 작성 후 가입 신청을
누르면 가입 신청 완료.
관리자가 가입을 승인해야 이용할 수 있다.

화면 제목

담당자 : 최수민



1. 헤더 내비게이션 바
2. 알림 내역 버튼
3. 푸터 내비게이션 바
4. 메인 화면으로 이동하는 버튼
5. 스케줄 관리 화면으로 이동하는 버튼
6. 글로벌 내비게이션 메뉴 버튼.
앱에서 사용되는 모든 메뉴를 조회
7. 로그인 한 직원의 정보를 간략하게 표시
클릭 시 마이페이지로 이동한다.
8. 앱에서 제공하는 모든 서비스를 조회하고
클릭 시 해당 페이지로 이동한다.
9. 관리자에게만 활성화되는 메뉴
가입을 신청한 직원 목록을 조회할 수 있다.
10. 로그아웃 버튼

화면 제목

담당자 : 최수민

1

구성원

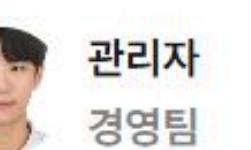
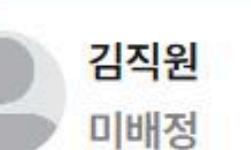
직원 검색 (이름)

관리자
경영팀

3

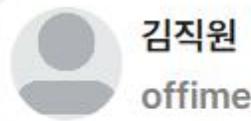
구성원

직원 검색 (이름)

관리자
경영팀김직원
미배정

2

가입 승인

김직원
offime2@user.com

1. 관리자의 승인을 받아 가입이 완료된 직원 목록 및 직원들의 마이페이지를 조회할 수 있다.

2. 가입을 신청한 직원의 목록을 조회 및 정보 확인 후 클릭 시 가입 승인을 할 수 있다.

3. 승인 대기중인 상태의 직원은 구성원 목록에 조회되지 않고, 승인 받아야만 조회할 수 있다.

화면 제목

담당자 : 최수민

Screenshot 1: Manager Profile (관리자)

- 1. Profile Picture (관리자)
- 2. Name: 최수민
- 3. Department: 경영팀
- 4. Work Status: 근무중
- 5. Department: 경영팀
- 6. Work Status: 근무중
- 7. Email: offime@offime.com
- 8. Phone Number: 01012345678
- 9. Change Password
- 10. Delete Account

Screenshot 2: Employee Profile (직원)

- 1. Profile Picture (김직원)
- 2. Name: 김직원
- 3. Department: 미배정
- 4. Work Status: 퇴근
- 5. Department: 미배정
- 6. Work Status: 퇴근
- 7. Email: offime2@user.com
- 8. Phone Number: 01078904561
- 9. Change Password
- 10. Log Out

1. 관리자 (리더)의 마이페이지.
2. 직원의 마이페이지.
3. 프로필 사진이 표시되며 클릭 시 프로필 사진을 등록할 수 있다.
4. 직원의 직급을 직접 지정할 수 있다.
관리자에게만 활성화되는 기능
5. 사번을 확인할 수 있으며 가입 순으로 결정
6. 직원의 부서를 직접 지정할 수 있다.
관리자에게만 활성화되는 기능
7. 가입 시 입력했던 이메일 정보와 휴대폰 번호를 확인할 수 있다.
8. 현재 사용하고 있는 비밀번호를 변경할 수 있다. 다른 직원에게는 변경 버튼이 노출되지 않는다.
9. 관리자가 직원을 탈퇴시키거나, 직원이 직접 탈퇴할 수 있는 기능. 완전 삭제가 아닌 비활성화로 처리된다.

담당자 : 이영현

03. 프로젝트 시연

3-1. 화면 설계 - 로그인 및 회원 관리

3-2. 화면 설계 - 출퇴근

3-3. 화면 설계 - 스케줄

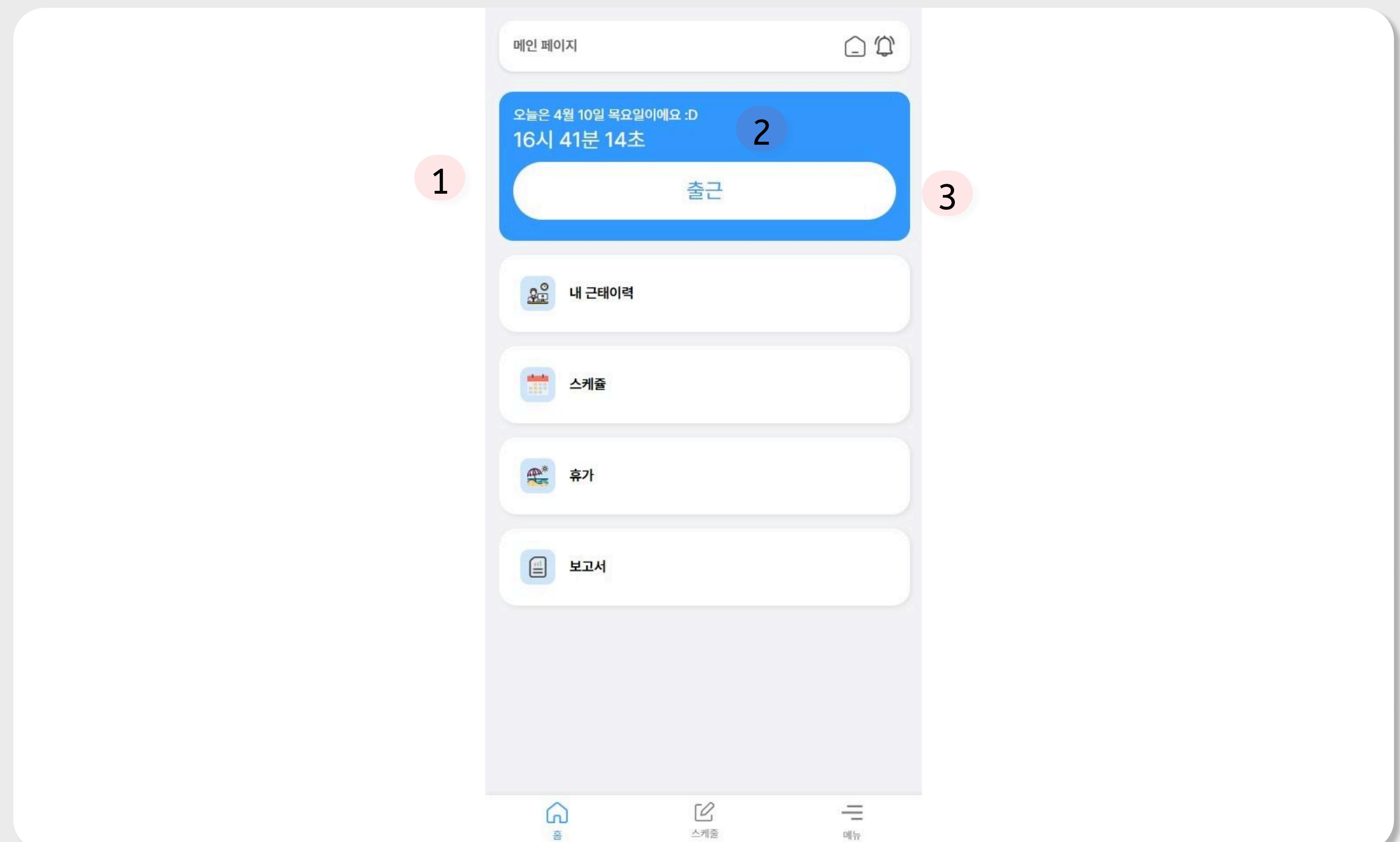
3-4. 화면 설계 - 경비관리

3-5. 화면 설계 - 템플릿 및 보고서

3-6. 화면 설계 - 휴가 및 알림

기본 배너

담당자 : 이영현



1. 기본 배너

- 메인 페이지 진입 시 **axios** 비동기 통신으로 **member** 객체의 **workStatus**를 조회한다. 결과가 '퇴근' 일 경우에 기본 배너가 렌더링된다.
- 새로 가입한 직원은 기본 상태가 '퇴근' 이므로 기본 배너가 보인다.

2. 기본 배너의 문구

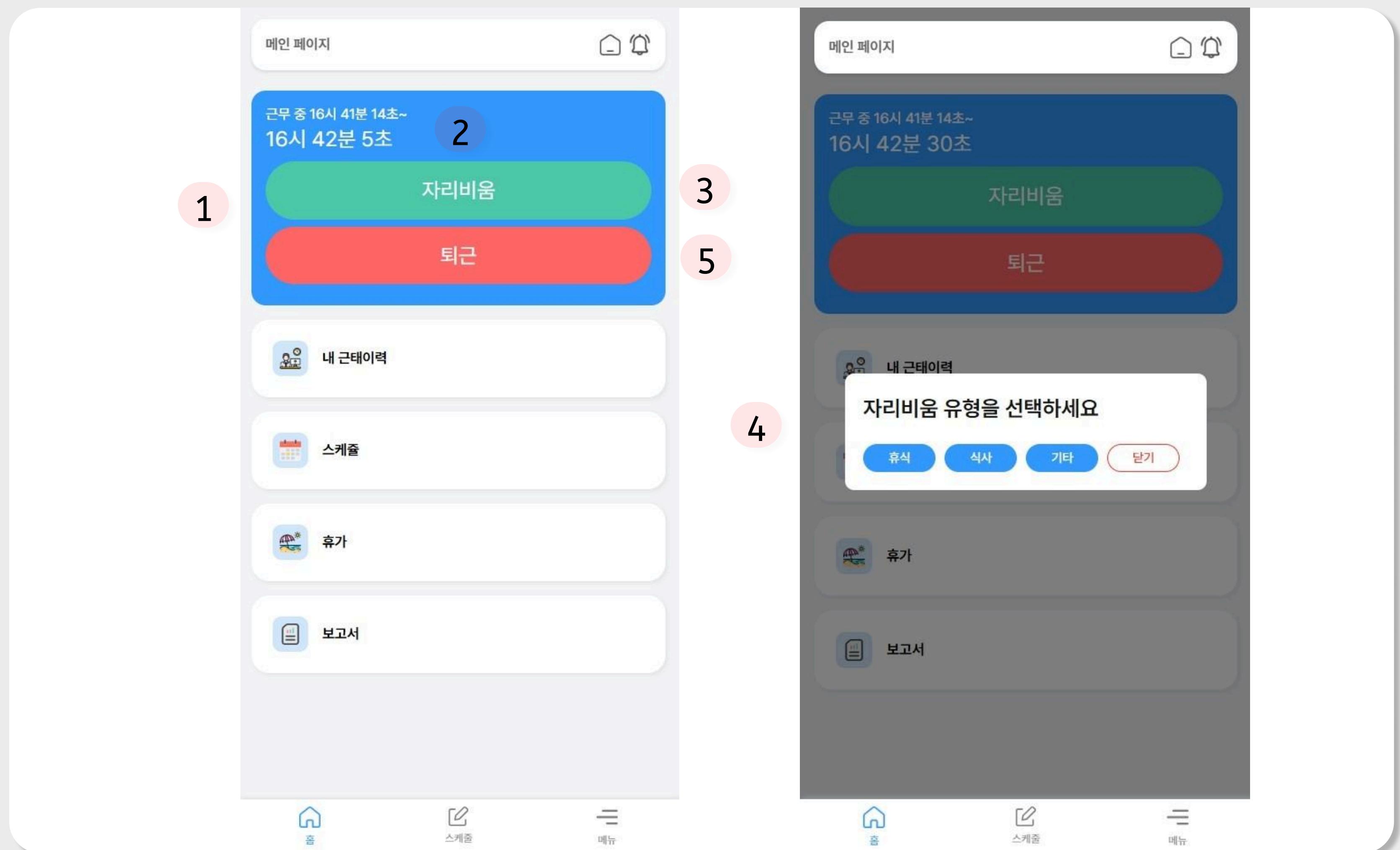
오늘의 날짜와 요일, 현재 시간을 표시한다. 1초마다 함수를 실행해서 매초 시계를 갱신한다.

3. 출근 버튼

- 클릭 시 **geoLocation API**가 브라우저에서 위치 권한을 받아 **axios** 비동기 통신으로 현재 위치의 위도와 경도를 백엔드 서버로 보낸다.
- 현재 위도와 경도에 **Haversine Formula**를 사용해 지정된 회사 좌표와의 거리를 구한다.
- 거리가 지정된 허용 오차범위(**50m**) 이내인지 검증한다.
- 출근시간(**09시**) 이후인 경우 지각한 시간과 함께 저장한다.
- 직원의 **workStatus**를 '근무 중'으로 변경한다.

근무 중 배너

담당자 : 이영현



1. 근무 중 배너

메인 페이지 진입 시 **axios** 비동기 통신으로 **member** 객체의 **workStatus**를 조회한다. 결과가 '근무 중' 일 경우에 근무 중 배너가 렌더링된다.

2. 근무 중 배너의 문구

출근 시간과 현재 시간을 표시한다. **1초마다** 함수를 실행해서 매초 시계를 갱신한다.

3. 자리비움 버튼

클릭 시 자리비움 유형 선택창이 팝업된다.

4. 자리비움 유형 선택창

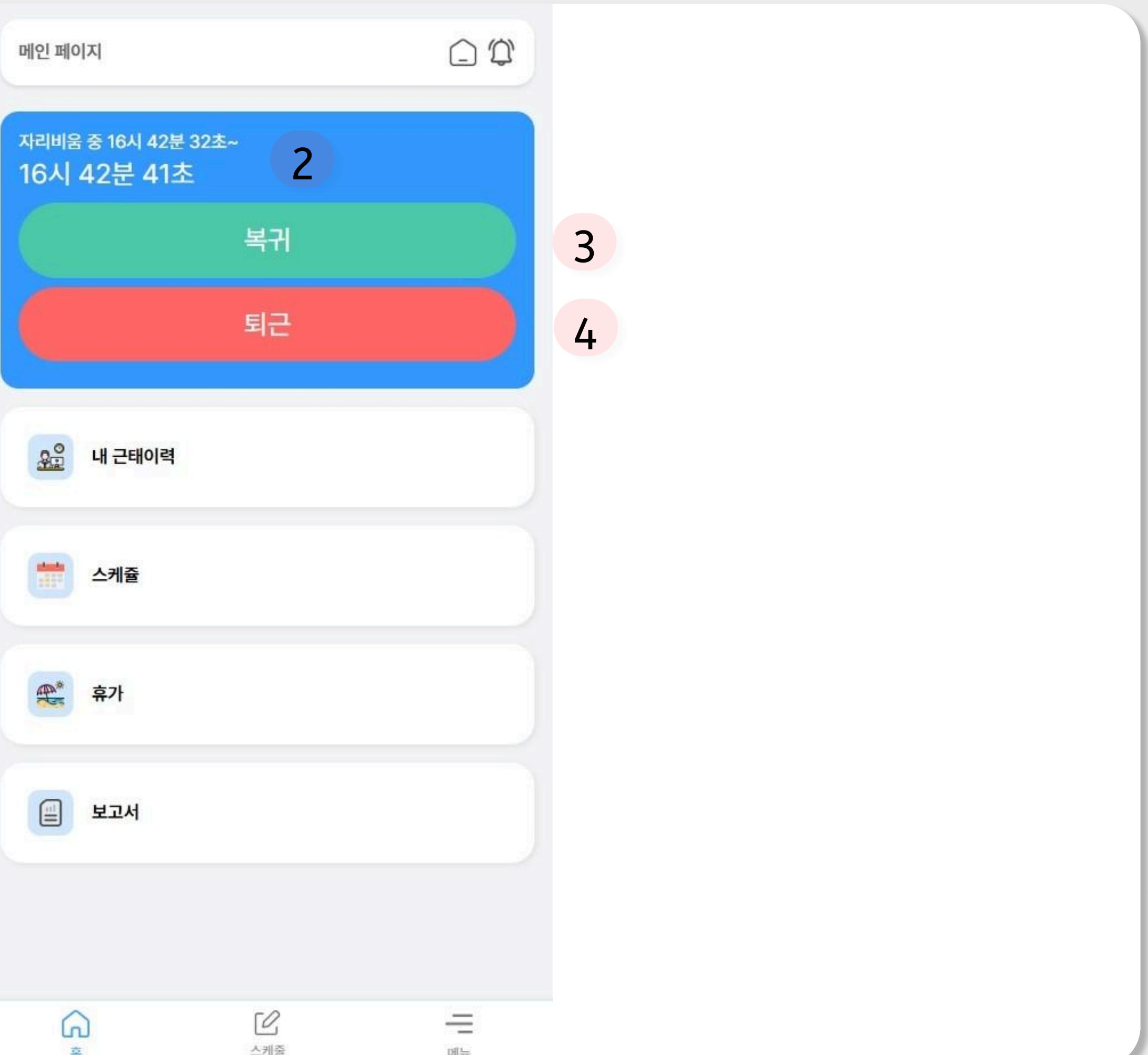
- 각각의 자리비움의 유형을 클릭 시 **axios** 비동기 통신으로 자리비움 요청을 백엔드 서버로 보낸다.
- 직원의 자리비움 시간과 유형이 저장된다.
- 직원의 **workStatus**를 '자리비움 중'으로 변경한다.

5. 퇴근 버튼

- 클릭 시 **axios** 비동기 통신으로 퇴근 요청을 백엔드 서버로 보낸다.
- 퇴근시간(**18시**) 이전인 경우 조퇴한 시간과 함께 퇴근 기록을 저장한다.
- 직원의 **workStatus**를 '퇴근'으로 변경한다.

자리비움 중 배너

담당자 : 이영현



1. 자리비움 중 배너

메인 페이지 진입 시 **axios** 비동기 통신으로 **member** 객체의 **workStatus**를 조회한다. 결과가 '자리비움 중' 일 경우에 자리비움 중 배너가 렌더링된다.

2. 자리비움 중 배너의 문구

자리비움 요청 시간과 현재 시간을 표시한다. **1초마다** 함수를 실행해서 매초 시계를 갱신한다.

3. 복귀 버튼

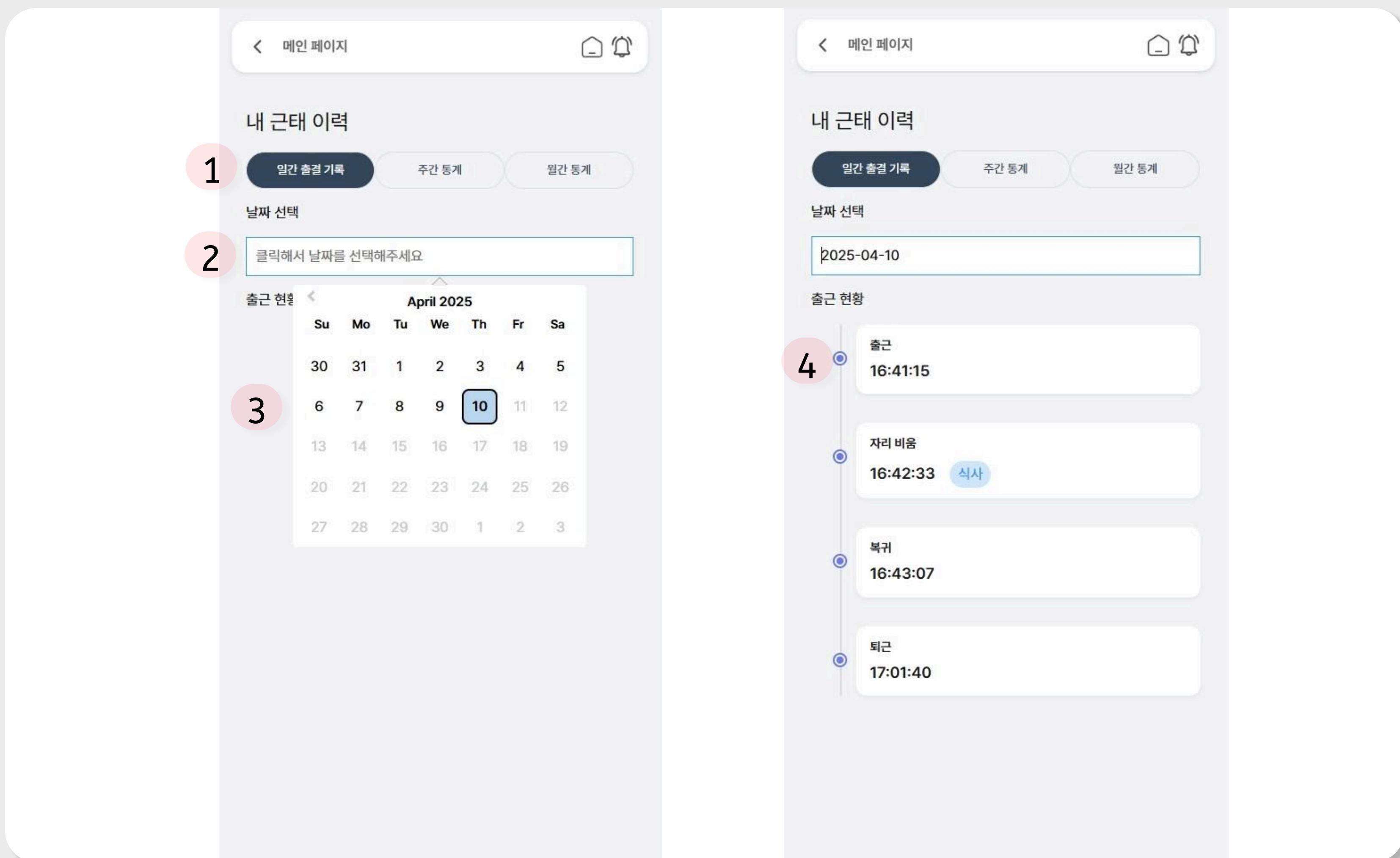
- 클릭 시 **geoLocation API**가 **axios** 비동기 통신으로 현재 위치의 위도와 경도를 백엔드 서버로 보낸다.
- 현재 위도와 경도에 **Haversine Formula**를 사용해 지정된 회사 좌표와의 거리를 구한다.
- 거리가 지정된 허용 오차범위(**50m**) 이내라면 직원의 복귀 기록이 저장된다.
- 직원의 **workStatus**를 '근무 중'으로 변경한다.

4. 퇴근 버튼

- 클릭 시 **axios** 비동기 통신으로 퇴근 요청을 백엔드 서버로 보낸다.
- 퇴근시간(**18시**) 이전인 경우 조퇴한 시간과 함께 퇴근 기록을 저장한다.
- 직원의 **workStatus**를 '퇴근'으로 변경한다.

내 근태 이력 - 일간 출결 기록

담당자 : 이영현



1. 일간 출결 기록 버튼

특정 날짜의 내 근태 기록을 조회할 수 있다.

2. 날짜 선택

클릭시 **react-datepicker**가 팝업된다.

