QnA ToCafe **CafeOn.** 마이페이지 로그아웃

흩어진 정보를 한눈에 '지금' 필요한 답을 빠르게.

상황.분위기 기반 카페 탐색 서비스

한눈에 보는 리뷰와 실시간 커뮤니티로
카페 선택까지 걸리는 시간을 줄입니다.

기간: 9/19~11/5 (6주)
팀: 백엔드(3명), 프론트엔드(2명)
파트: 프론트엔드
-마이페이지 (회원정보, 리뷰, 북마크, 채팅방, 히스토리, 문의)
-채팅 (카페 단체 채팅, DM)
-커뮤니티 페이지
-QnA 페이지
-개인 실시간 알림

문제 정의

"정보 파편화로 인한 탐색 과부하"

한 카페를 결정하기 위해 인스타그램.카카오맵.네이버 등 여러 채널을 전전해야 해서 과다한 시간 소모
동일 정보(주소/영업시간/분위기)가 매체마다 달라 맥락 정리와 비교가 어려움

통합 요약
태그,기본정보를 한 화면으로
탐색 시간 단축

"실시간성 부재로 인한 재탐색 반복"

갑작스러운 휴무/만석/공지를 사전에 확인하기 어려워, 방문 후에야 닫힘을 알게 되고 처음부터 다시 탐색
현재 자리 상황/대기/소음 등 같은 현장 정보가 없어 결정 리스크가 높음

실시간 현재 정보
카페별 실시간 채팅
-헛걸음 방지

"신뢰도 저하(정보 포화+광고/생성형 리뷰 노이즈)"

기존 플랫폼은 광고성.협찬성.AI 생성 리뷰가 뒤섞여 있어 핵심 판단 근거 추출이 어려움
과포화된 결과 속에서 정확한 리뷰와 요점을 찾는 과도한 탐색 비용

핵심 근거
공통 키워드 중심 요약 명시로 판단 신뢰도 강화
-광고 신고 버튼

프로젝트 목표

카페 이용자들의 생생한 후기와 실시간 소통을 통해
카페의 분위기와 **'지금'** 필요한 정보를
빠르게 파악할 수 있는 상황.분위기 기반 카페 탐색 서비스

기술 스택

협업 방식

Notion	개발 문서/회의 정리, 팀 내 정보 체계적 관리
Figma	서비스 화면 구조 및 UI 시각적 설계
slack	작업 진행 상황 실시간 공유
git	기능 구현 단위로 커밋하여 변경 이력 관리
GitHub	팀 레포지토리 소스 코드 공유 및 협업 흐름 통합
Swagger, POSTMAN	프론트&백 간 API 명세 문서/요청 및 응답 검증

프론트엔드

NEXT.js	SSR/SSG로 SEO 최적화와 빠른 초기 로딩 속도, App Router 페이지 관리
React	컴포넌트 단위로 화면 구성, UI 재사용성, Hooks API 상태 관리
TypeScript	타입으로 데이터 구조 명확히 관리, 컴파일 단계에서 버그 예방
AXIOS	인터셉터 기능으로 JWT 토큰 자동 추가, 에러 통일 처리, API 통신 관리
tailwindcss	클래스 조합만으로 빠른 스타일링, 일관된 디자인 구현
Websockets Stomp	STOMP 프로토콜로 Spring과 연동, 토픽 기반 메시징으로 실시간 채팅 기능
Zustand	가볍고 직관적인 전역 상태 관리, 컴포넌트 간 데이터 공유, 서버 응답 실시간 반영, UI 상태 즉시 업데이트
Google Maps	풍부한 지도 기능과 위치 정확도 위치 기반 서비스, TypeScript 지원

백엔드

Java 17	안정적, 성능 향상 LTS 버전 서버 사이드 개발에 적합한 주요 언어
spring boot	내장 Tomcat과 의존성 관리로 RESTful API 서버 빠른 구축
Spring Data JPA	(QueryDSL: 타입 안정성, Native Query: 복잡한 쿼리) ORM 기반 데이터 접근 단순화
spring security, JWT	세션 없이 토큰 기반 인증 효율적 인증.인가, 사용자 권한 안전 관리
amazon RDS, MySQL 8.0	AWS RDS(MySQL 8) 클라우드 기반 관리형 데이터베이스 자동 백업/장애 복구 안정적
amazon S3	이미지를 클라우드 객체 스토리지에 저장해 서버 부하 줄이고 확장성 높임
kakaomap	위치 기반 검색/거리 계산 기능 구현 외부 지도 서비스와 연동 강화
python	카카오맵,네이버 블로그 데이터 자동 수집, 수집 정보 MySQL 저장→ 데이터셋 구축
Websockets Stomp	