3. react-datepicker 달력

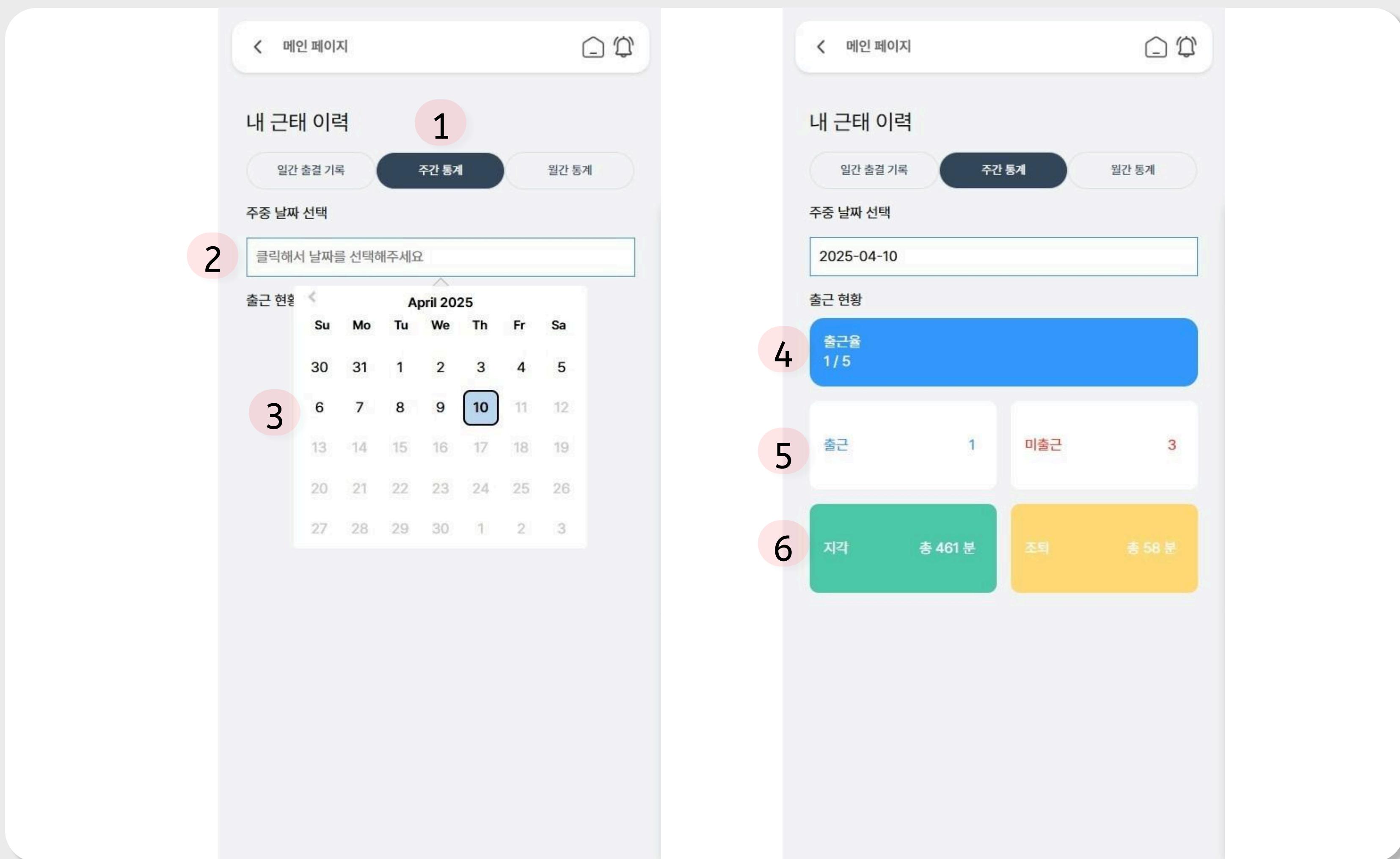
- 원하는 날짜를 선택하면 **axios** 비동기 통신으로 해당 일자의 내 근태 기록을 요청한다.
- 최대 선택 가능 일자는 오늘까지.

4. 내 근태 기록 데이터

- 데이터가 없다면 '출결 기록이 없습니다' 메시지를 출력한다.
- 데이터가 있다면 근태 유형과 시간을 순서대로 표시한다.

내 근태 이력 - 주간 통계

담당자 : 이영현

**1. 주간 통계 버튼**

특정 주의 내 근태 통계를 조회할 수 있다.

2. 날짜 선택클릭시 **react-datepicker**가 팝업된다.**3. react-datepicker 달력**

- 원하는 날짜를 선택하면 **axios** 비동기 통신으로 선택일을 포함하고 주말을 제외한 해당 주 전체의 내 근태 기록을 요청한다.
- 최대 선택 가능 일자는 오늘까지.

4. 출근율

(출근한 일 수) / (일주일 중 주말을 제외한 평일) 을 표시한다.

5. 출근 / 미출근

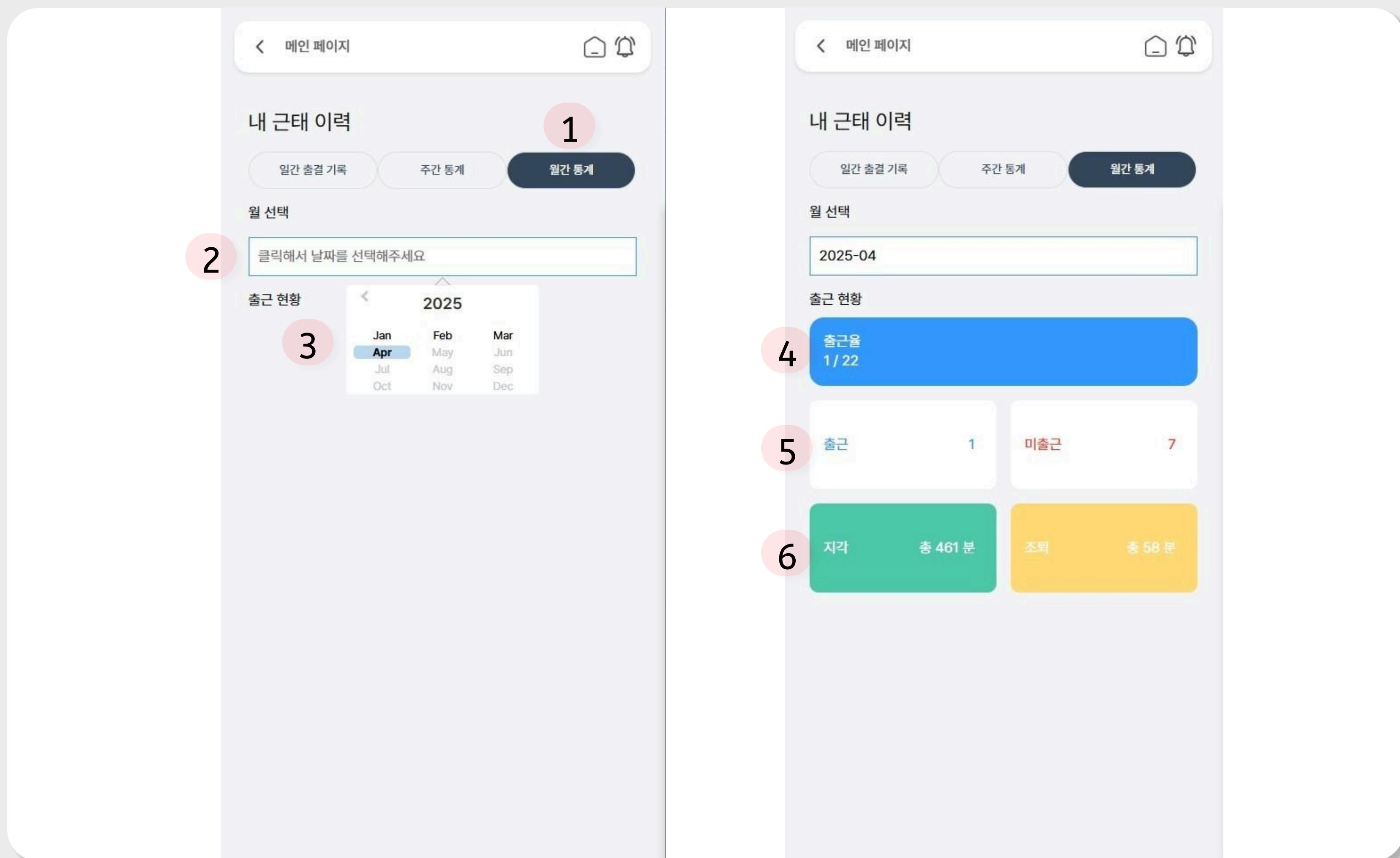
(출근한 일 수)와 (미출근한 일 수)를 표시한다.

6. 지각 / 조퇴

(지각한 총 분 수)와 (조퇴한 총 분 수)를 표시한다.

내 근태 이력 - 월간 통계

담당자 : 이영현



1. 월간 통계 버튼

특정 월의 내 근태 통계를 조회할 수 있다.

2. 날짜 선택

클릭시 react-datepicker(MonthYear)가 팝업된다.

3. react-datepicker(MonthYear) 달력

- 원하는 월을 선택하면 axios 비동기 통신으로 해당 월 전체의 내 근태 기록을 요청한다.
- 최대 선택 가능 월은 오늘까지.

4. 출근율

(출근한 일 수) / (한달 중 주말을 제외한 평일)을 표시한다.

5. 출근 / 미출근

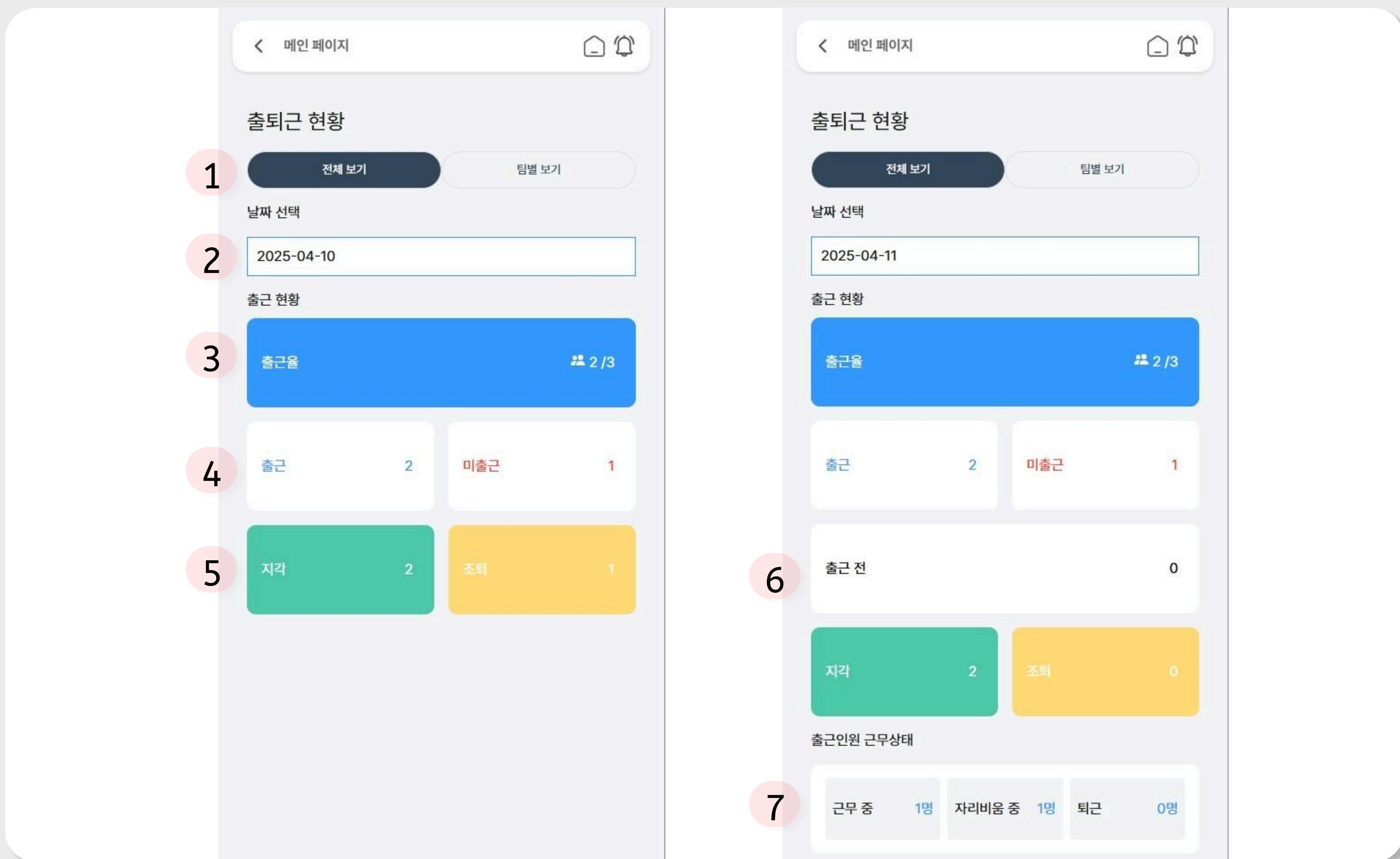
(출근한 일 수)와 (미출근한 일 수)를 표시한다.

6. 지각 / 조퇴

(지각한 총 분 수)와 (조퇴한 총 분 수)를 표시한다.

출퇴근 현황 - 전체 보기

담당자 : 이영현



1. 전체 보기 버튼

전체 직원의 출퇴근 현황을 조회할 수 있다.

2. 날짜 선택

클릭시 react-datepicker로 날짜를 선택할 수 있다.

3. 출근율

(출근한 직원 수) / (전체 직원 수)를 표시한다.

4. 출근 / 미출근

(출근한 직원 수)와 (미출근한 직원 수)를 표시한다. 해당일의 휴가자는 미출근자로 계산된다.

5. 지각 / 조퇴

(지각한 직원 수)와 (조퇴한 직원 수)를 표시한다.

6. (오늘 조회만 렌더링) 출근 전

- 출근시간(09)시 이전의 경우, 아직 출근하지 않은 직원은 '출근 전' 상태로 표시된다.
- 출근시간(09)시 이후의 경우, 아직 출근하지 않은 직원은 '미출근' 상태로 표시된다.

7. (오늘 조회만 렌더링) 출근인원 근무상태

출근한 직원의 현재 근무상태를 표시한다.

출퇴근 현황 - 팀별 보기

담당자 : 이영현

출퇴근 현황 - 팀별 보기

1. 팀별 보기 버튼

팀별 직원의 출퇴근 현황을 조회할 수 있다.

2. 팀 선택

팀을 선택할 수 있다.

3. 날짜 선택

클릭시 react-datepicker로 날짜를 선택할 수 있다.

4. 출근율

(출근한 직원 수) / (전체 직원 수)를 표시한다.

5. 출근 / 미출근

(출근한 직원 수)와 (미출근한 직원 수)를 표시한다.
해당일의 휴가자는 미출근자로 계산된다.

6. 지각 / 조퇴

(지각한 직원 수)와 (조퇴한 직원 수)를 표시한다.

7. (오늘 조회만 렌더링) 출근 전

- 출근시간(09)시 이전의 경우, 아직 출근하지 않은 직원은 '출근 전' 상태로 표시된다.
- 출근시간(09)시 이후의 경우, 아직 출근하지 않은 직원은 '미출근' 상태로 표시된다.

7. (오늘 조회만 렌더링) 출근인원 근무상태

출근한 직원의 현재 근무상태를 표시한다.

담당자 : 김성우

03. 프로젝트 시연

3-1. 화면 설계 - 로그인 및 회원 관리

3-2. 화면 설계 - 출퇴근

3-3. 화면 설계 - 스케줄

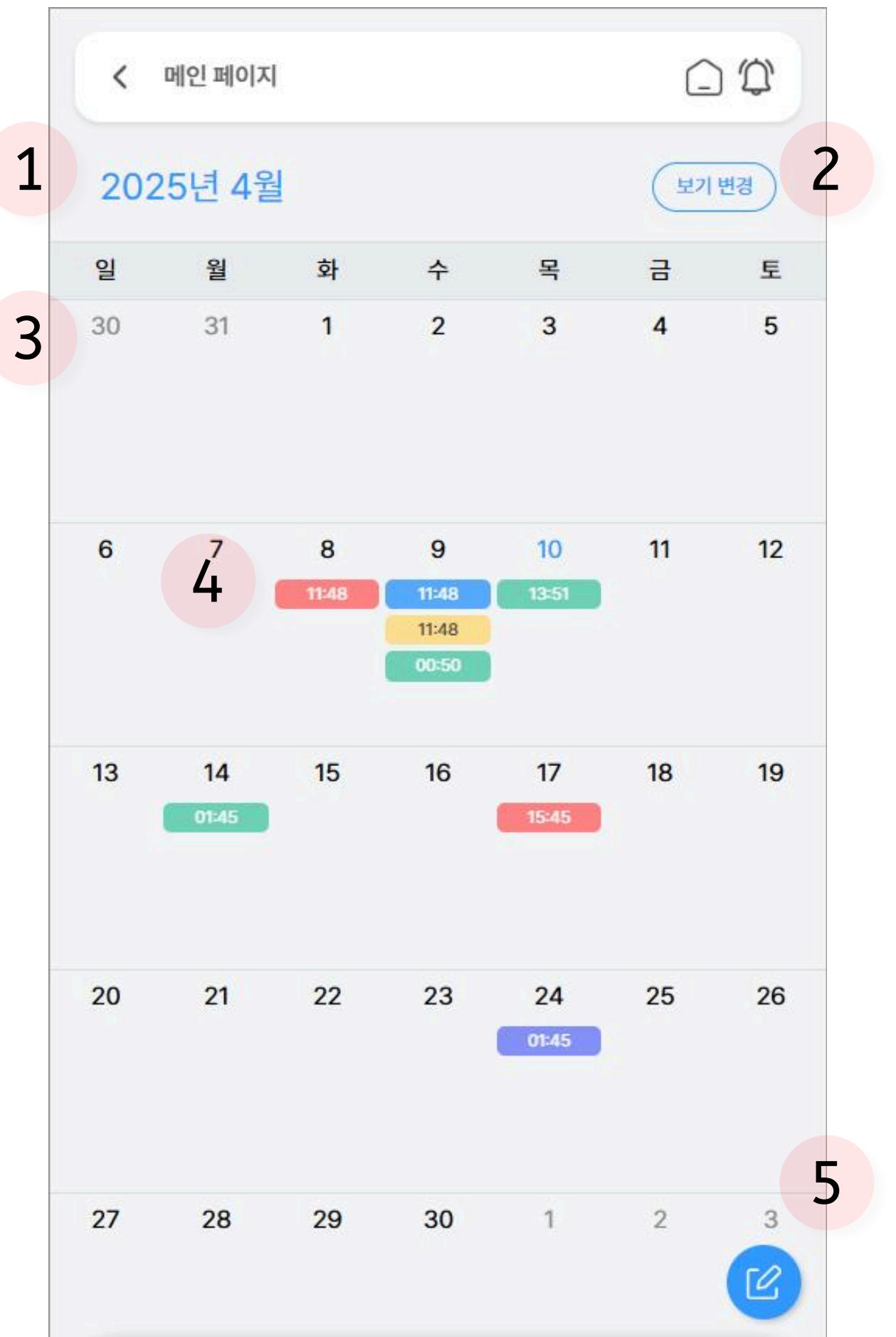
3-4. 화면 설계 - 경비관리

3-5. 화면 설계 - 템플릿 및 보고서

3-6. 화면 설계 - 휴가 및 알림

스케줄 캘린더

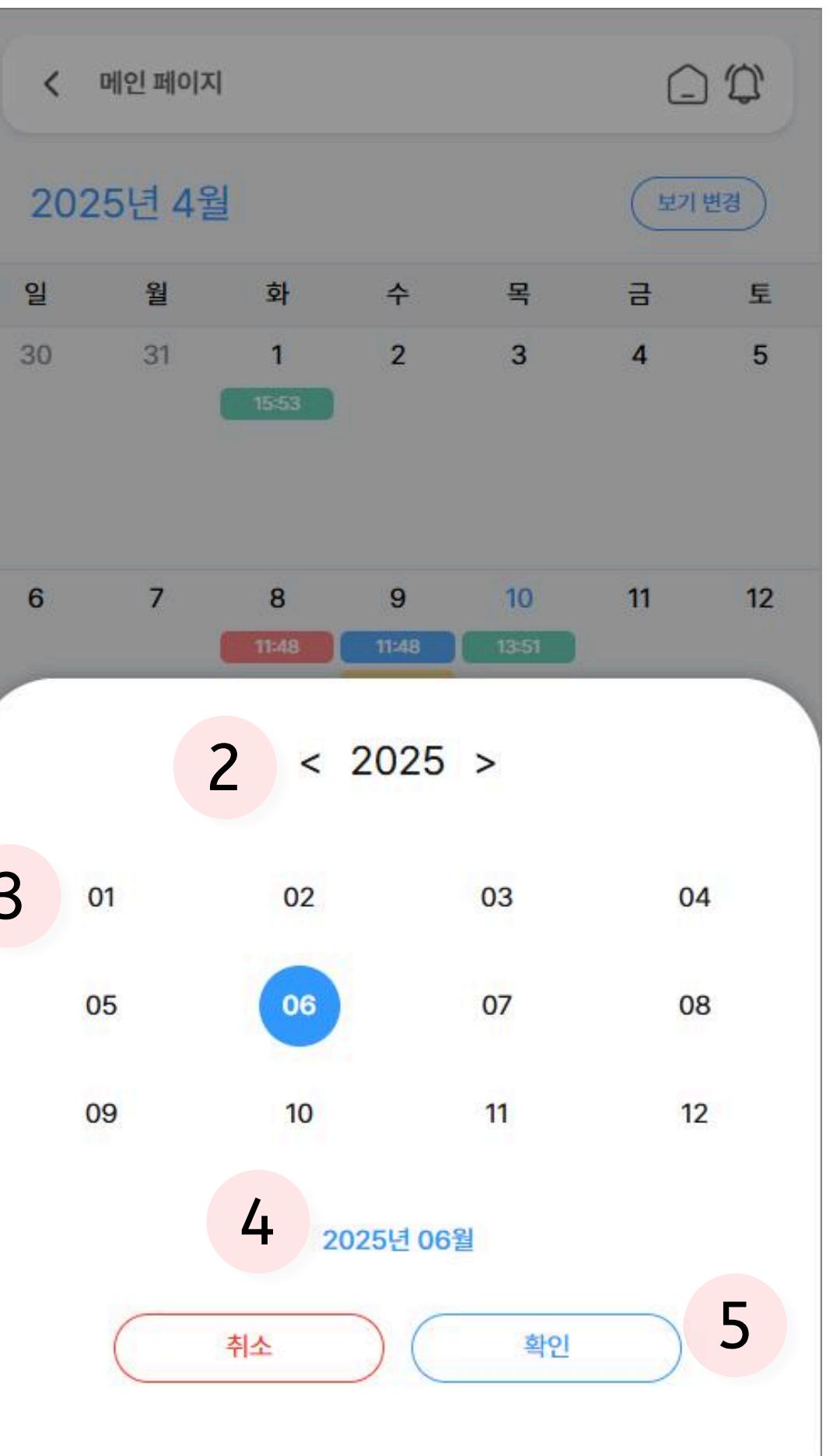
담당자 : 김성우



1. 연, 월 표시
클릭 시 연, 월 변경 창 팝업
2. 보기 변경 버튼
클릭 시 스케줄 시간에서 메모로 표시
3. 캘린더 뷰
전, 후 월의 날짜는 회색으로 표시
오늘 날짜는 파란색으로 표시
4. 스케줄
등록된 스케줄을 해당 날짜에 표시
클릭 시 해당 스케줄 상세 내용 조회
5. 스케줄 등록 버튼
클릭 시 스케줄 등록 폼 팝업

캘린더 연, 월 변경 창

담당자 : 김성우



1. 연, 월 변경 창 활성화
어두운 영역 클릭시 비활성화
2. 연도 선택
좌, 우 화살표로 연도 변경
3. 월 선택
클릭 하여 월 변경. 선택된 월은 파란색으로 활성화
4. 선택된 날짜
선택한 연, 월을 표시
5. 취소, 확인 버튼

스케줄 등록 폼 / 등록 폼 상세

담당자 : 김성우

1. 스케줄 등록 폼

2. 선택한 날짜 활성화

3. 스케줄 등록 취소 버튼

4. 스케줄을 등록할 날짜
캘린더의 날짜를 클릭하여 변경

5. 스케줄 색상 선택
활성화된 색상은 검정 태두리로 표시

6. 시작 시간, 종료 시간 입력 필드
클릭 시 시간 선택 옵션 드롭다운

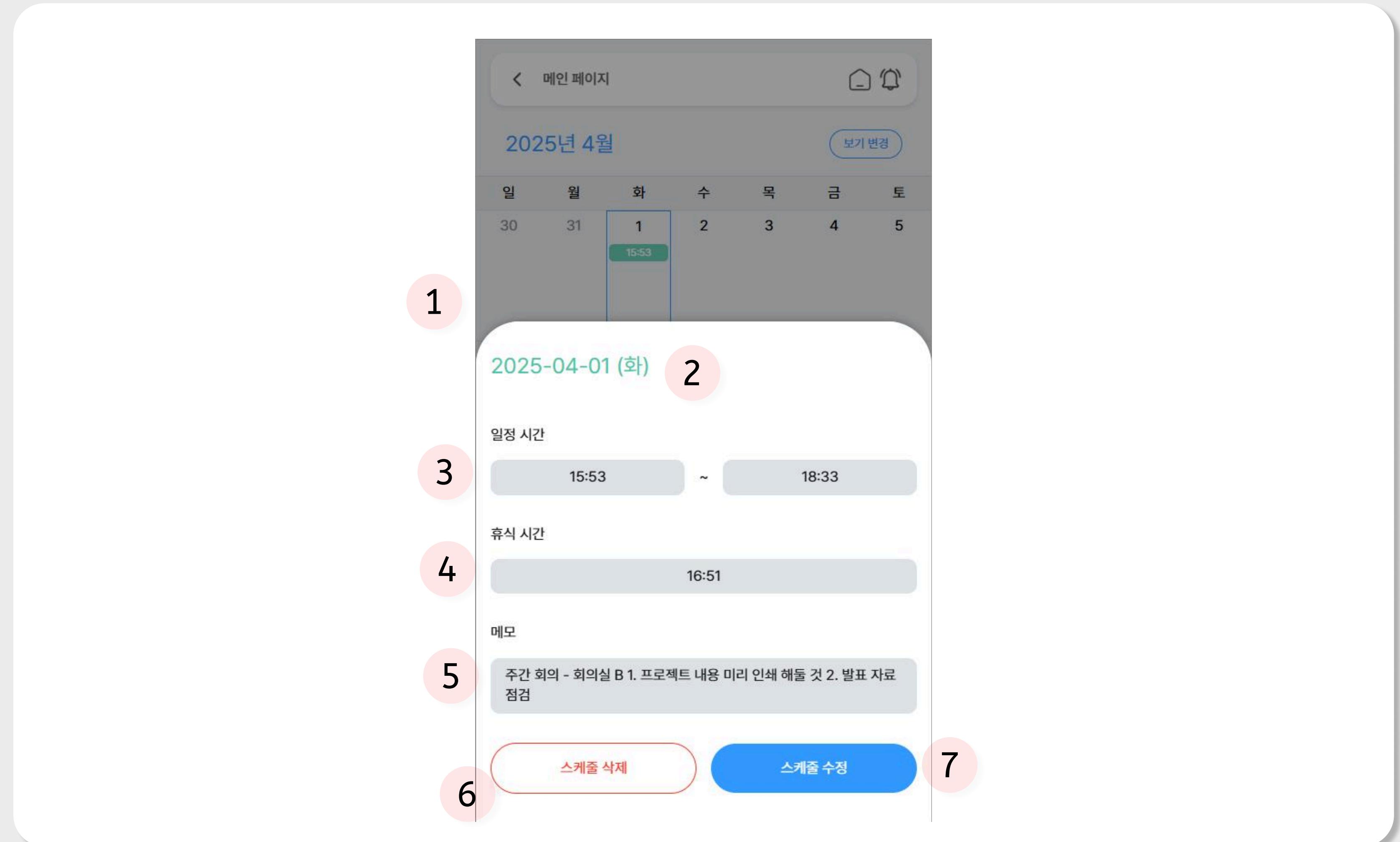
7. 휴식 시간 입력 필드
클릭 시 시간 선택 옵션 드롭다운

8. 메모 입력 필드
스케줄 상세 내용을 작성하는 필드

9. 스케줄 등록 버튼

스케줄 상세

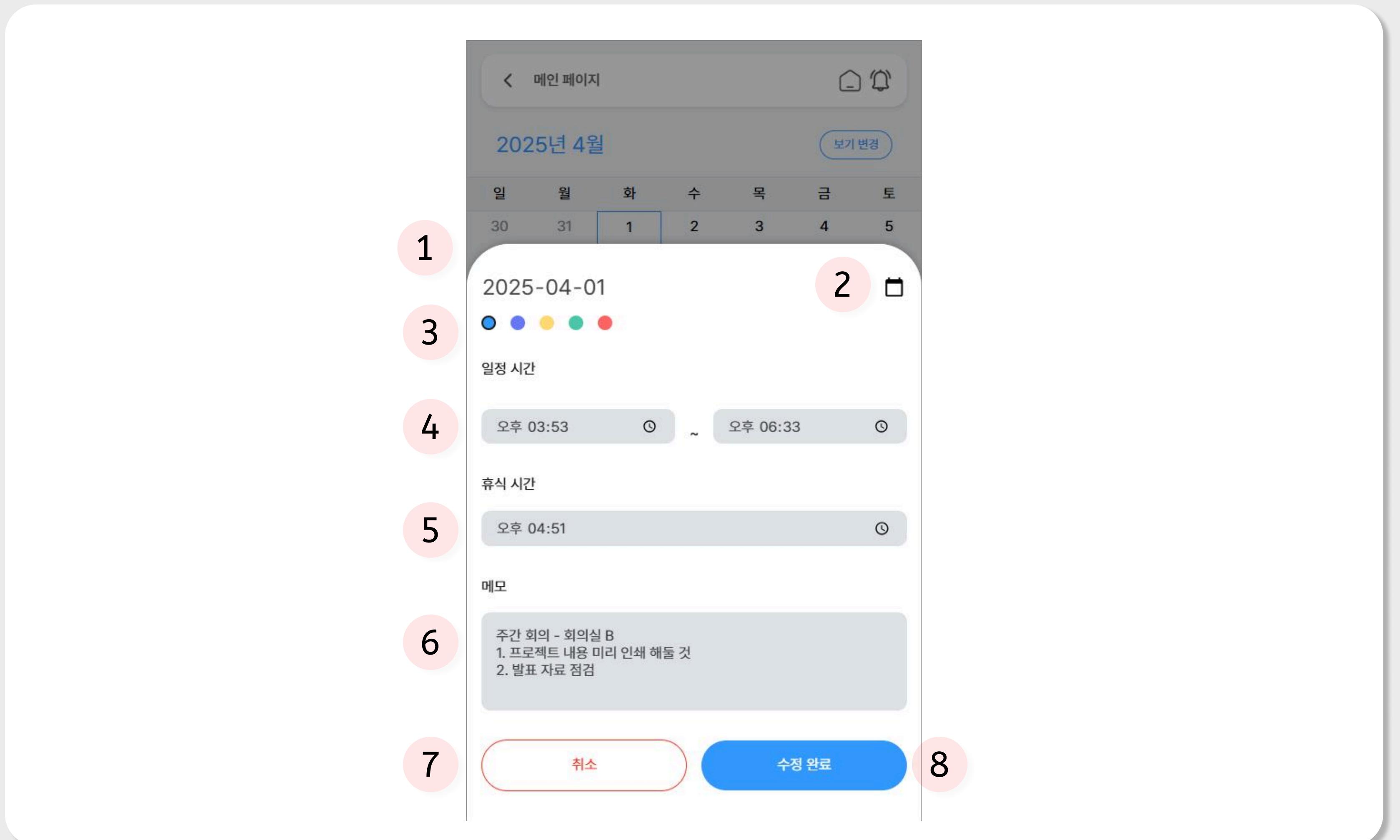
담당자 : 김성우



1. 스케줄 상세 창
2. 등록된 날짜
스케줄 색상과 같은 색으로 표시
3. 시작 시간, 종료 시간 표시
4. 휴식시간 표시
5. 메모 내용 표시
6. 스케줄 삭제 버튼
클릭시 삭제 여부를 다시 확인
7. 스케줄 수정 버튼
클릭시 스케줄 수정 폼 활성화

스케줄 수정

담당자 : 김성우



1. 스케줄 수정 폼
2. 날짜 수정 필드
아이콘 클릭시 수정 캘린더 팝업
3. 스케줄 색상 변경 필드
4. 시작 시간, 종료 시간 변경 필드
5. 휴식 시간 변경 필드
6. 메모 변경 필드
7. 스케줄 수정 취소 버튼
8. 스케줄 수정 완료 버튼

담당자 : 박시진

03. 프로젝트 시연

3-1. 화면 설계 - 로그인 및 회원 관리

3-2. 화면 설계 - 출퇴근

3-3. 화면 설계 - 스케줄

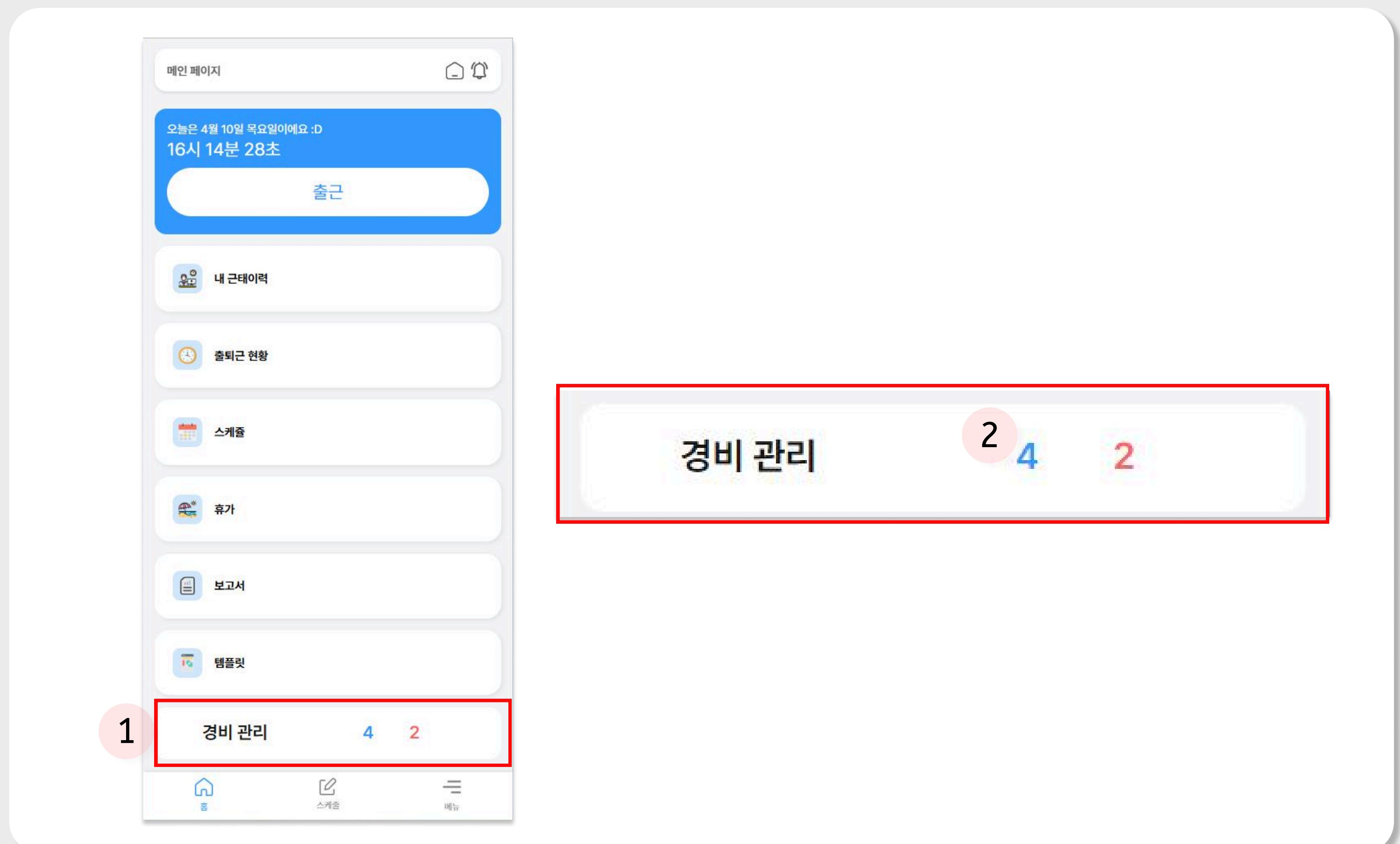
3-4. 화면 설계 - 경비관리

3-5. 화면 설계 - 템플릿 및 보고서

3-6. 화면 설계 - 휴가 및 알림

경비관리 메인 버튼

담당자 : 박시진



1. 경비관리 메인 버튼

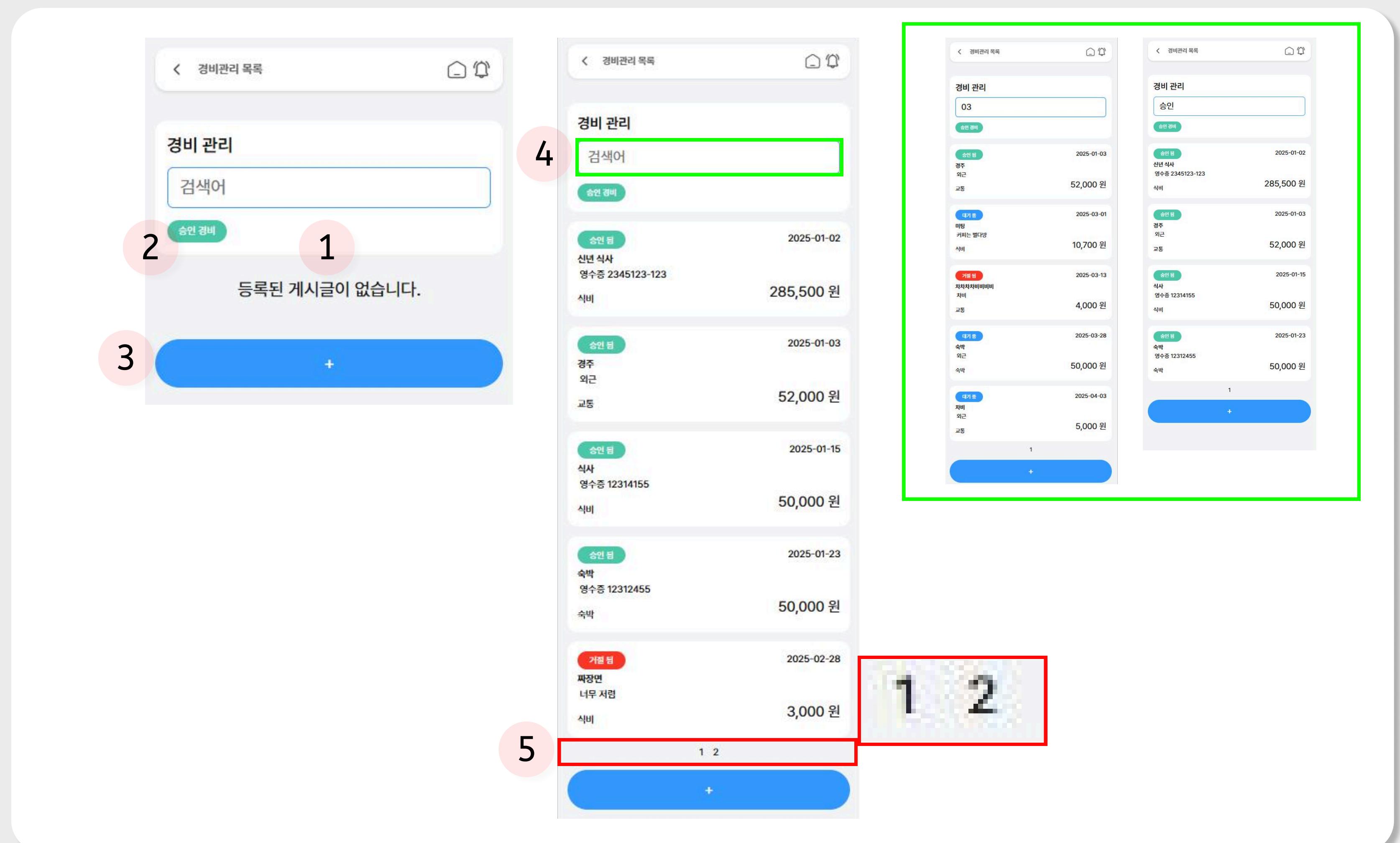
메인화면의 경비관리 버튼으로 관리자만 보임
클릭시 경비관리 목록으로 이동

2. 대기중인 경비 수

파란색은 대기 중인 경비 게시물 수를 나타냄
붉은색은 거절 된 경비 게시물 수를 나타냄

경비관리 목록

담당자 : 박시진



1. 경비관리 목록

작성 된 경비 게시물이 나타남

2. 경비관리 승인 경비 버튼

클릭 시 경비 승인 목록 페이지로 이동

3. 경비관리 작성 버튼

클릭 시 경비관리 작성페이지로 이동

4. 경비관리 검색

경비관리 관련 어떤 검색이든 모두 가능

5. 경비관리 목록 페이지네이션

게시물의 수가 5개가 되면 다음 게시물은 다음 페이지로 넘어가며 페이지로 넘어갈 수 있는 버튼이 생김

경비관리 작성

담당자 : 박시진

1. 제목

2. 사진

3. 금액
금액란 추가

4. 날짜
연도-월-일

5. 작성

1. 경비관리 작성

경비관리 게시물 작성 가능

제목, 금액, 날짜, 유형, 내용은 필수 작성

2. 사진 추가

사진을 추가 할 수 있음

3. 경비 관리 작성 금액란

금액란을 추가 할 수 있으며 삭제도 가능

금액을 적으면 합계 금액을 볼 수 있음

4. 날짜 선택

날짜를 선택 가능

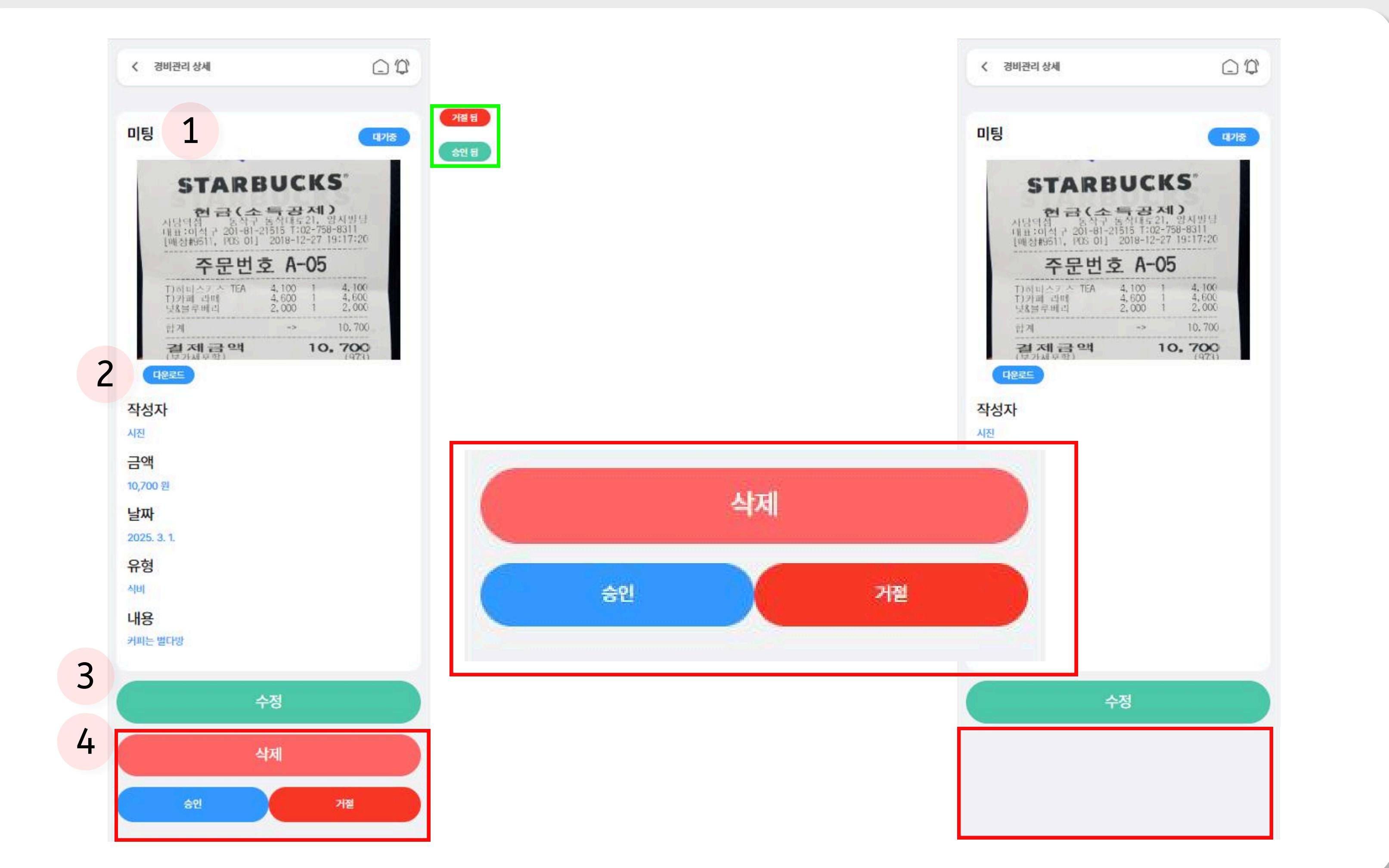
5. 경비관리 작성 버튼

버튼 클릭 시 게시물이 작성 오나로 되며

경비관리 목록으로 이동

경비관리 상세

담당자 : 박시진



1. 경비관리 상세

경비관리 게시물의 상세 내용을 볼 수 있음

2. 이미지 다운 버튼

클릭 시 이미지를 다운 받을 수 있음

3. 수정 페이지 이동 버튼

클릭 시 경비관리 수정페이지로 이동

4. 관리자용 버튼

삭제, 승인, 거절 버튼으로 관리자만 보임

경비관리 수정

담당자 : 박시진

The screenshot displays the expense management modification screen. It includes:

- 제목**: 미팅 (Title)
- 파일 선택** button (File Selection) highlighted with a red box.
- 금액**: 10,700 (Amount) highlighted with a green box.
- 날짜**: 2025-03-01 (Date).
- 유형**: 식비 (Type: Food).
- 내용**: 커피는 별다방 (Content: Starbucks coffee).
- 수정** button (Modify) at the bottom.

1. 경비관리 수정

상세 내용을 볼 수 있으며 수정이 가능함

2. 사진 추가 버튼

사진 선택 창이 열리며 사진 추가가 가능

3. 사진 삭제 버튼

기존의 사진을 삭제 할 수 있음

4. 경비관리 수정 버튼

클릭 시 수정이 완료 되고 상세페이지로 이동

경비관리 승인 내역

담당자 : 박시진

The screenshots illustrate the 'Expense Approval' application interface:

- Screenshot 1:** Shows the total approved amount as 437,500 원. It lists three expense items: 신년 식사 (285,500 원), 경주 (52,000 원), and 식사 (50,000 원). The date range is set to 2025년 1월.
- Screenshot 2:** Shows the total approved amount as 335,500 원. It lists two expense items: 신년 식사 (285,500 원) and 식사 (50,000 원). The date range is set to 2025년 1월.
- Screenshot 3:** Shows the total approved amount as 52,000 원 for transportation. It lists one expense item: 경주 (52,000 원). The date range is set to 2025년 4월.

1. 경비관리 승인 내역 목록
경비관리 승인 내역을 볼 수 있음

2. 승인 내역 카테고리
승인 내역을 카테고리 별로 볼 수 있음

3. 승인 내역 날짜
날짜를 선택 할 수 있음

담당자 : 손수용

03. 프로젝트 시연

3-1. 화면 설계 - 로그인 및 회원 관리

3-2. 화면 설계 - 출퇴근

3-3. 화면 설계 - 스케줄

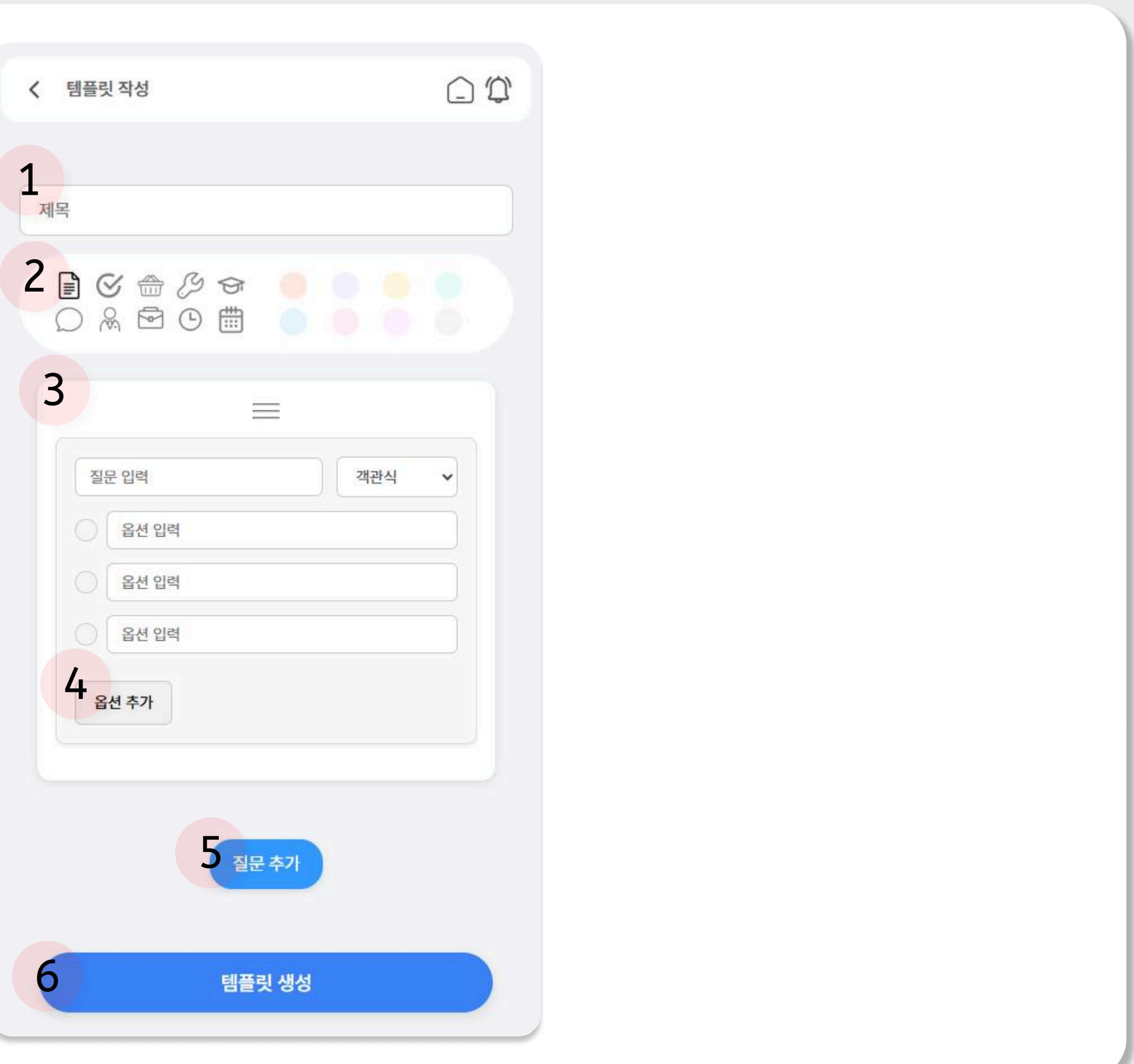
3-4. 화면 설계 - 경비관리

3-5. 화면 설계 - 템플릿 및 보고서

3-6. 화면 설계 - 휴가 및 알림

템플릿 생성 페이지

담당자 : 손수용



1. 제목 입력칸

제목을 입력할 수 있다.

2. 설정 블록

아이콘 및 템플릿 컬러를 지정할 수 있다.

3. 질문 블록

질문 텍스트 및 질문 유형을 지정할 수 있다.

4. 옵션 추가 버튼

객관식의 경우, 옵션을 추가할 수 있다.

5. 질문 추가 버튼

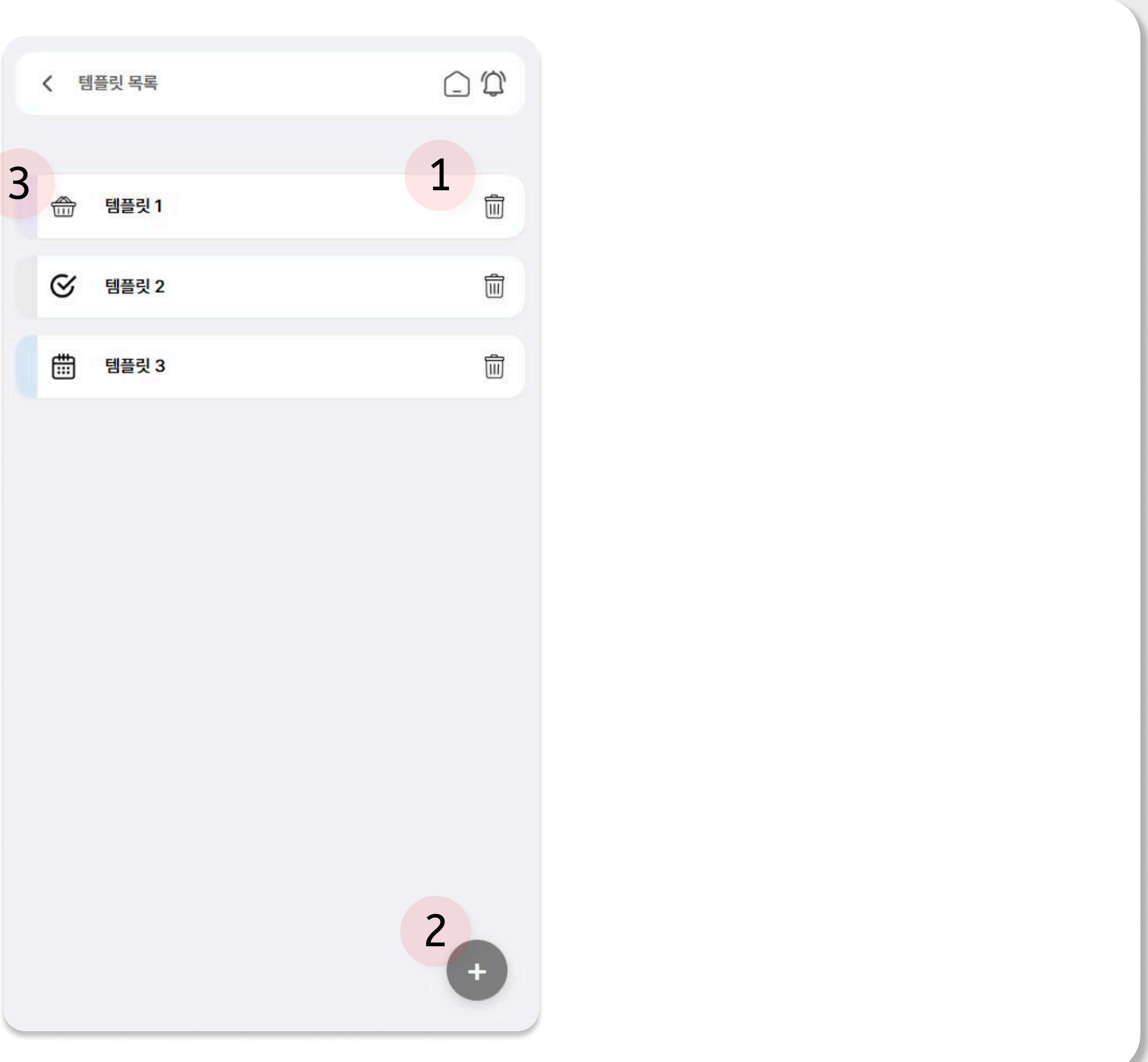
질문 블록을 추가할 수 있다.

6. 템플릿 생성 버튼

입력된 내용을 바탕으로 템플릿을 생성할 수 있다.

템플릿 리스트 페이지

담당자 : 손수용



1. 템플릿 삭제 버튼
템플릿을 삭제 할 수 있다.

2. 템플릿 생성 버튼
템플릿을 생성 할 수 있다.

3. 템플릿 디테일 이동 버튼
템플릿 디테일 페이지로 이동할 수 있다.

템플릿 디테일 페이지

담당자 : 손수용

The screenshot displays a template detail page with the following sections:

- Template 1:** A large input field.
- Question 1:** A dropdown menu with two options: "옵션 1" and "옵션 2".
- Question 2:** A dropdown menu with two options: "옵션 1" and "옵션 2".
- Question 3:** A text input field containing three dashes: "---".
- Question 4:** A date input field with the placeholder "연도-월-일".
- Question 5:** An image upload input field with a camera icon.
- Question 6:** A file upload input field with the placeholder "파일 선택" and the message "선택된 파일 없음".

1. 템플릿 제목

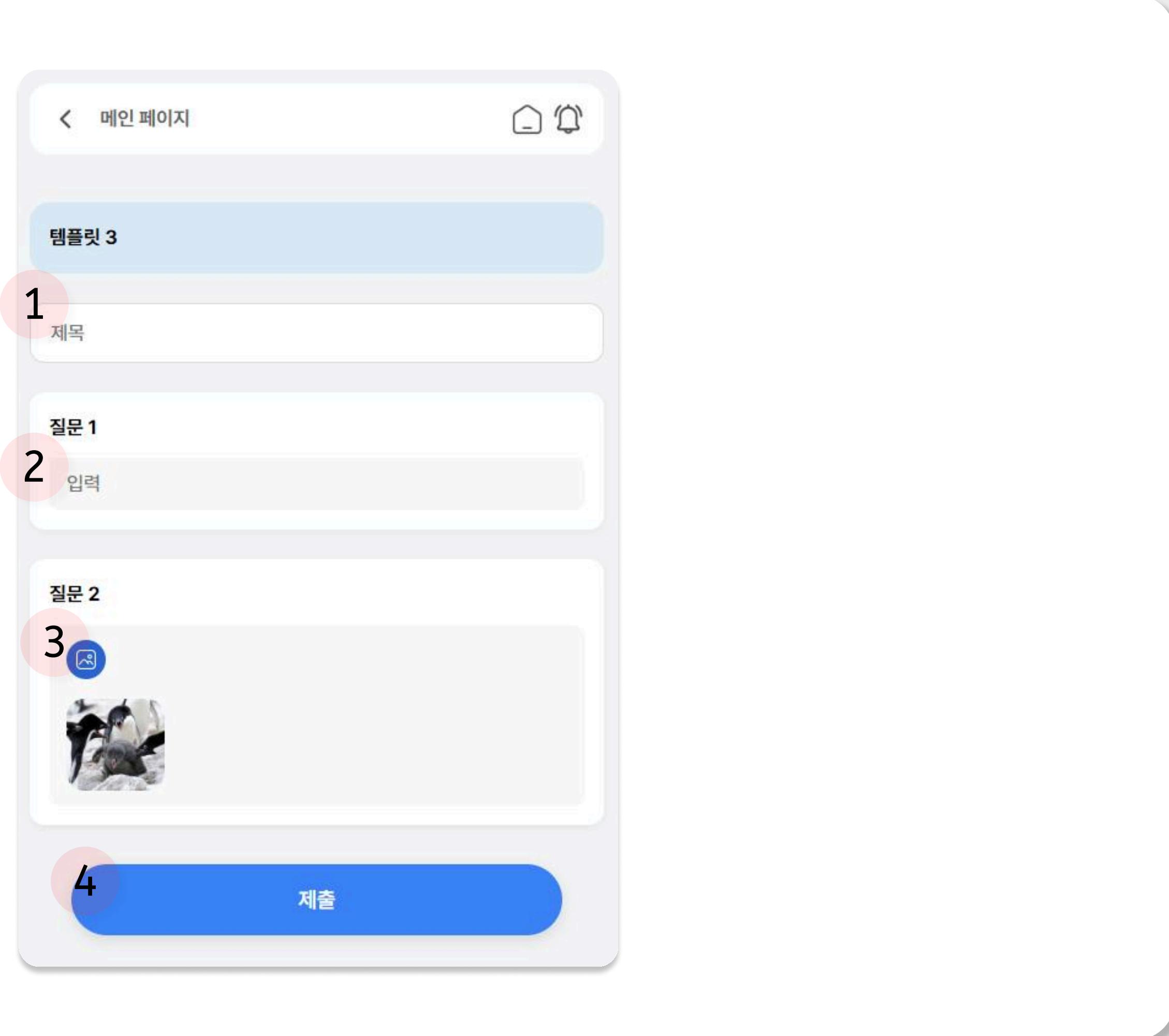
템플릿의 제목을 조회할 수 있다.

2. 템플릿 질문 리스트

템플릿의 질문 리스트를 조회할 수 있다.

보고서 작성 페이지

담당자 : 손수용



1. 보고서 제목 입력칸
보고서 제목을 입력 할 수 있다.

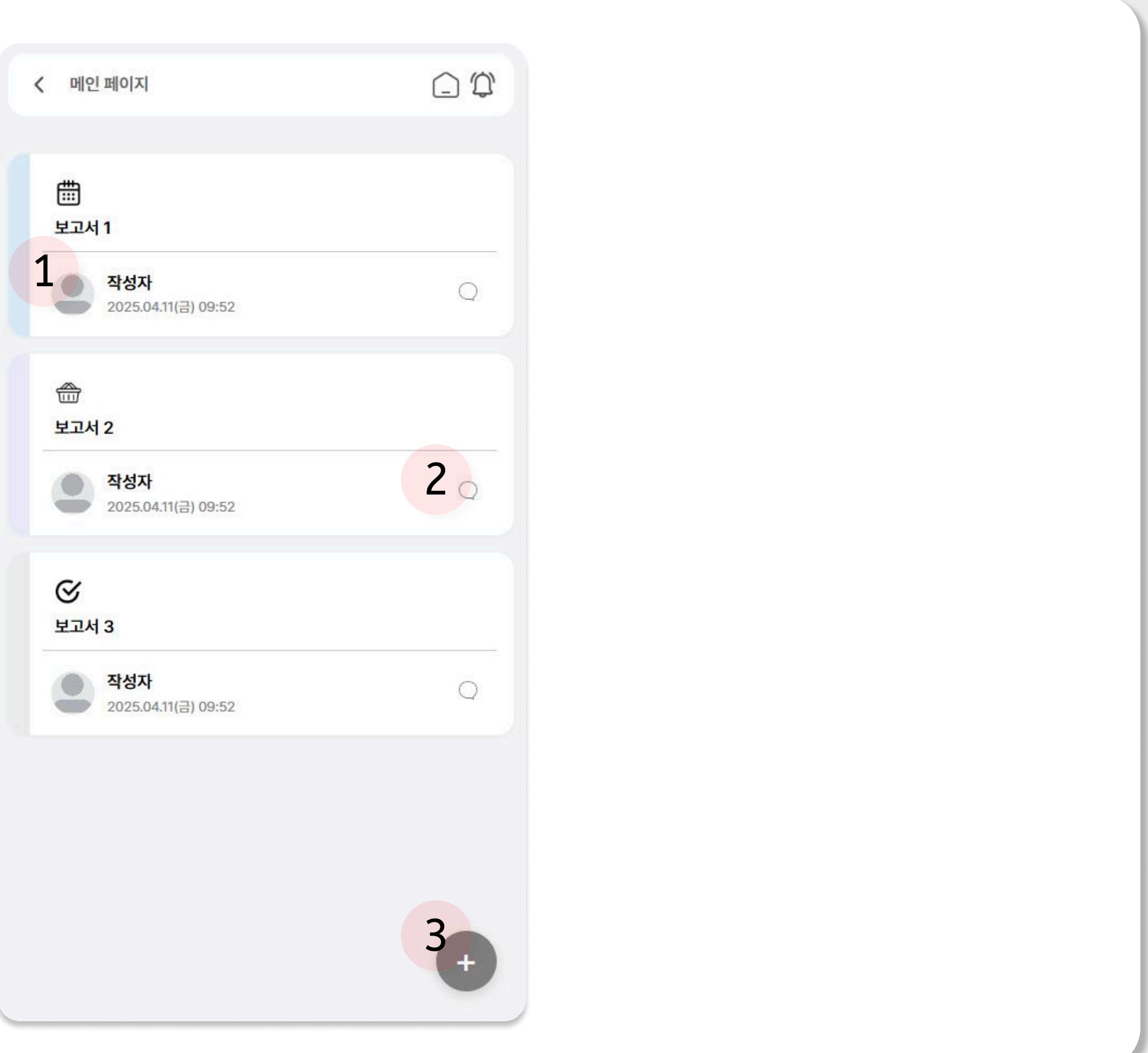
2. 답변 입력칸
질문의 답변을 입력 할 수 있다.

3. 사진 첨부 버튼
사진 유형의 질문일 경우, 사진을 첨부 할 수 있
으며 첨부할 경우 첨부한 사진을 미리볼 수 있다.

4. 제출 버튼
입력된 내용을 제출 할 수 있다.

보고서 리스트 페이지

담당자 : 손수용



1. 작성자 정보 필드

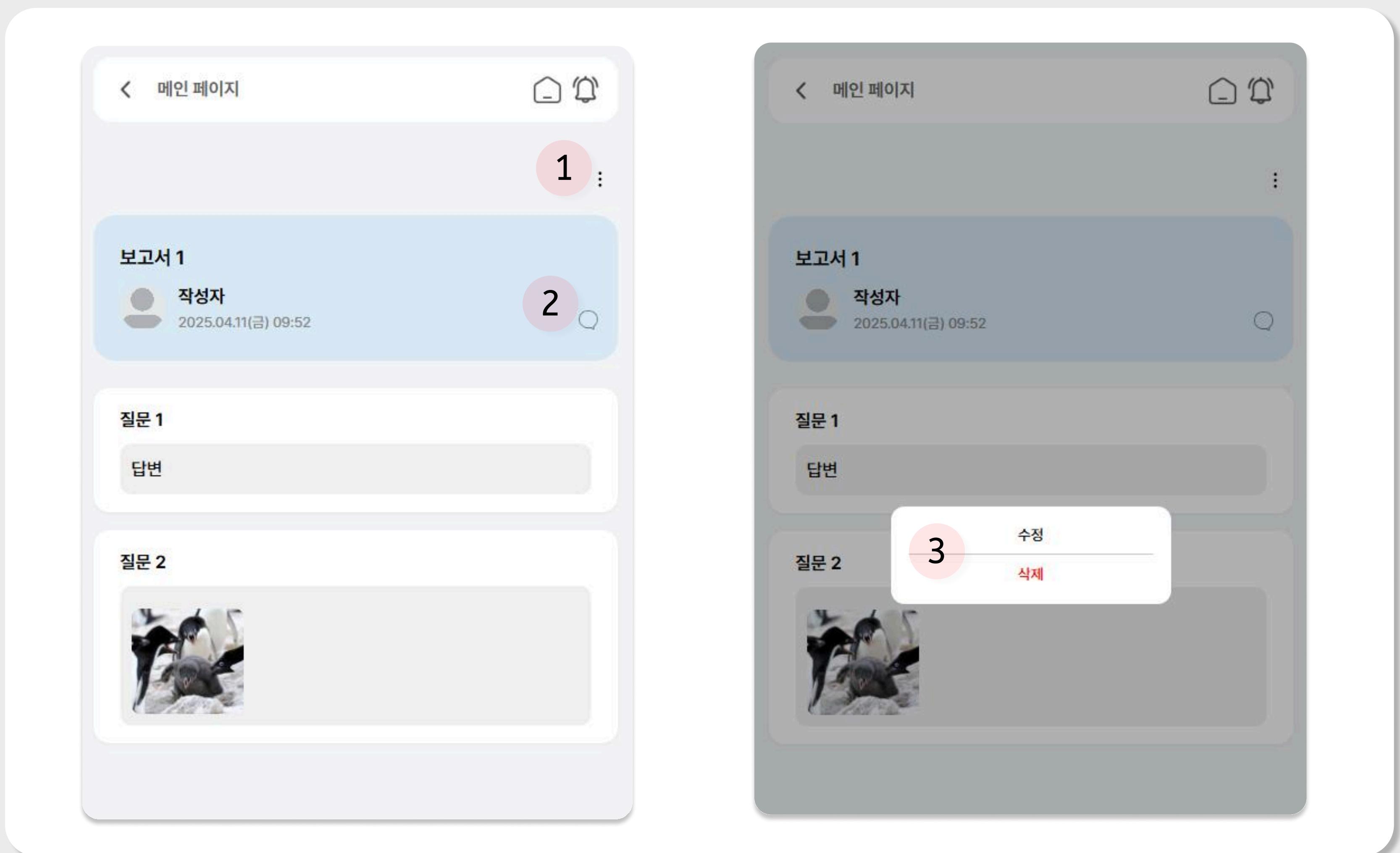
2. 댓글 버튼

해당 보고서의 댓글 페이지로 이동할 수 있다.

3. 보고서 작성 버튼

보고서 디테일 페이지

담당자 : 손수용



1. 자세히 버튼

클릭 시 수정/삭제 버튼이 나타난다.

2. 댓글 버튼

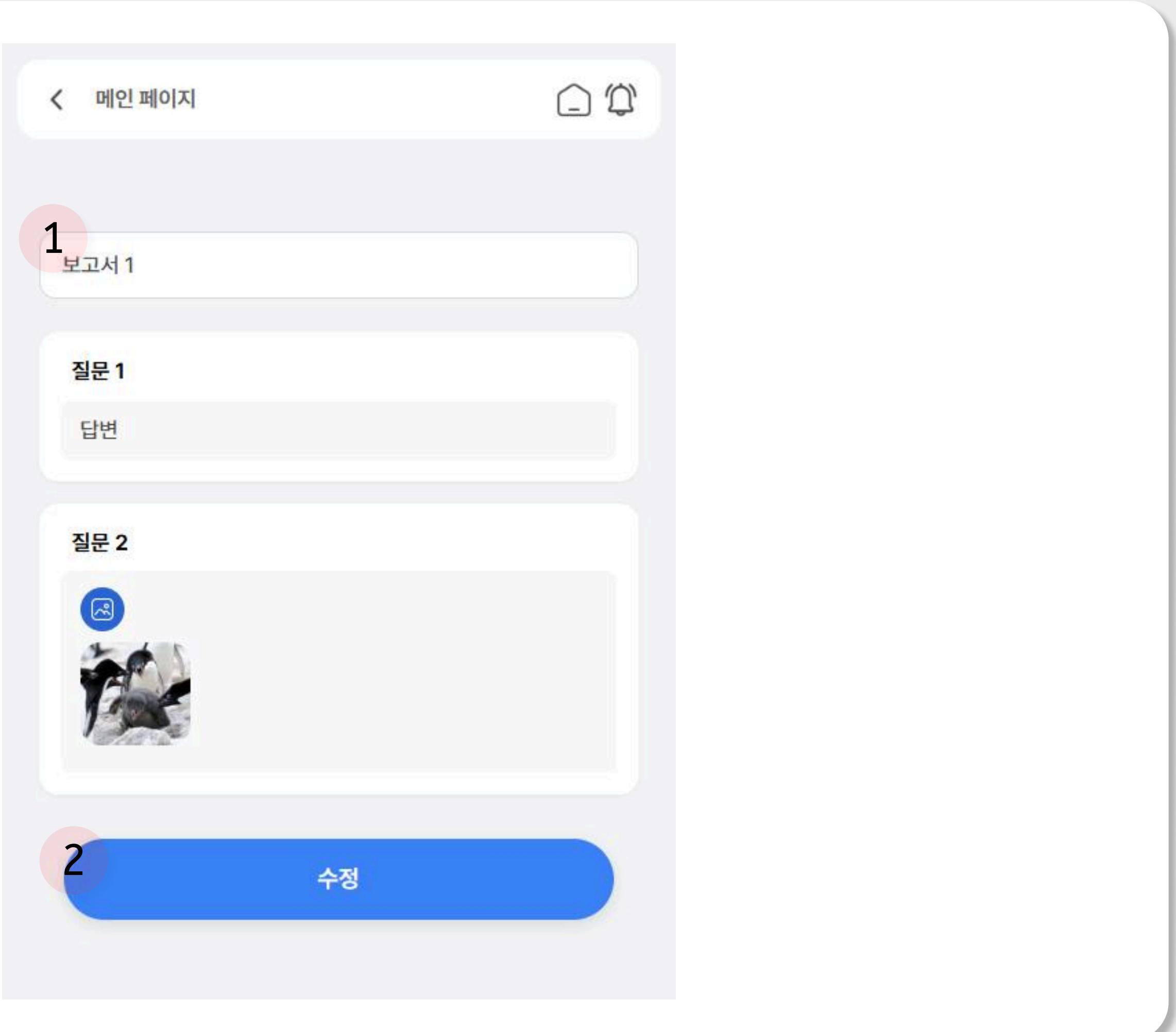
3. 수정 / 삭제 버튼

보고서 수정 페이지

담당자 : 손수용

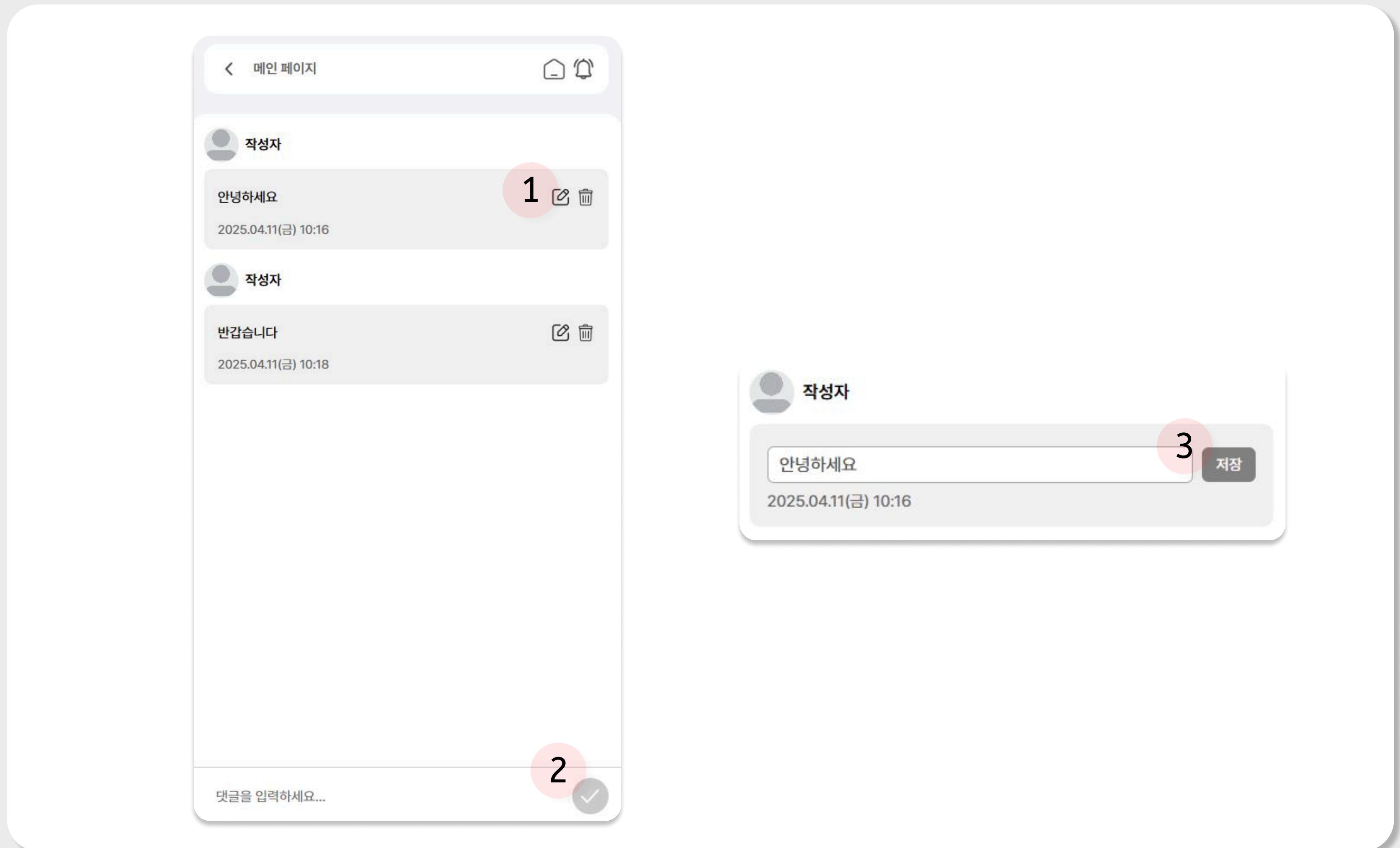
1. 보고서 수정 필드

2. 보고서 수정 버튼



보고서 수정 페이지

담당자 : 손수용



1. 댓글 수정 / 삭제 버튼

댓글 수정 버튼을 누를 시 입력창이 나타난다.

2. 댓글 작성 버튼

3. 댓글 수정 완료 버튼



담당자 : 오예준

03. 프로젝트 시연

3-1. 화면 설계 - 로그인 및 회원 관리

3-2. 화면 설계 - 출퇴근

3-3. 화면 설계 - 스케줄

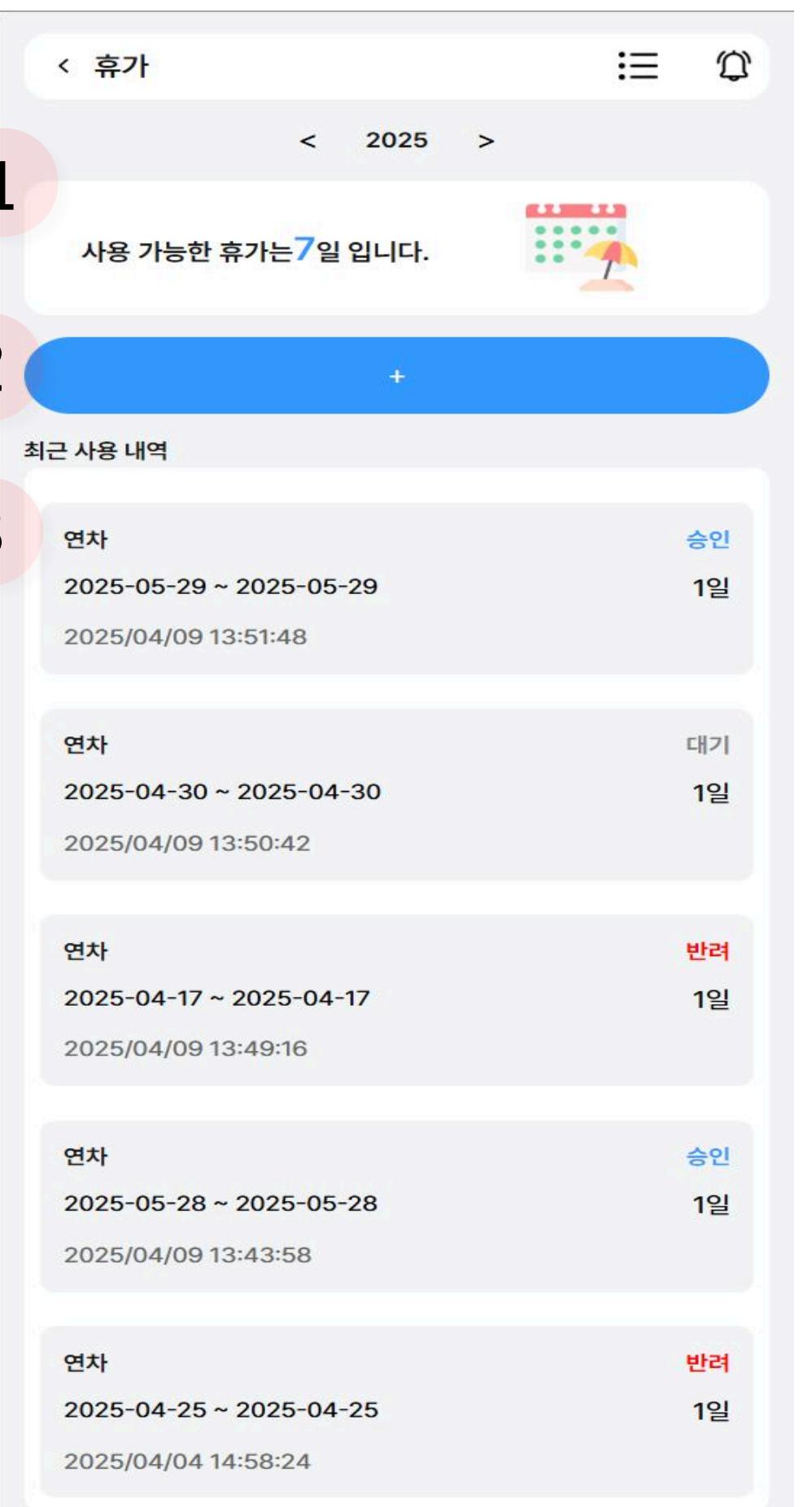
3-4. 화면 설계 - 경비관리

3-5. 화면 설계 - 템플릿 및 보고서

3-6. 화면 설계 - 휴가 및 알림

화면 제목

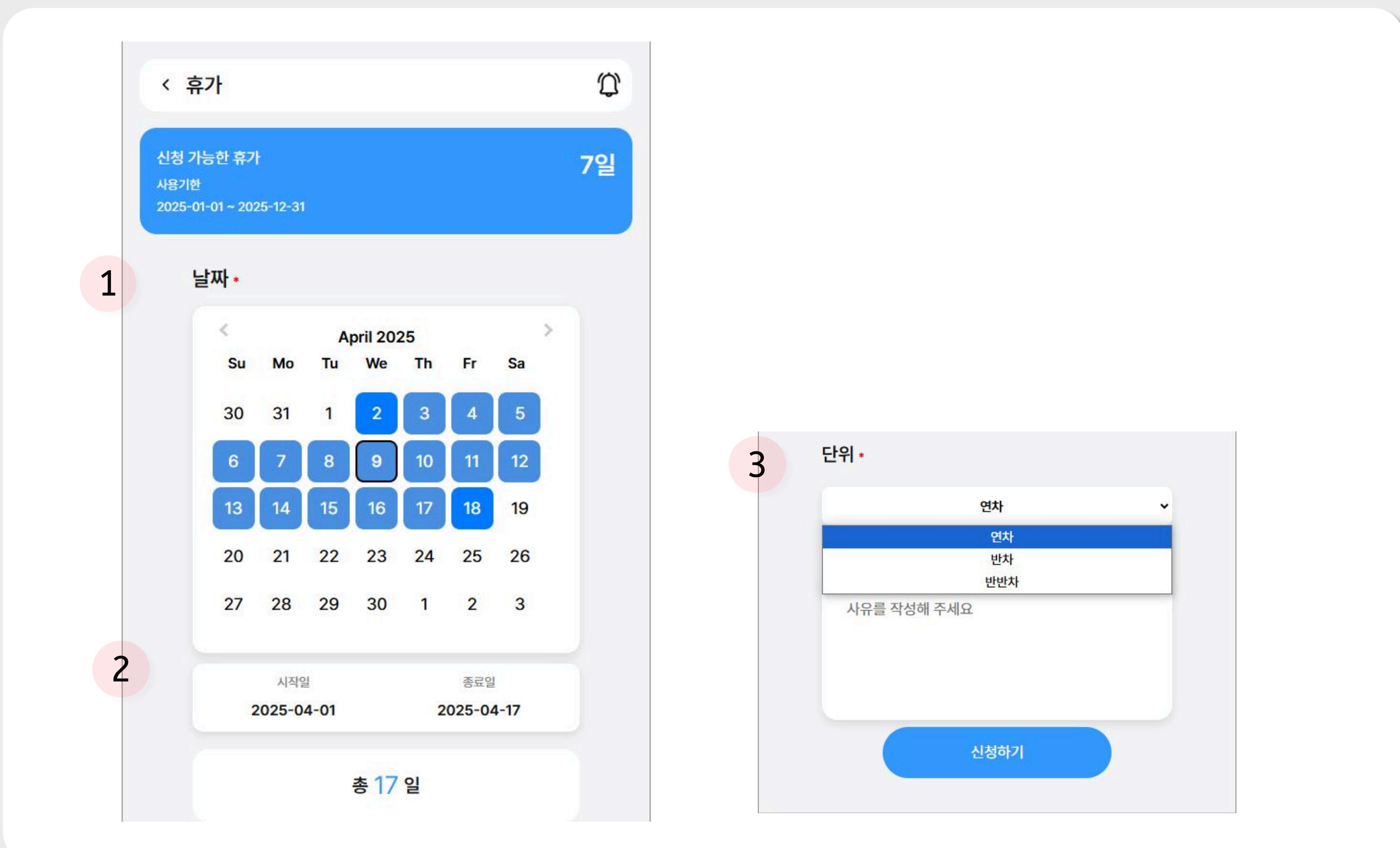
담당자 : 오예준



1. 휴가 페이지 사용 가능 연차 확인 가능
2. 휴가 신청 페이지로 이동하는 버튼
3. 휴가 최근 사용 내역 리스트를 보여줌 5개

화면 제목

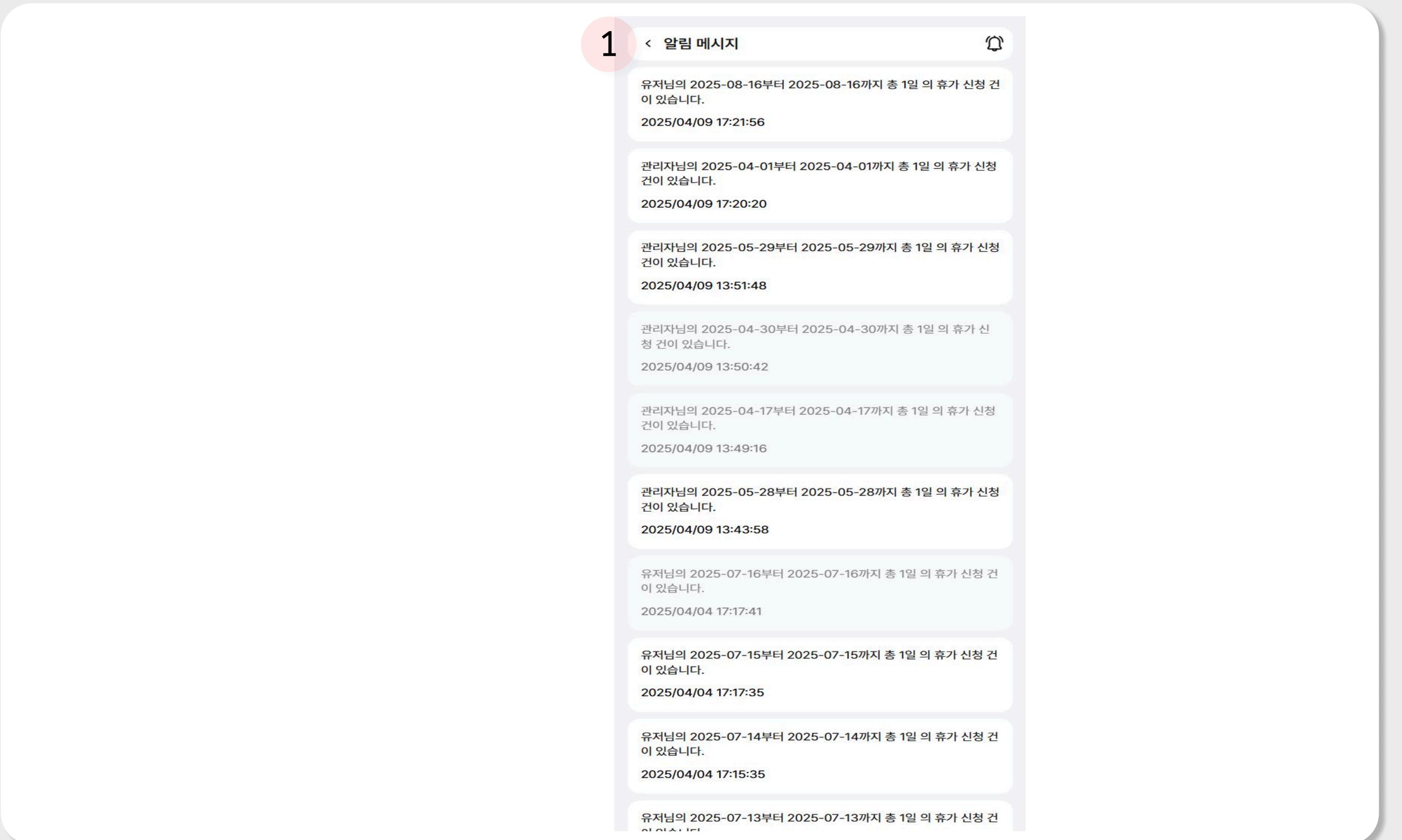
담당자 : 오예준



1. 휴가 신청 날짜 첫번째 날짜 선택 시 시작일로 자동 변경되며 두번째 날짜 선택시 종료일자로 자동으로 변경 휴가 기간에 대한 날짜를 백그라운드 변경을 통해 표시
2. 자동 변경된 시작일과 종료일을 통해 휴가 신청 총 일수가 계산되어 하단에 표시 됨
3. 단위 선택 연차, 반차, 반반차를 선택할 수 있고 사유란을 작성하는 입력폼이 있음. 이후 신청하기 버튼을 통해 신청 요청

화면 제목

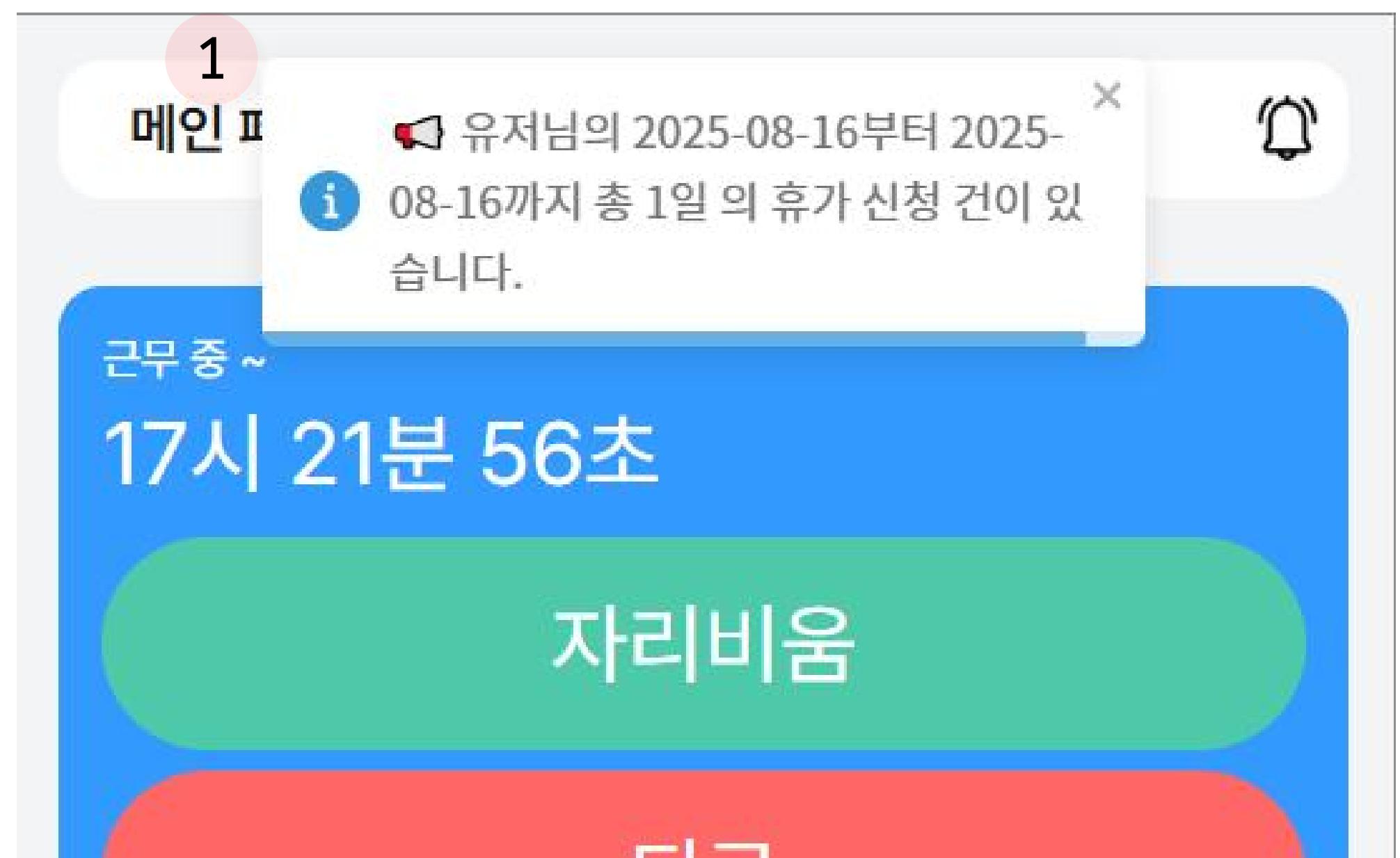
담당자 : 오예준



1. 관리자는 휴가 신청에 대한 알림 메시지를 받을 수 있고 직원들은 리더의 휴가 승인 및 반려 정보에 대한 알림 메시지를 실시간으로 받을 수 있다.

화면 제목

담당자 : 오예준



1. 실시간 알림 메시지는 페이지 상단 중앙부에 SSE를 통해 비동기적으로 처리되며 프론트에서 수신시 react-toastify를 통한 알림 메시지를 제공한다.

04. 코드 리뷰

4-1. 코드

04. 코드 리뷰

4-1. 코드

MemberService.java

담당자 : 최수민

```

public class MemberRegisterDto {
    private String name;
    private String email;
    private String password;
    private String phone;

    @Builder no usages sumin
    public MemberRegisterDto(String na
        this.name = name;
        this.email = email;
        this.password = password;
        this.phone = phone;
    }
}

// 회원가입
public MemberResponseDto register(MemberRegisterDto memberRegisterDto) { sumin
    String encodedPassword = passwordEncoder.encode(memberRegisterDto.getPassword());
    memberRegisterDto.setPassword(encodedPassword);
    isExistUserEmail(memberRegisterDto.getEmail());

    Member saveMember = MemberRegisterDto.ofEntity(memberRegisterDto);
    if (saveMember.getWorkStatus() == null) {
        saveMember.setWorkStatus(WorkStatus.퇴근);
    }
    if (saveMember.getSignUpStatus() == null) {
        saveMember.setSignUpStatus(SignUpStatus.PENDING);
    }
    saveMember = memberRepository.save(saveMember);
    return MemberResponseDto.fromEntity(saveMember, profileImageUrl: null);
}

private void isExistUserEmail(String email) { 1 usage sumin
    if (memberRepository.findByEmail(email).isPresent()) {
        throw new IllegalArgumentException("이미 사용 중인 이메일입니다.");
    }
}

```

1. 프론트에서 입력 받은 데이터를 받고 비밀번호는 암호화, 암호화 된 비밀번호를 **DTO**에 저장

2. **DTO**를 **DB**에 저장 가능한 **Member** 엔티티로 변환한다. **SignUpStatus**(가입 상태) 가 **null** 이면 **PENDING** 으로 기본값을 지정하고 가입 대기중 상태로 저장한다. 그 후 **saveMember** 객체를 **DB**에 전달, 프론트에서 받은 데이터를 저장

3. **MemberRepository** 에서 주어진 **email** 로 회원을 조회해서 이미 존재하는 **email** 이면 예외를 발생시켜 중복된 **email**로 회원가입 을 할 수 없게 한다,

MemberService.java

담당자 : 최수민

```
// 로그인
public MemberTokenDto login (MemberLoginDto memberLoginDto) { sumin +1
    Member member = memberRepository.findByEmail(memberLoginDto.getEmail()).orElseThrow(
        () -> new ResourceNotFoundException("Member", "Member Email", memberLoginDto.getEmail()));
    authenticate(memberLoginDto.getEmail(), memberLoginDto.getPassword());
    if (member.getSignUpStatus() == SignUpStatus.PENDING) {
        throw new UnauthorizedAccessException("가입 승인 대기중입니다.");
    }
    if (!member.isEnabled()) {
        throw new DisabledException("비활성화 된 계정입니다.");
    }
    String password = memberLoginDto.getPassword();
    if (passwordEncoder.matches(password, member.getPassword())) {
        String token = jwtTokenUtil.generateToken(member);
        return MemberTokenDto.fromEntity(member, token);
    } else {
        throw new IllegalArgumentException("잘못된 비밀번호입니다.");
    }
}

private void authenticate(String email, String password) { 1 usage sumin
    try {
        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(email,
    } catch (DisabledException e) {
        throw new MemberException("인증되지 않은 아이디입니다.", HttpStatus.BAD_REQUEST);
    } catch (BadCredentialsException e) {
        throw new MemberException("비밀번호가 일치하지 않습니다.", HttpStatus.BAD_REQUEST);
    }
}
```

```
public class MemberLoginDto
{
    private String email;
    private String password;
}

@Builder no usages sumin
public MemberLoginDto(St
{
    this.email = email;
    this.password = pass
}
```

1

2

3

1. 입력받은 이메일로 회원을 조회한다.
회원이 없으면 **ResourceNotFoundException** 예외를 발생시킨다.
이메일과 비밀번호를 기반으로 인증을 수행
2. 로그인 중 발생할 수 있는 예외를 설정한다.
 - **SignUpStatus** 가 **ACTIVE** 가 아닐 때 (승인X)
 - 탈퇴한 계정일 때

회원 가입시 입력한 비밀번호를 가져와서 DB에 저장된 암호화된 비밀번호와 비교한다.

로그인 성공시 **JWT** 토큰을 생성하고 회원을 증명하기 위해 클라이언트에 반환하고 생성된 토큰과 회원 정보를 **MemberTokenDto**로 감싸서 반환한다.

3. **email** 과 비밀번호를 받아 인증을 실행하는 메소드 **authenticationManager** 를 통해 인증을 하고 이메일과 비밀번호를 토큰에 담아 인증을 요청한다.

인증 과정에서 비활성화 처리 된 계정 혹은 인증 정보가 잘못된 경 예외가 발생한다.

MemberService.java

담당자 : 최수민

1. memberRepository에서 가입 상태가 PENDING인 회원을 DB에서 조회한다.

가입 승인 대기 중인 회원이 없다면
return new ArrayList<>(); 빈 배열을 반환

```
// 가입 신청자 조회
public List<MemberPendingDto> getPendingMembers() { sumin
    List<Member> members = memberRepository.findBySignUpStatus(SignUpStatus.PENDING);

    if (members.isEmpty()) {
        return new ArrayList<>();
    }

    return members.stream() Stream<Member>
        .map(MemberPendingDto::fromEntity) Stream<MemberPendingDto>
        .collect(Collectors.toList());
}
```

2. 특정 이메일을 가진 회원의 signUpStatus를 ACTIVE(활성화)로 변경하는 메소드

memberRepository에서 해당 email을 가진 회원을 조회한다.

조회한 signUpStatus를 ACTIVE로 변경하고
변경된 데이터를 다시 DB에 저장한다.

```
public void updateSignUpStatusToActive(String email) { sumin
    Member member = memberRepository.findByEmail(email)
        .orElseThrow(() -> new ResourceNotFoundException("회원", "이메일", email));
    member.setSignUpStatus(SignUpStatus.ACTIVE);
    memberRepository.save(member);
}
```

AttendanceService.java

담당자 : 이영현

1

```
private static final double EARTH_RADIUS = 6371.0088; // 지구 평균 반지름(km) 1 usage
private static final int MAX_DISTANCE = 50; // 허용할 최대 거리(m) 1 usage
private static final double COMPANY_LATITUDE = 37.482175; // 회사 위도 1 usage
private static final double COMPANY_LONGITUDE = 126.898233; // 회사 경도 1 usage
private static final LocalTime COMPANY_START_TIME = LocalTime.of(hour: 9, minute: 0); 2 usages
private static final LocalTime COMPANY_END_TIME = LocalTime.of(hour: 18, minute: 0); 2 usages
```

2

```
@Transactional
public void clockIn(Member member, ReqClockInDto dto, LocalDateTime now) {
    if (!isInDistance(dto.getLatitude(), dto.getLongitude())) {
        throw new IllegalArgumentException(" - " + "허용 범위를 벗어났습니다.");
    }
    Member currentMember = memberRepository.findByEmail(member.getEmail()).orElseThrow(
        () -> new MemberException("확인된 사용자가 아닙니다.", HttpStatus.BAD_REQUEST)
    );
    int lateMinutes = late(now);
    EventRecord eventRecord = ReqClockInDto.toEntity(currentMember, lateMinutes);
    eventRecordRepository.save(eventRecord);
    currentMember.updateWorkStatus(WorkStatus.근무중);
}

@Transactional 1 usage
public void outOfOffice(Member member, ReqOutOfOfficeDto dto, LocalDateTime now) {...}

@Transactional 1 usage
public void returnToOffice(Member member, ReqReturnToOfficeDto dto, LocalDateTime now) {...}

@Transactional 1 usage
public void clockOut(Member member, LocalDateTime now) {...}
```

1. 상수 선언부

- **Haversine** 공식을 위한 지구의 반지름. 정확도를 높이기 위해 대한민국(북위 33~38도)의 평균 반지름을 소수 4번째 자리까지 구하여 대입했다.
- 오차범위를 위한 최대 허용 거리.
- 회사의 위도와 경도 좌표.
- 근태 관리를 위한 회사 영업 시간.

2. 출근 / 자리비움 / 복귀 / 퇴근 메소드

- 출근: 요청된 현재 위치와 회사의 거리를 **isInDistance** 메소드로 검증, **late** 메소드로 지각 여부를 검증한 뒤 근태 기록을 저장하고, 직원의 **workStatus**를 ‘근무중’으로 변경한다.
- 자리비움: 자리비움 유형을 선택해 요청하면 근태 기록을 저장하고, 직원의 **workStatus**를 ‘자리비움중’으로 변경한다.
- 복귀: 요청된 현재 위치와 지정된 회사의 거리를 **isInDistance** 메소드로 검증, 근태 기록을 저장하고, 직원의 **workStatus**를 ‘근무중’으로 변경한다.
- 퇴근: **leaveEarly** 메소드로 조퇴 여부를 검증한 뒤 근태 기록을 저장하고, 직원의 **workStatus**를 ‘퇴근’으로 변경한다.

AttendanceService.java

담당자 : 이영현

```

private int late(LocalDateTime clockInTime) { 1 usage
    LocalTime Time = clockInTime.toLocalTime();
    if (Time.isAfter(COMPANY_START_TIME)) {
        Duration duration = Duration.between(COMPANY_START_TIME, Time);
        return (int) duration.toMinutes();
    }
    return 0;
}

private int leaveEarly(LocalDateTime clockOutTime) { 4 usages
    LocalTime Time = clockOutTime.toLocalTime();
    if (Time.isBefore(COMPANY_END_TIME)) {
        Duration duration = Duration.between(Time, COMPANY_END_TIME);
        return (int) duration.toMinutes();
    }
    return 0;
}

private boolean isInDistance(double latitude, double longitude) { 2 usages
    double distance = calculateDistance(COMPANY_LATITUDE, COMPANY_LONGITUDE, latitude, longitude);
    return distance <= MAX_DISTANCE;
}

// 하버사인 공식 - 구면 위 두 개의 좌표(위도, 경도) 사이의 거리를 구하는 메소드
private double calculateDistance(double lat1, double lon1, double lat2, double lon2) { 1 usage
    // 위도와 경도 차이를 라디안으로 변환
    double latDistance = Math.toRadians(lat2 - lat1);
    double lonDistance = Math.toRadians(lon2 - lon1);
    // 하버사인 공식 적용
    double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
            + Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
            * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    // 중심각 계산
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    // 최종 거리 계산
    double distance = EARTH_RADIUS * c * 1000; // 미터로 변환
    log.info("거리 : " + String.format("%.3f", distance) + "미터.");
    return distance;
}

```

1. late 메소드

clockIn 메소드에서 사용되며, 상수로 선언된 지정 출근 시간보다 늦게 출근하는지 검증하여 지각이 아니라면 0, 지각이라면 지각한 분 수를 리턴한다.

2. leaveEarly 메소드

clockOut 메소드에서 사용되며, 상수로 선언된 지정 퇴근 시간보다 일찍 퇴근하는지 검증하여 조퇴가 아니라면 0, 조퇴라면 조퇴한 분 수를 리턴한다.

3. isInDistance 메소드

clockIn, returnToOffice 메소드에서 사용되며, **Haversine** 공식으로 계산된 거리가 최대 허용 거리(50m)보다 작은지 검증하여 이내라면 **true**, 벗어났다면 **false**를 리턴한다.

4. calculateDistance 메소드

isInDistance 메소드에서 사용되며, 지정된 회사의 위도, 경도와 요청된 위도, 경도를 **Haversine** 공식에 대입해 거리의 차이를 리턴한다.

AttendanceManagerForEmployeeService.java

담당자 : 이영현

```

1   public ResAttendanceRecordDto getDailyAttendanceRecord(Member member, LocalDate date) { 1 usage
2       List<EventRecord> records = eventRecordRepository.findByMemberAndDate(member, date);
3       return ResAttendanceRecordDto.fromEntity(records, date);
4   }
5
6   public ResAttendanceHistoryForEmployeeDto getWeeklyAttendanceHistory(Member member, LocalDate date) { 1 usage
7       LocalDate startOfWeek = date.with(java.time.DayOfWeek.MONDAY);
8       LocalDate endOfWeek = startOfWeek.plusDays( daysToAdd: 6 );
9       List<EventRecord> records = eventRecordRepository.findByMemberAndDateBetween(member, startOfWeek, endOfWeek);
10
11      List<LocalDate> weekdays = startOfWeek.datesUntil(endOfWeek.plusDays( daysToAdd: 1 ))
12          .filter(d -> !d.getDayOfWeek().equals(java.time.DayOfWeek.SATURDAY) &&
13              !d.getDayOfWeek().equals(java.time.DayOfWeek.SUNDAY))
14          .toList();
15
16      LocalDate today = LocalDate.now();
17      return ResAttendanceHistoryForEmployeeDto.fromEntity(records, weekdays, today);
18  }
19
20  public ResAttendanceHistoryForEmployeeDto getMonthlyAttendanceHistory(Member member, int year, int month) { 1 usage
21      LocalDate startOfMonth = LocalDate.of(year, month, dayOfMonth: 1);
22      LocalDate endOfMonth = startOfMonth.plusMonths( monthsToAdd: 1 ).minusDays( daysToSubtract: 1 );
23
24      List<EventRecord> records = eventRecordRepository.findByMemberAndDateBetween(member, startOfMonth, endOfMonth);
25
26      List<LocalDate> weekdaysInMonth = startOfMonth.datesUntil(endOfMonth.plusDays( daysToAdd: 1 ))
27          .filter(date -> !date.getDayOfWeek().equals(java.time.DayOfWeek.SATURDAY) &&
28              !date.getDayOfWeek().equals(java.time.DayOfWeek.SUNDAY))
29          .toList();
30
31      LocalDate today = LocalDate.now();
32
33      return ResAttendanceHistoryForEmployeeDto.fromEntity(records, weekdaysInMonth, today);
34  }

```

1. **getDailyAttendanceRecord** 메소드

요청한 날짜와 해당 직원의 근태 기록을 **List**로 반환받은 뒤, 응답 **dto**에서 데이터를 구조화 하도록 반환한다.

2. **getWeeklyAttendanceHistory** 메소드

요청한 날짜가 포함된 주를 계산해 **7일간**의 해당 직원의 근태 기록과 평일 목록을 구해서 응답 **dto**에서 데이터를 구조화 하도록 반환한다.

3. **getMonthlyAttendanceHistory** 메소드

요청한 연도와 월을 받아 한달을 계산해 **한달간**의 해당 직원의 근태 기록과 평일 목록을 구해서 응답 **dto**에서 데이터를 구조화 하도록 반환한다.

AttendanceManagerForLeaderService.java

담당자 : 이영현

```

public ResAttendanceHistoryForLeaderDto getDailyAttendanceForAll(LocalDate date) { 1 usage
    List<EventRecord> records = eventRecordRepository.findByDate(date);
    int absentPersonnelCount = 0; // 미출근 카운트
    int notYetArrivedCount = 0; // 출근 전 카운트
    List<Member> allMembers = memberRepository.findAll();

    for (Member member : allMembers) {
        int status = updateAttendanceStatus(member, date);
        if (status == -1) {
            notYetArrivedCount++;
        } else if (status == 1) {
            absentPersonnelCount++;
        }
    }

    int workdayPersonnel = allMembers.size() - absentPersonnelCount; // 전체 직원 수에서 미출근 인원 제외
    return ResAttendanceHistoryForLeaderDto.fromEntity(records, workdayPersonnel, absentPersonnelCount, date);
}

public ResAttendanceHistoryForLeaderDto getDailyAttendanceForTeam(LocalDate date, Team team) { 1 usage
    List<EventRecord> records = eventRecordRepository.findByDateAndTeam(date, team);
    int absentPersonnelCount = 0; // 미출근 카운트
    int notYetArrivedCount = 0; // 출근 전 카운트
    List<Member> teamMembers = memberRepository.findByTeam(team);

    for (Member member : teamMembers) {
        int status = updateAttendanceStatus(member, date);
        if (status == -1) {
            notYetArrivedCount++;
        } else if (status == 1) {
            absentPersonnelCount++;
        }
    }

    int workdayPersonnel = teamMembers.size() - absentPersonnelCount; // 팀 멤버 수에서 미출근 인원 제외
    return ResAttendanceHistoryForLeaderDto.fromEntity(records, workdayPersonnel, absentPersonnelCount, date);
}

```

1

2

1. getDailyAttendanceForAll 메소드

특정 날짜에 대해 전체 직원의 근태 기록을 조회하고, 미출근 인원 및 출근 전 인원을 계산하여 dto에서 데이터를 구조화 하도록 반환한다.

2. getDailyAttendanceForTeam 메소드

특정 날짜와 팀에 대해 해당 팀의 근태 기록을 조회하고, 미출근 인원 및 출근 전 인원을 계산하여 dto에서 데이터를 구조화 하도록 반환한다.

- (리더) 출퇴근 현황 카운팅 규칙
 - 해당 날짜의 휴가자는 ‘미출근’으로 계산한다.
 - 이전 날짜를 조회할 경우, 출근 기록이 없는 직원은 ‘미출근’으로 계산한다.
 - 오늘을 조회하고, 09시 이전에 조회할 경우, (휴가자를 제외한) 아직 출근 기록이 없는 직원은 ‘출근 전’으로 계산한다.
 - 오늘을 조회하고, 09시 이후에 조회할 경우, (휴가자를 제외한) 아직 출근기록이 없는 직원은 ‘미출근’으로 계산한다.

AttendanceManagerForLeaderService.java

담당자 : 이영현

1

```
public long getTotalEmployeeCount() { 1 usage
    return memberRepository.count();
}
```

2

```
public long getEmployeeCountByTeam(Team team) { 1 usage
    return memberRepository.countByTeam(team);
}
```

3

```
private boolean isOnVacation(Member member, LocalDate date) { 1 usage
    return vacationRepository.existsVacationOverlap(date, date, member);
}
```

4

```
private int updateAttendanceStatus(Member member, LocalDate date) { 2 usages
    boolean hasAttendanceRecord = eventRecordRepository.existsByMemberAndDate(member, date);

    if (isOnVacation(member, date)) {
        return 1;
    } else if (!hasAttendanceRecord) {
        if (date.isEqual(LocalDate.now())) { // 오늘 날짜인 경우
            if (LocalTime.now().isBefore(COMPANY_START_TIME)) { // 회사 시작 시간 이전
                return -1; // 출근 전 상태
            } else {
                return 1; // 미출근
            }
        } else { // 지난 날짜인 경우
            return 1; // 지난 날짜는 모두 미출근으로 처리
        }
    }
    return 0; // 출근 기록이 있는 경우
}
```

1. getTotalEmployeeCount 메소드

전체 직원의 수를 리턴한다.

2. getEmployeeCountByTeam 메소드

팀별 직원의 수를 리턴한다.

3. isOnVacation

요청한 날짜에 해당 직원의 휴가 여부를 리턴한다.

4. updateAttendanceStatus

요청한 날짜에

- 해당 직원이 휴가중이라면 **1** 리턴.
- 근태 기록이 없고, 요청한 날짜가 지난 날짜인 경우 **1** 리턴.
- 근태 기록이 없고, 요청한 날짜가 오늘이며, 요청한 시간이 회사 시작 시간 이전인 경우 **-1** 리턴.
- 근태 기록이 없고, 요청한 날짜가 오늘이며, 요청한 시간이 회사 시작 시간 이후인 경우 **1** 리턴.

AttendanceBanner.js

```

AttendanceBanner.js

1
const handleClockIn = async () => {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      async (position) => {
        const { latitude, longitude } = position.coords;

        try {
          const response = await axios.post(
            "http://localhost:8080/clockIn",
            { latitude, longitude },
            {
              headers: {
                Authorization: `Bearer ${token}`,
                "Content-Type": "application/json",
              },
            }
          );
          alert(response.data);
          setStartTime(currentDate.time);
          setStatus("근무중");
        } catch (error) {
          console.error(`출근 실패:`, error.response?.data || error.message);
          alert(`출근 실패: ${JSON.stringify(error.response?.data || error.message)}`);
        }
      },
    );
  }
}

2
(error) => {
  alert("위치 정보 로드 실패.");
  console.error(error);
}

3
);
} else {
  alert("Geolocation 로드 실패.");
}
};


```

담당자 : 이영현

- 1. Geolocation API 사용.**
- 2. 요청 헤더를 검사한 뒤 **HTTP POST**로 본문에 사용자의 위도와 경도를 담아 서버에 출근 요청, 실패 처리.**
- 3. 위치 정보 로드 실패 처리, **Geolocation API** 지원 실패 처리.**

AttendanceManagerForEmployee.js

담당자 : 이영현

1

```
AttendanceManagerForEn

const fetchAttendanceData = async (date) => {
    try {
        setLoading(true);
        const token = localStorage.getItem("CL_access_token");
        const formattedDate = date.toLocaleDateString('en-CA', { year: 'numeric', month: '2-digit', day: '2-digit' });
        console.log("요청한 날짜:", formattedDate);
        let response;

        if (viewType === "daily") {
            response = await axios.get("http://localhost:8080/daily", {
                headers: { Authorization: `Bearer ${token}` },
                params: { date: formattedDate }
            });
        } else if (viewType === "weekly") {
            response = await axios.get("http://localhost:8080/weekly", {
                headers: { Authorization: `Bearer ${token}` },
                params: { date: formattedDate },
            });
        } else if (viewType === "monthly") {
            const year = date.getFullYear();
            const month = date.getMonth() + 1;
            response = await axios.get("http://localhost:8080/monthly", {
                headers: { Authorization: `Bearer ${token}` },
                params: { year: year, month: month },
            });
        }
        console.log("불러온 데이터:", response.data);
        setAttendanceData(response.data);

    } catch (error) {
        console.error("데이터 로드 오류:", error);
    } finally {
        setLoading(false);
    }
};
```

2

- 로딩 상태 결정, 로컬 스토리지에서 토큰 가져오기, **DatePicker**로 요청할 날짜를 **YYYY-MM-DD** 형식으로 보장하기.
- 버튼 선택으로 달라지는 **viewType**에 따라 서로 다른 **API** 엔드포인트 호출, 데이터 설정.
- 오류 처리, 로딩상태 해제.

3

AttendanceManagerForLeader.js

담당자 : 이영현

1. 로딩 상태 결정, 로컬 스토리지에서 토큰 가져오기, **DatePicker**로 요청할 날짜를 **YYYY-MM-DD** 형식으로 보장하기.
2. 버튼 선택으로 달라지는 **viewType**에 따라 서로 다른 **API** 엔드포인트 호출, 데이터 설정.
3. 오류 처리, 로딩상태 해제.

```

AttendanceManagerForLe

const fetchAttendanceData = async (date) => {
    try {
        setLoading(true);
        const token = localStorage.getItem("CL_access_token");
        const formattedDate = date.toLocaleDateString('en-CA', { year: 'numeric', month: '2-digit', day: '2-digit' });
        console.log("요청한 날짜:", formattedDate);
        let response;

        if (viewType === "forAll") {
            response = await axios.get("http://localhost:8080/forAll", {
                headers: { Authorization: `Bearer ${token}` },
                params: { date: formattedDate }
            });
        } else if (viewType === "forTeam" && selectedTeam) {
            response = await axios.get("http://localhost:8080/forTeam", {
                headers: { Authorization: `Bearer ${token}` },
                params: { date: formattedDate, team: selectedTeam }
            });
        }

        if (response && response.data) {
            console.log("불러온 데이터:", response.data);
            setAttendanceData(response.data);
        } else {
            setAttendanceData(null);
        }
    } catch (error) {
        console.error("데이터 로드 오류:", error);
        setAttendanceData(null);
    } finally {
        setLoading(false);
    }
};

```

ScheduleController

담당자 : 김성우

```

1  @GetMapping("list")  ▲ seongwoo+
public ResponseEntity<List<ResScheduleDto>> getSchedules (@AuthenticationPrincipal Member member,
                                                               @RequestParam Integer year,
                                                               @RequestParam Integer month) {
    Long memberId = member.getId();
    List<ResScheduleDto> listDto = scheduleService.getScheduleByMemberIdAndYearAndMonth(memberId,year,month);
    return ResponseEntity.status(HttpStatus.OK).body(listDto);
}

2  @PostMapping("write")  ▲ seongwoo
public ResponseEntity<ResScheduleWriteDto> writeSchedule (@RequestBody ScheduleWriteDto dto) {
    ResScheduleWriteDto savedDto = scheduleService.write(dto);
    return ResponseEntity.status(HttpStatus.CREATED).body(savedDto);
}

3  @PatchMapping("update")  ▲ seongwoo
public ResponseEntity<ResScheduleDto> updateSchedule (@RequestBody ScheduleUpdateDto dto) {
    Long id = dto.getId();
    ResScheduleDto updateDto = scheduleService.update(id,dto);
    return ResponseEntity.status(HttpStatus.OK).body(updateDto);
}

4  @DeleteMapping("delete")  ▲ seongwoo
public ResponseEntity<String> deleteSchedule (@RequestParam Long id) {
    scheduleService.delete(id);
    return ResponseEntity.status(HttpStatus.OK).body("Schedule deleted");
}

```

1. 모든 스케줄 정보 조회
캘린더에 스케줄 정보를 가져옴
2. 로그인한 맴버, 연도, 월을 파라미터로 전달 받고 서비스 단에 요청
3. 스케줄 등록
스케줄 작성 버튼 클릭 시 요청
4. 작성한 스케줄 정보를 json 데이터로 받아 dto로 매핑. 해당 dto를 서비스단에 전달
5. 스케줄 정보 수정
스케줄 수정 버튼 클릭 시 요청
6. 수정한 스케줄 정보를 json 데이터로 받아 dto로 매핑. 해당 dto를 서비스 단에 전달
7. 스케줄 삭제
선택한 스케줄 id를 서비스단에 전달

ScheduleService / ScheduleRepository

담당자 : 김성우

```

1 public List<ResScheduleDto> getScheduleByMemberIdAndYearAndMonth(Long memberId, Integer year, Integer month) { 1 usage
    List<Schedule> schedules = scheduleRepository.findByMemberIdAndYearAndMonth(memberId, year, month);
    List<ResScheduleDto> ListDto = schedules.stream() Stream<Schedule>
        .map(ResScheduleDto::fromEntity) Stream<ResScheduleDto>
        .collect(Collectors.toList());
    return ListDto;
}

2 public ResScheduleWriteDto write(ScheduleWriteDto dto) { ✎ seongwoo
    Member member = memberRepository.findById(dto.getMemberId())
        .orElseThrow(() -> new IllegalArgumentException("Member not found"));
    Schedule schedule = ScheduleWriteDto.ofEntity(member, dto);
    Schedule savedSchedule = scheduleRepository.save(schedule);
    return ResScheduleWriteDto.fromEntity(savedSchedule);
}

3 public ResScheduleDto update(Long id, ScheduleUpdateDto dto) { ✎ seongwoo
    Schedule updateSchedule = scheduleRepository.findById(id).orElseThrow(
        () -> new IllegalArgumentException("Schedule not found")
    );
    updateSchedule.update(dto);
    return ResScheduleDto.fromEntity(updateSchedule);
}

4 public void delete(Long id) { scheduleRepository.deleteById(id); } ✎ seongwoo

5 public interface ScheduleRepository extends JpaRepository<Schedule, Long> { 2 usages ✎ seongwoo
    @Query("SELECT s FROM Schedule s WHERE s.member.id = :memberId AND FUNCTION('YEAR', s.date) = :year AND FUNCTION('MONTH', s.date) = :month")
    List<Schedule> findByMemberIdAndYearAndMonth(@Param("memberId") Long memberId, @Param("year") Integer year, @Param("month") Integer month);
}

```

1. 맴버 아이디, 연도, 월 정보로 데이터를 요청하는 매서드. 해당 정보를 리포지토리에 전달 후 **List** 객체로 반환

2. 스케줄 등록 매서드
전달 받은 **dto**를 **entity**로 변환하여 리포지토리에 전달 후 데이터 베이스에 저장

3. 스케줄 수정 매서드
수정할 스케줄이 존재하는지 확인 후
전달 받은 **dto**를 **entity**로 변환하여 리포지토리에 전달 하여 데이터 베이스에 저장

4. 스케줄 삭제 매서드
스케줄 **id**를 리포지토리에 전달하여 삭제

5. 스케줄 리파지토리
전달받은 맴버 아이디, 연도, 월을 포함한
데이터를 조회하는 매서드.

JPQL 방식으로 **SQL**문을 작성하여 데이터 베이스를 조회하는 방식

CalendarUtils.js

담당자 : 김성우

1

```
export function renderCalendar(currentDate, schedules) { Show usages ▾ seongwoo

const today : Date = new Date();
const dayNames : string[] = ['일', '월', '화', '수', '목', '금', '토'];
const year : number = currentDate.getFullYear();
const month : number = currentDate.getMonth();
const firstDay : number = new Date(year, month, date: 1).getDay();
const lastDay : number = new Date(year, monthIndex: month + 1, date: 0).getDay();
const lastDate : number = new Date(year, monthIndex: month + 1, date: 0).getDate();
const prevLastDate : number = new Date(year, month, date: 0).getDate();

const daysArray : any[] = [];
```

2

```
for (let i : number = firstDay - 1; i >= 0; i--) {
  const date : Date = new Date(year, monthIndex: month - 1, date: prevLastDate - i);
  const prevMonth : number = month === 0 ? 11 : month - 1;
  const prevYear : number = month === 0 ? year - 1 : year;
  const scheduleItems = schedules.filter(s =>
    s.date === `${prevYear}-${String(value: prevMonth + 1).padStart(2, '0')}-${String(value: prevLastDate - i).padStart(2, '0')}`
  );
  daysArray.push({
    year: prevYear, month: prevMonth, day: prevLastDate - i,
    dayName: dayNames[date.getDay()],
    inactive: true, schedules: scheduleItems
  });
}
```

3

- 달력 UI를 만드는 컴포넌트
각 날짜의 정보를 객체의 형태로 배열에 담아 반환한다.
- firstDay** = 이번 달 1일의 요일
lastDay = 이번 달 마지막 요일
lastDate = 이번 달 마지막 날짜
prevLastDate = 지난 달 마지막 날짜
- 날짜 객체를 저장할 배열
- 이전 달 날짜 저장
이번 달 1일의 요일에 따라 이전 달의 마지막 며칠을 앞쪽에 채움

CalendarUtils.js

담당자 : 김성우

1

```
for (let i : number = 1; i <= lastDate; i++) {
    const date : Date = new Date(year, month, i);
    const isToday : boolean = year === today.getFullYear() && month === today.getMonth() && i === today.getDate();
    const scheduleItems = schedules.filter(s =>
        s.date === `${year}-${String(s.value).padStart(2, '0')}-${String(i).padStart(2, '0')}`
    );
    daysArray.push({
        year, month, day: i, dayName: dayNames[date.getDay()],
        today: isToday, inactive: false, schedules: scheduleItems
    });
}
```

2

```
for (let i : number = 1; i < 7 - lastDay; i++) {
    const date : Date = new Date(year, monthIndex: month + 1, i);
    const nextMonth : number = month === 11 ? 0 : month + 1;
    const nextYear : number = month === 11 ? year + 1 : year;
    const scheduleItems = schedules.filter(s =>
        s.date === `${nextYear}-${String(s.value).padStart(2, '0')}-${String(i).padStart(2, '0')}`
    );
    daysArray.push({
        year: nextYear, month: nextMonth, day: i,
        dayName: dayNames[date.getDay()],
        inactive: true, schedules: scheduleItems
    });
}

return daysArray;
```

3

1. 이번 달 날짜 저장
이번 달의 1일부터 마지막 날까지 반복

2. 다음 달 날짜 저장
달력 마지막 줄을 완성하기 위해 필요한 만큼 다음 달의 앞 날짜들을 채움

3. 저번 달, 이번 달, 다음 달 날짜를 채운 배열을 반환

ExpenseRepository.java

담당자 : 박시진

1

```
@Repository
public interface ExpenseRepository extends JpaRepository<Expense, Long> {
    List<Expense> findByUsername(String username);
    List<Expense> findByStatus(RequestStatus status);

    @Query("SELECT e FROM Expense e WHERE " +
        "(:searchTerm IS NULL OR " +
        "e.title LIKE %:searchTerm% OR " +
        "e.username LIKE %:searchTerm% OR " +
        "e.category LIKE %:searchTerm% OR " +
        "e.content LIKE %:searchTerm% OR " +
        "CAST(e.amount AS string) LIKE %:searchTerm% OR " +
        "CAST(e.expenseDate AS string) LIKE %:searchTerm% OR " +
        "(CASE " +
        "    WHEN e.status = 'PENDING' THEN '대기' " +
        "    WHEN e.status = 'APPROVED' THEN '승인' " +
        "    WHEN e.status = 'REJECTED' THEN '거절' " +
        "    ELSE CAST(e.status AS string) " +
        "END) LIKE %:searchTerm%)")
    List<Expense> searchExpenses(@Param("searchTerm") String searchTerm);
}
```

2

1. **JpaRepository**를 상속하는 **Expense** 전용 리포지토리. 기본적인 **CRUD** 기능은 자동 제공되고, 여기에 사용자 정의 메서드를 추가한 형태.
2. **searchTerm**이 **null**이 아니면 다양한 필드 (**title, username, category, content, amount, expenseDate, status**)에 대해 부분 일치 검색을 수행. **status**는 **Enum**이기 때문에, **CASE** 구문을 통해 "대기", "승인", "거절" 등의 한글 문자열로 매핑해 검색 가능하게 처리.

ExpenseService.java - 경비의 개수 카운트

담당자 : 박시진

1

```
public long getPendingExpensesCount() {  
    List<Expense> pendingExpenses = expenseRepository.findByStatus(RequestStatus.PENDING);  
    return pendingExpenses.size();  
}
```

2

```
public long getRejectedExpensesCount() {  
    List<Expense> rejectedExpenses = expenseRepository.findByStatus(RequestStatus.REJECTED);  
    return rejectedExpenses.size();  
}
```

- 상태가 **PENDING**인 경비의 개수를 구함
`expenseRepository.findByStatus(RequestStatus.PENDING)`를 호출하여 상태가 **PENDING**인 모든 경비를 찾음.

`pendingExpenses.size()`로 해당 리스트의 크기(즉, **PENDING** 상태인 경비의 개수)를 반환.

- 상태가 **REJECTED**인 경비의 개수를 구함

`expenseRepository.findByStatus(RequestStatus.REJECTED)`를 호출하여 상태가 **REJECTED**인 모든 경비를 찾음.

`rejectedExpenses.size()`로 해당 리스트의 크기(즉, **REJECTED** 상태인 경비의 개수)를 반환.

ExpenseService.java - 경비관리 목록 조회

담당자 : 박시진

```

public Expense getExpense(Long id) {
    Expense expense = expenseRepository.findById(id)
        .orElseThrow(() -> new ExpenseNotFoundException("Expense not found with id: " + id));

    1   Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        boolean isAdmin = authentication.getAuthorities().stream().anyMatch(authority -> authority.getAuthority().equals("ROLE_ADMIN"));

    2   if (!expense.getUsername().equals(authentication.getName()) && !isAdmin) {
        throw new org.springframework.security.access.AccessDeniedException("You do not have permission to access this expense.");
    }

    return expense;
}

public List<Expense> getExpenses() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    boolean isAdmin = authentication.getAuthorities().stream().anyMatch(authority -> authority.getAuthority().equals("ROLE_ADMIN"));

    3   if (isAdmin) {
        return expenseRepository.findAll();
    } else {
        4   String currentUsername = authentication.getName();
        return expenseRepository.findByUsername(currentUsername);
    }
}

```

1. **SecurityContextHolder.getContext().getAuthentication()**을 통해 현재 인증된 사용자의 정보를 가지고 옴.
authentication.getAuthorities()로 사용자의 권한을 확인하고, 관리자인지 아닌지 (**ROLE_ADMIN**)를 체크.
2. 만약 사용자가 **Expense**를 작성한 사용자와 다르고, 관리자가 아니라면, **AccessDeniedException**을 던져서 접근을 차단.
3. 관리자는 **expenseRepository.findAll()**을 호출해서 모든 경비를 조회.
4. 일반 사용자는 **expenseRepository.findByUsername(currentUsername)**을 호출하여 자기 자신의 경비만 조회.

ExpenseService.java - 이미지 생성 및 저장

담당자 : 박시진

1

```
private String saveImage(MultipartFile image) {
    try {
        File uploadDirFile = new File(uploadDir);
        if (!uploadDirFile.exists() && !uploadDirFile.mkdirs()) {
            logger.error("이미지 업로드 디렉토리 생성 실패: {}", uploadDirFile.getPath());
            throw new RuntimeException("이미지 업로드 디렉토리 생성 실패");
        }

        String fileName = System.currentTimeMillis() + "-" + Math.random() + "-" + image.getOriginalFilename();
        File uploadFile = new File(uploadDirFile, fileName);
        image.transferTo(uploadFile);

        return serverUrl + "/" + fileName;
    } catch (IOException e) {
        logger.error("이미지 업로드 실패: {}", e.getMessage());
        throw new RuntimeException("Image upload failed: " + e.getMessage());
    }
}
```

2

3

4

5

1. 업로드된 이미지를 서버에 저장하고, 접근 가능한 **URL**을 리턴하는 함수.

2. **uploadDir** 디렉토리가 없으면 생성 시도 생성 실패하면 로그 찍고 예외 발생.

3. 현재 시간 + 랜덤값 + 원래 파일명 조합. 중복 방지용

4. 지정된 경로에 이미지 파일 저장

5. 저장된 이미지의 **URL** 리턴

ExpenseWrite.js - 금액 계산 및 합계 표시

담당자 : 박시진

1

```
const calculateTotalAmount = () => {
  const total = expense.amounts.reduce(
    (total, item) => total + (Number(item.amount.replace(/,/g, "")) || 0),
    0
  );
  return total > 0 ? total : 0;
};
```

2

```
<p className="fs_lg">전체 금액 </p>
<input
  className="input fs_md mb_md input-max p_sm"
  name="totalAmount"
  type="text"
  value={formatAmount(String(calculateTotalAmount()))}
  disabled
/>
```

1. **expense.amounts** 배열의 각 항목에서 금액 문자열(1,000 같은 거)에서 , 제거하고 숫자로 바꾼 뒤 더해줌.

2. 위에서 계산된 총합을 **formatAmount()**로 포맷팅해서 3자리마다 , 찍어서 보여줌.
disabled 속성으로 사용자가 수정 못 하게 막아놨

ApprovedExpensesPage.js - 카테고리별 조회

담당자 : 박시진

1

```
const [selectedCategory, setSelectedCategory] = useState("");
```

2

```
const handleCategoryChange = (category, e) => {
  if (category === selectedCategory) {
    setSelectedCategory("");
  } else {
    setSelectedCategory(category);
  }
};
```

3

```
const categoryFilter =
  selectedCategory === "" || expense.category === selectedCategory;

return (
  expense.status === "APPROVED" &&
  expenseYear === selectedYear &&
  expenseMonth === selectedMonth &&
  categoryFilter
);
```

4

```
<div className="flex space-around">
  {categories.map((category) => (
    <button
      key={category}
      type="button"
      className={`expense-button mb_md ${(
        selectedCategory === category ? "selected" : ""
      )}`}
      onClick={(e) => handleCategoryChange(category, e)}
    >
      {category}
    </button>
  )))
</div>
```

1. 기본값은 빈 문자열 ""이어서 모든 카테고리가 보이도록 설정.

2. 이미 선택된 카테고리를 다시 클릭하면 전체 보기로 전환.

다른 카테고리를 클릭하면 그걸 선택한 걸로 바꿔.

3. 선택한 연도 + 월 + (카테고리)에 맞는 승인된 지출 항목들만 화면에 보이도록 정리

4. **categories** 배열에 있는 다섯 개의 항목(식비, 교통, 숙박, 경조사, 기타)을 반복하면서 버튼을 생성.

선택된 버튼은 "**selected**" 클래스를 추가해서 스타일이 바뀌도록 설정.

버튼 클릭 시 **handleCategoryChange** 함수가 실행.

ExpenseUpdate - 이미지 미리보기 및 삭제

담당자 : 박시진

1

```
const handleFileChange = (e) => {
  const selectedFiles = Array.from(e.target.files);
  const imageUrl = selectedFiles.map((file) => URL.createObjectURL(file));
  setPreviewImages((prevImages) =>
    Array.isArray(prevImages) ? [...prevImages, ...imageUrl] : [...imageUrl]
  );
  setFiles((prevFiles) =>
    Array.isArray(prevFiles)
      ? [...prevFiles, ...selectedFiles]
      : [...selectedFiles]
  );
};
```

2

```
const handleRemoveImage = (index) => {
  const updatedImages = previewImages.filter((_, i) => i !== index);
  const updatedFiles = files.filter((_, i) => i !== index);
  setPreviewImages(updatedImages);
  setFiles(updatedFiles);
};
```

3

```
{previewImages && previewImages.length > 0 && (
  <div className="">
    {previewImages.map((src, index) => (
      <div key={index} className="item">
        <img
          className="item"
          src={src}
          alt={`미리보기 ${index + 1}`}
        />
        <button
          className="btn btn-sm btn-pl fs_sm"
          type="button"
          onClick={() => handleRemoveImage(index)}
        >
          X
        </button>
      </div>
    )));
  </div>
)}
```

1. 사용자가 파일을 선택하면

handleFileChange가 호출. 이 함수는 선택된 파일들을 **selectedFiles** 배열에 저장하고, 이를 사용해 **URL.createObjectURL**로 임시 **URL**을 생성. 이 **URL**은 이미지를 미리 보기 위해 사용.

setPreviewImages를 사용하여 이미지 미리 보기 위한 **URL**을 **previewImages** 상태에 저장하고, **setFiles**로 실제 파일들을 **files** 상태에 저장.

2. 사용자가 삭제 버튼을 클릭하면

handleRemoveImage가 호출되고, 해당 인덱스에 해당하는 이미지를 **previewImages**와 **files**에서 제거.

3. **previewImages** 배열에 저장된 이미지 **URL**들을 화면에 표시.

각 이미지는 **img** 태그로 렌더링되고, 각 이미지 옆에는 삭제 버튼이 추가. 삭제 버튼은 해당 이미지를 삭제

FileController.java

담당자 : 손수용

1

```
@PostMapping("/upload") public ResponseEntity<String> uploadFile(@RequestParam("file") MultipartFile file) { try { // 프로젝트 루트 + uploads String rootPath = System.getProperty("user.dir"); String uploadPath = rootPath + "/uploads/"; File dir = new File(uploadPath); if (!dir.exists()) dir.mkdirs(); // URL도 상대 경로 기반으로 구성 String fileName = UUID.randomUUID() + "_" + file.getOriginalFilename(); File savedFile = new File( pathname: uploadPath + fileName); file.transferTo(savedFile); } catch (IOException e) { return ResponseEntity.status(500).body(e.getMessage()); } }
```

2

```
// 프로젝트 루트 + uploads  
String rootPath = System.getProperty("user.dir");  
String uploadPath = rootPath + "/uploads/";  
File dir = new File(uploadPath);  
if (!dir.exists()) dir.mkdirs();
```

3

```
String fileName = UUID.randomUUID() + "_" + file.getOriginalFilename();  
File savedFile = new File( pathname: uploadPath + fileName);  
file.transferTo(savedFile);
```

4

```
// URL도 상대 경로 기반으로 구성  
String fileUrl = "http://localhost:8080/uploads/" + fileName;  
return ResponseEntity.ok(fileUrl);
```

1. **System.getProperty("user.dir")** 를 이용해 현재 프로젝트 루트 경로를 가져온 후 뒤에 "**/uploads/**" 를 붙여 실제 파일 저장 경로를 만든다.

2. 만일 **uploads** 폴더가 존재하지 않을 경우, **mkdirs()** 를 이용하여 폴더를 만든다.

3. 파일 이름 충돌을 막기 위해 **UUID**로 파일 이름 앞에 랜덤문자열을 붙인 후, 지정 한 경로에 그 이름으로 빈 파일을 만들고 **file.transferTo()**를 이용하여 **file** 정보를 해당 파일로 데이터를 복사한다.

4. 브라우저에서 업로드 된 파일에 접근 할 수 있도록 파일 경로를 **return** 한다.

FileController.java

담당자 : 손수용

```
1 @GetMapping("/download/{fileName:.+}") public ResponseEntity<Resource> downloadFile(@PathVariable String fileName) throws IOException {  
    2     Path filePath = Paths.get("uploads", fileName);  
    Resource resource = new UrlResource(filePath.toUri());  
  
    if (!resource.exists()) {  
        return ResponseEntity.notFound().build();  
    }  
  
    3     return ResponseEntity.ok()  
        .contentType(MediaType.APPLICATION_OCTET_STREAM)  
        .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" + fileName + "\"")  
        .body(resource);  
}
```

1. GET 방식으로 /download/{fileName} 경로로 들어오는 요청을 처리한다. {fileName:.+}는 “.”이 포함된 파일 이름도 허용하게 하는 정규표현식을 사용한 것이다.

2. Paths.get 을 이용해 “uploads” + fileName 과 같은 경로를 생성 후 UrlResource 객체를 생성한다.

3. 이와 같은 객체를 리턴할 때, MediaType.APPLICATION_OCTET_STREAM 을 이용해 바이너리 형식의 데이터(즉, 파일 다운로드용 데이터)로 지정 후, “attachment; filename=...” 으로 헤더를 지정해 브라우저가 응답을 받으면 다운로드 창이 뜨게 만들어준다.

TemplateCreate.js

담당자 : 손수용

create - 템플릿을 생성하는 함수이다.

1

```
const create = async (e) : Promise<void> => { Show usages ⚡ gadley*
  e.preventDefault();

  const orderedQuestionList :[] = questionList.map((q, index : number ) : any & {} => ({
    ...q, order: index + 1
  }));

  console.log(orderedQuestionList);

  const data :{} = {title, icon, color, questionList: orderedQuestionList};

  try {
    const response :AxiosResponse<any> = await axios.post( url: "http://localhost:8080/templates/create", data);
    navigate(-1);
  } catch (error) {
    console.error(error);
  }
}
```

2

```
...q, order: index + 1
});
```

3

```
const data :{} = {title, icon, color, questionList: orderedQuestionList};
```

```
try {
  const response :AxiosResponse<any> = await axios.post( url: "http://localhost:8080/templates/create", data);
  navigate(-1);
} catch (error) {
  console.error(error);
}
```

```
const questionAdd = () :void => { Show.usages ⚡ gadley
  const newId :string = `q${Date.now()}`;
  const newQuestion :{} = {id: newId, type: "TEXT", content: "", optionList: []};
  setQuestionList( value: (prev :any[]) => [...prev, newQuestion]);
}
```

4

```
const newId :string = `q${Date.now()}`
```

```
const newQuestion :{} = {id: newId, type: "TEXT", content: "", optionList: []};
setQuestionList( value: (prev :any[]) => [...prev, newQuestion]);
```

5

1. 폼의 기본 동작인 페이지 새로고침을 막는다.

2. **questionList**의 각 질문에 **order** 값을 추가해 정렬 정보를 명시한다. 그리고 ...q를 이용해 기존 질문 객체를 복사한다.

3. 최종적으로 보낼 데이터 객체 구성 후 **POST** 요청을 백엔드에 보낸다. 그 후 이전 페이지로 이동한다.

questionAdd - 질문을 추가하는 함수이다.

4. 각 질문을 구분하기 위해 **Date.now**를 이용하여 **id**를 생성한다.

5. **questionList**에 ...prev를 이용해 기존 질문들을 담고, **newQuestion**을 추가 한다.

TemplateCreate.js

담당자 : 손수용

1

```
<DragDropContext onDragEnd={handleDragEnd}>
  <Droppable droppableId="questions">
    {(provided: DroppableProvided) => (
      <div
        {...provided.draggableProps}
        ref={provided.innerRef}
      >
        {questionList.map((q, questionIndex: number) => (
          <Draggable key={q.id} draggableId={q.id}
                      index={questionIndex}>
            {(provided: DraggableProvided) => (
              <div
                className={"draggableItem"}
                ref={provided.innerRef}
                {...provided.draggableProps}
              >
                <div className="dragHandle"
                    {...provided.dragHandleProps}
                  >
                    <hr/>
                    <hr/>
                    <hr/>
                  </div>
                <TemplateQuestionBlock
                  question={q}
                  questionIndex={questionIndex}
                  updateQuestion={(i, newQ): void => {
                    const updated: any[] = [...questionList];
                    updated[i] = newQ;
                    setQuestionList(updated);
                  }}
                />
              </div>
            )})
          </Draggable>
        )));
      {provided.placeholder}
    </div>
  </Droppable>
</DragDropContext>
```

2

3

4

5

1. DnD 전체 컨트롤러인 **DragDropContext**이다. 드래그가 끝났을때 **handleDragEnd** 함수를 실행하라고 명시되어있다. **Droppable**은 드롭 가능한 영역을 명시한다.
2. 각 질문을 **Draggable**로 감싸서 드래그 가능한 영역임을 명시한다.
3. **provided.innerRef**는 해당 둘 요소를 추적할 수 있게 도와주는 레퍼런스이며, **...provided.draggableProps**는 드래그 기능을 부여하는 속성들이 담겨있다.
4. 실제로 마우스로 클릭해서 끌 수 있는 영역을 명시한다.
5. 드래그 중인 요소의 자리를 비워주는 자리 표시자이다.

ReportUpdate.js

담당자 : 손수용

- 1
- 2
- 3
- 4

```
const handleFileChange = async (e, questionId) : Promise<void> => { Show usages
    const file = e.target.files[0];
    if (!file) return;
    const formData : FormData = new FormData();
    formData.append("name", "file", file);
    try {
        const res : AxiosResponse<any> = await axios.post(url: "http://localhost:8080/file/upload", formData, config: {
            headers: { "Content-Type": "multipart/form-data" },
        });
        const fileUrl = res.data;
        setResponseData( value: (prev : any[]) => prev.map((r) : any | {} => r.questionId === questionId ? { ...r, fileUrl } : r));
    } catch (err) {
        console.error("파일 업로드 실패", err);
    }
};
```

1. **input** 으로 받은 파일을 가져온다. 만약 존재하지 않을 경우 바로 **return** 한다.

2. **formData** 객체를 만들어서 그 안에 파일을 담는다. **multipart/form-data** 방식으로 파일을 전송할 때 사용된다.

3. **POST** 요청을 보낸다. **Content-Type** 또한 **multipart/form-data** 형식임을 알려준다.

4. 응답으로 받은 **URL**을 **responseData**에 담아 넣는다. **responseData** 배열을 순회하면서 현재 처리중인 **questionId** 와 같을 경우에 **fileUrl**을 담아넣고 아닐 경우 그대로 유지한다.

05. 프로젝트 회고

5-1. 회고

05. 프로젝트 회고

5-1. 회고



프론트 / 백엔드 개발
스케줄 파트 담당
공통 스타일 CSS 제작
공용 캘린더 개발
캘린더 시스템 내 CRUD 기능 개발

김성우

근태 관리 웹 어플리케이션 '**Offime**' 개발 프로젝트에서 **PM(팀장)**을 맡게되었습니다.

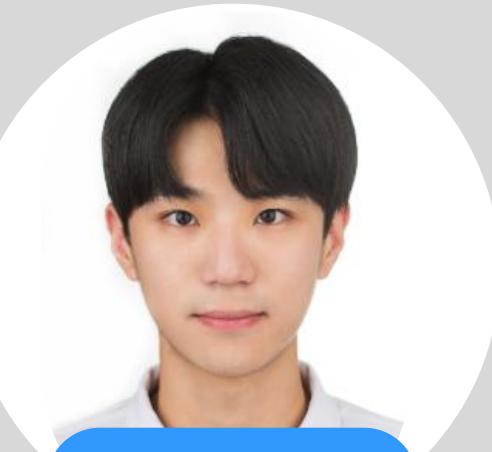
React 기반의 캘린더 프론트엔드와 **Java, Spring Boot**를 활용한 백엔드 **API** 서버를 직접 구현하여, 일정을 등록, 조회, 수정, 삭제할 수 있는 일정 관리 시스템(**CRUD**)을 개발하였습니다. 프론트엔드와 백엔드를 모두 담당하며, 전체 흐름을 이해하고 설계부터 구현까지 주도적으로 수행한 경험이었습니다.

프론트 엔드 파트에서 특히 신경을 쓴 부분은 컴포넌트의 리펙토링이었습니다. 달력 전체를 감싸고 **State**를 초기화 하고 전달하는 **CalendarContainer.js**, 선택한 월과 연도를 표시하고 월과 연도를 변경하는 컴포넌트를 호출하는 **CalendarHeader.js**, 날짜가 들어갈 자리인 **DayGrid.js**, 선택한 연,월에 맞는 날짜를 **DayGrid**에 채우는 **CalendarUtils.js**로 분리하여 캘린더를 구현하였고, 그외 스케줄을 등록하는 **ScheduleForm.js**, 스케줄을 수정하는 **ScheduleEditForm.js**, 스케줄을 상세 조회하는 **ScheduleDetail.js** 등 모든 기능을 각각의 컴포넌트로 분리하여 수정과 유지보수를 용이하게 만들고자 노력했습니다.

특히 달력 컴포넌트를 만드는 것은 재미있는 경험이었는데요, 라이브러리를 사용하지 않고 순수 **JavaScript**만을 이용하여 날짜를 계산하고 해당하는 연,월에 맞는 날짜를 반환하는 알고리즘을 만들며 **JavaScript**를 활용하는 능력이 크게 향상되었습니다.

이번 프로젝트에서 클라이언트와 서버를 동시에 개발하면서 전체 시스템의 데이터 흐름을 총체적으로 이해하게 되었고, 인터페이스 설계의 중요성을 체감하였습니다. 특히 프론트에서 사용하는 날짜 포맷이 백엔드에서의 매핑에 직접적인 영향을 준다는 점에서, 계층 간 규약의 중요성을 실감했습니다.

팀장 역할을 맡아 정기, 비정기 적으로 회의를 열어 주체적으로 프로젝트를 진행하며 전반적인 일정을 관리하고, 유동적으로 팀원들의 담당 업무와 역할을 분배하며 프로젝트를 이끈 것 또한 값진 경험이었습니다.



프론트 / 백엔드 개발
멤버 파트 담당
JWT를 이용한 [인증 시스템](#) 개발
회원가입 신청 및 승인 기능 개발

최수민

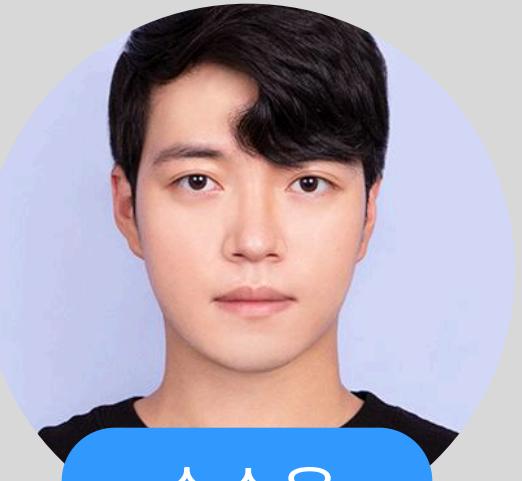
이번 프로젝트에서 회원가입, 로그인, 마이페이지 기능을 맡았고 이 기능들은 사용자 인증 및 관리와 관련된 핵심적인 부분들이기 때문에, 보다 신중하게 구현해야 했다. 특히, **JWT** 토큰을 사용하여 인증을 처리하는 부분에서 많은 고민을 했다. 물론 **JWT** 토큰을 사용하는 것은 학원 수업 중에도 몇 번 다뤄본 적이 있었다. 사용자가 로그인하면 서버에서 **JWT** 토큰을 발급하고, 그 토큰을 이용해 이후의 요청을 인증하는 방식인데, 처음에는 그 흐름을 이해하는데 어려움이 있었다.

구체적으로, 사용자가 로그인할 때 입력한 이메일과 비밀번호를 서버에서 인증한 후, 서버는 **JWT** 토큰을 발급한다. 이 토큰을 클라이언트로 전달하고, 클라이언트는 이후 요청 시 이 토큰을 헤더에 담아서 서버에 전달한다. 서버는 이 토큰을 디코딩하여 유효성을 검사하고, 이를 통해 사용자가 인증된 사용자임을 확인한다. 이 과정에서 가장 어려웠던 점은 **JWT** 토큰을 발급하고 검증하는 흐름을 명확하게 이해하고 구현하는 것이었다.

토큰 자체가 짧은 시간 동안만 유효하므로, 그 유효기간을 설정하는 부분도 신경을 써야 했다. 그리고 토큰이 올바르지 않거나 만료된 경우에는 적절한 예외를 처리해야 했다. 이러한 부분은 문서나 자료를 찾아보면서 조금씩 이해할 수 있었고, 결국 데이터가 어떻게 흐르고, 어떻게 보안을 유지하는지를 명확히 알게 되었다.

아쉬운 점은, 프로젝트 요구 사항과 시간 제약으로 인해 **JWT**만을 사용하게 되었는데, **OAuth2**와 세션 같은 다른 인증 방식이 가지는 장점과 단점을 직접 경험해보지 못한 점이 아쉬움으로 남는다. 향후에는 다른 방식들을 적용해보며 다양한 인증 기법에 대한 깊이 있는 이해를 넓혀가고 싶다.

프로젝트를 끝내고 나니, 회원 인증과 관련된 기능을 담당하면서 백엔드와 클라이언트의 상호작용이 얼마나 중요한지 다시 한 번 깨닫게 되었다. 특히, **JWT** 토큰을 통한 인증은 이전에 다뤄본 적이 있지만, 이렇게 실제 프로젝트에서 적용해보니 많은 점에서 실제 서비스에서 어떻게 활용되는지에 대한 감이 잡히게 되었다. 이 프로젝트는 단순한 기능 구현 이상의 많은 배움을 주었고, 앞으로 더 나은 개발자가 될 수 있도록 도와준 좋은 경험이었다.



프론트 / 백엔드 개발
보고서 파트 담당
템플릿 작성 기능 개발
보고서, 댓글 CRUD 개발
드래그 앤 드롭 기능 구현
손수용

이번 **Offime** 프로젝트에서 가장 기억에 남는 기능은 **DnD** 기반 템플릿 생성 기능이다.
드래그 앤 드롭은 처음 도전해보는 기술이었고, 동작했을 때 굉장히 신기했다.
하지만 복잡한 구조와 많은 상태 변화 때문에 구현 난이도도 높아 특히 인상 깊었다.
그 외에도 객관식 답변 추가 기능 구현에서 꽤 막혔던 기억이 있다.
배열 안에 배열이 있는 구조에 컴포넌트 분리까지 겹치며 로직이 꼬여 어려움을 겪었지만,
이 과정을 통해 컴포넌트 상태 관리와 데이터 흐름에 대한 이해를 높일 수 있었다.
협업할 때는 마감 기한을 맞추는 것에 신경 썼다.
팀원 간의 흐름을 끊지 않도록 내 작업을 늦추지 않으려고 노력했다.
특히 이번 프로젝트에선 **GitHub** 협업 경험이 좋았다.
이전보다 훨씬 적극적으로 커밋하고 **Pull**을 받아 최신 버전으로 유지하는 연습이 많이 됐다.
개인적으로는 **CRUD** 구현에 익숙해졌다는 점이 가장 큰 수확이다.
이전 프로젝트에서 멤버 관련 **CRUD**를 해본 경험이 있었기에 이번엔 복습처럼 느껴졌고,
그만큼 자신감도 생겼다.
아쉬운 점이 있다면 코드 구조가 조금 번잡했던 점이다.
다음 프로젝트에서는 컴포넌트 구조와 폴더 설계를 사전에 명확히 계획하고 시작해야겠다고 느꼈다.



프론트 / 백엔드 개발
비용처리 파트 담당
비용처리 관련 CRUD 개발
헤더 / 푸터 제작
제미니 AI 연동 챗봇 기능 구현

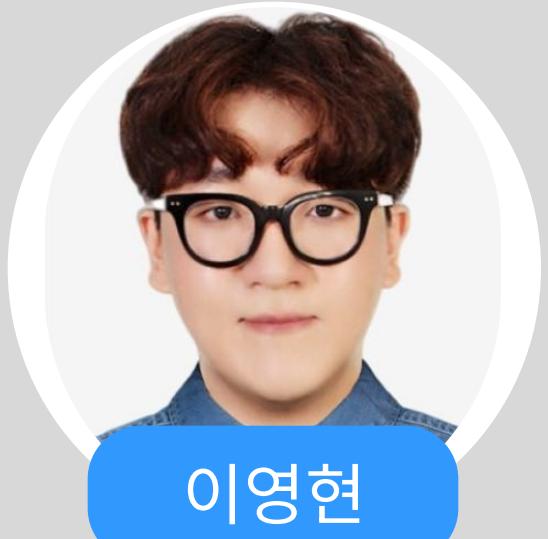
박시진

이번 프로젝트는 전프로젝트보다 크기가 크지 않았던 것 같다. 그렇기에 조금이라도 사용자를 생각해 편의를 위한 기능들을 추가하기 위해 노력한 것 같다. 경비 작성 기능을 개발하면서 사용자의 실제 사용 흐름을 최대한 고려해보려 했다. 예를 들어 금액 입력 시 숫자가 자동으로 쉼표 처리되도록 하여 가독성을 높였고, 금액이 여러 항목으로 나뉘는 경우를 고려해 복수 입력 및 총합 자동 계산이 가능하도록 구성했다. 이 과정에서 **amount** 데이터를 배열로 관리하고 이를 기반으로 총합을 실시간으로 반영하는 로직을 구현했다. 이를 통해 금액이 변경될 때마다 자동으로 합산되도록 하여 사용자가 직접 계산할 필요 없이 편리하게 사용할 수 있도록 했다.

이미지 업로드 기능에서도 단순히 파일을 선택하는 수준을 넘어서 이미지 미리보기와 개별 삭제 기능을 제공하여 사용자의 편의성을 높였다. 이 과정에서 파일과 미리보기 **URL** 간의 일관성 유지가 중요하다는 점을 배웠고, 버그를 잡아가며 디테일한 **UI** 동작 설계의 필요성을 느꼈다. 카테고리 선택 또한 버튼 **UI**로 구성하여 보다 명확하고 직관적인 입력이 가능하도록 구성했으며, 전체적인 흐름 속에서 작성 후 데이터를 서버에 **FormData**로 묶어 보내는 백엔드 연동도 함께 구현함으로써 프론트와 백을 함께 고려한 구조를 경험할 수 있었다.

이번 작업을 통해 "작은 기능도 어떻게 구성하느냐에 따라 사용자 경험이 완전히 달라질 수 있다"는 것을 실감했고, 앞으로 더 나은 **UX**를 설계하기 위한 고민과 훈련이 필요하다는 자극을 받았다.

나름 매끄럽게 기능들이 이상 없이 잘 작동하니, 스스로도 잘 만들었다는 느낌을 받을 수 있었다. 디자인 역시 크게 투지 않으면서도 무난하게 마무리된 것 같아 만족스러웠다. 다만 아직 실력이 부족한 만큼, 코드 정리도 미흡했고 구현 방식이나 구조에서 개선할 부분이 많다는 것을 느꼈다. 앞으로는 다른 사람들이 만든 프로젝트들을 더 많이 참고하면서, 직접 만들어보고 시행착오를 겪는 과정 자체가 성장에 있어 가장 중요하다고 생각한다. 이번 작업을 통해 작동하는 것만으로 만족하지 않고, 더 나은 코드와 사용자 경험을 고민해야 한다는 동기부여를 얻었다.



프론트 / 백엔드 개발
출결 및 근태 현황 파트 담당
Geolocation api를 이용한 GPS 기반 출퇴근
처리 기능 개발
출퇴근 현황 페이지 개발

이영현

이번 **Offime** 프로젝트에서 내가 맡은 출결 및 근태관리 파트에서 중점은 둔 부분은 첫째로 코드의 품질을 올려 실무와 흡사한 깔끔한 코딩을 구현하는 것과, 둘째론 편리한 사용자 경험의 추구였다.

백엔드에서는 **setter**를 쓰지 않고 비즈니스 메소드와 생성자/팩토리 메소드로 객체의 상태를 관리해서 유지보수성, 안정성은 물론 내 코드를 리뷰할 다른 개발자들에게도 쉽게 이해 될 수 있도록 신경 썼고, 필드의 값을 제어할 때에도 문자열 대신 **enum** 타입을 사용하여 타입 안정성과 데이터 무결성을 보장할 수 있도록 하였다. 또한, **DB**의 테이블인 **eventRecord**에 출결 기록이 저장될 때에 같은 사람의 하루 동안의 기록이라면 자리비움과 복귀 시에는 출근 기록의 **clockIn, late** 시간을 받아와서 함께 저장되게 하고, 퇴근 시에도 **clockOut, leaveEarly** 시간을 그날의 모든 출결 기록에 업데이트하도록 설계하거나 **JPA AttributeConverter**를 활용하여 각 기록의 시간을 저장할 때 **localDateTime**과 **localTime**의 나노초를 절삭해서 **DB** 조회 시에도 정보의 가독성을 높였으며 프론트에서 표시될 때 가독성을 위해 별도의 가공이 필요하지 않도록 설계했다.

프론트엔드에서는 출결 배너는 필요한 정보만을 좋은 가독성으로 제공하고, 내 근태 이력과 출퇴근 현황에 진입 시에는 항상 오늘의 정보를 먼저 표시하도록 해서 사용자 경험을 고려했다. 또한, 직원의 출결 상태 판단을 위해 ‘정상 출근’, ‘출근 전’, ‘미출근’으로 나누어 휴가자는 ‘미출근’ 인원으로, 09시 이전에 조회했을 때 미리 출근한 인원을 제외한 다른 인원들은 ‘출근 전’으로 표시하다 09시 이후에는 ‘미출근’으로 표시되도록 해서 출퇴근 현황에서 리더에게 직원의 출결 상태를 일목요연하게 표현할 수 있도록 구조화에 많은 신경을 썼다.

전체적으로 이번 프로젝트를 진행하며 데이터를 생성하는 모든 로직은 백엔드에서 전담하고, 프론트엔드는 백엔드에서 넘겨주는 정보를 선택적으로 표시하기만 할 수 있는 깔끔한 설계를 추구했다. 그래서 많은 기능을 구현하는 데에 욕심을 부리기보다는 규모를 한정되게 잡고, 맡은 부분에서 퀄리티를 올리는 것에 집중했다. 또한 6명의 팀으로 진행하며 **GitHub**를 적극 사용해서 버전관리를 다함께 참여했기에 값진 협업 경험이 될 수 있었던 것 같다.

감사합니다!