기술적 의사결정

1. 실시간 통신 기술 스택 결정

결정 배경
실시간 채팅 기능을 구현하기 위해 양방향 통신이 가능한 프로토콜 필요
메시지 구조화, 확장성 요구

선택 사유
Spring 공식 지원으로 설정 및 확장 용이, 메시지 전송 및 구독 관리 직관적
STOMP 프로토콜을 통해 메시지 라우팅(1:1 및 단체 채팅 구분) 효율적 처리
Publish/Subscribe 구조로 실시간 단체 채팅 기능을 안정적 구현

선택
Websockets& STOMP

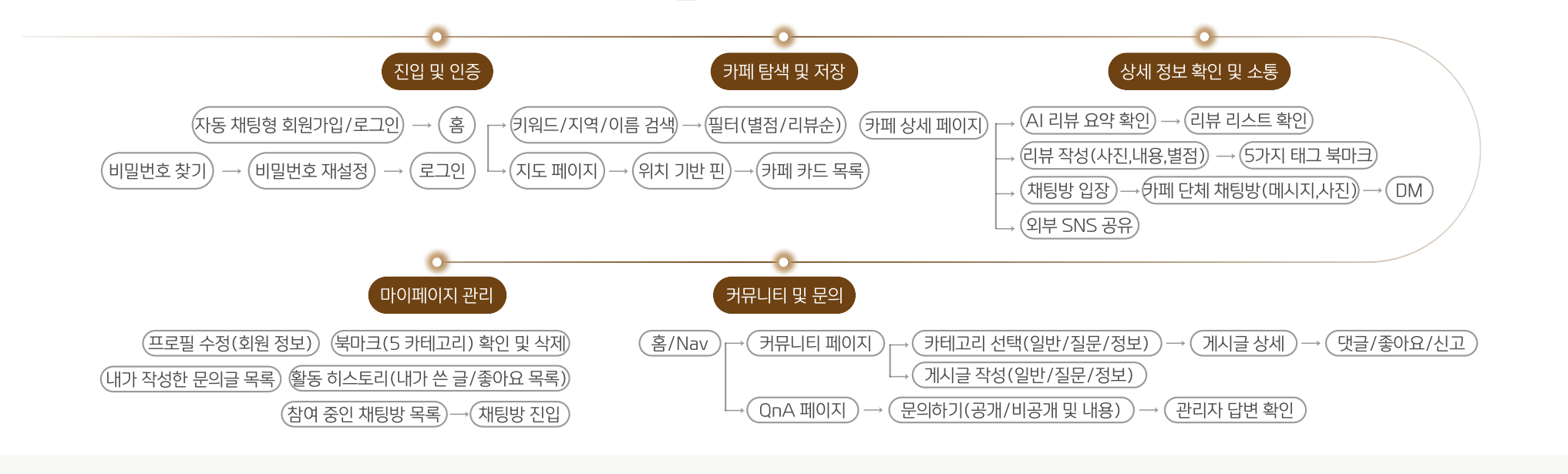
2. 전역 상태 관리 방식 선택

결정 배경
서버와 클라이언트의 상태를 명확히 분리하고 어떤 실행 환경(React 내/외부)에서도 동일한 데이터에 안전하게 접근할 수 있는 아키텍처

선택 사유 (Zustand)
하이브리드 아키텍처: 경량 Zustand 전역 저장소+Context API 인터페이스 레이어
→ 성능/유지보수성 향상
서버/클라이언트 동기화: React Query+Zustand Persist 미들웨어
→ 새로고침 시에도 데이터 무결성 유지/서버 상태 안정적 동기화
환경 제약 극대화: Zustand 클로저 기반 스토어
→ 리액트 훅 제약 없이 Axios 인터셉터 등 외부 로직에서도 전역 상태에 직접 접근하도록 설계

선택
도구별 강점을 극대화하기 위해 서버 데이터-로컬 상태-비즈니스 로직을 분리한 하이브리드 아키텍처를 설계하여 유연성과 안정성을 동시에 확보했다.
서버(DB) → **React Query**(가져오기) → **Context**(가공/명령) → **Zustand**(저장/유지) → UI(컴포넌트)

유저 플로우 (User Flow)



진입 및 인증
자동 채팅형 회원가입/로그인 → 홈
비밀번호 찾기 → 비밀번호 재설정 → 로그인

카페 탐색 및 저장
키워드/지역/이름 검색 → 필터(별점/리뷰순) → 카페 상세 페이지
지도 페이지 → 위치 기반 핀 → 카페 카드 목록

상세 정보 확인 및 소통
AI 리뷰 요약 확인 → 리뷰 리스트 확인
리뷰 작성(사진,내용,별점) → 5가지 태그 북마크
채팅방 입장 → 카페 단체 채팅방(메시지,사진) → DM
외부 SNS 공유

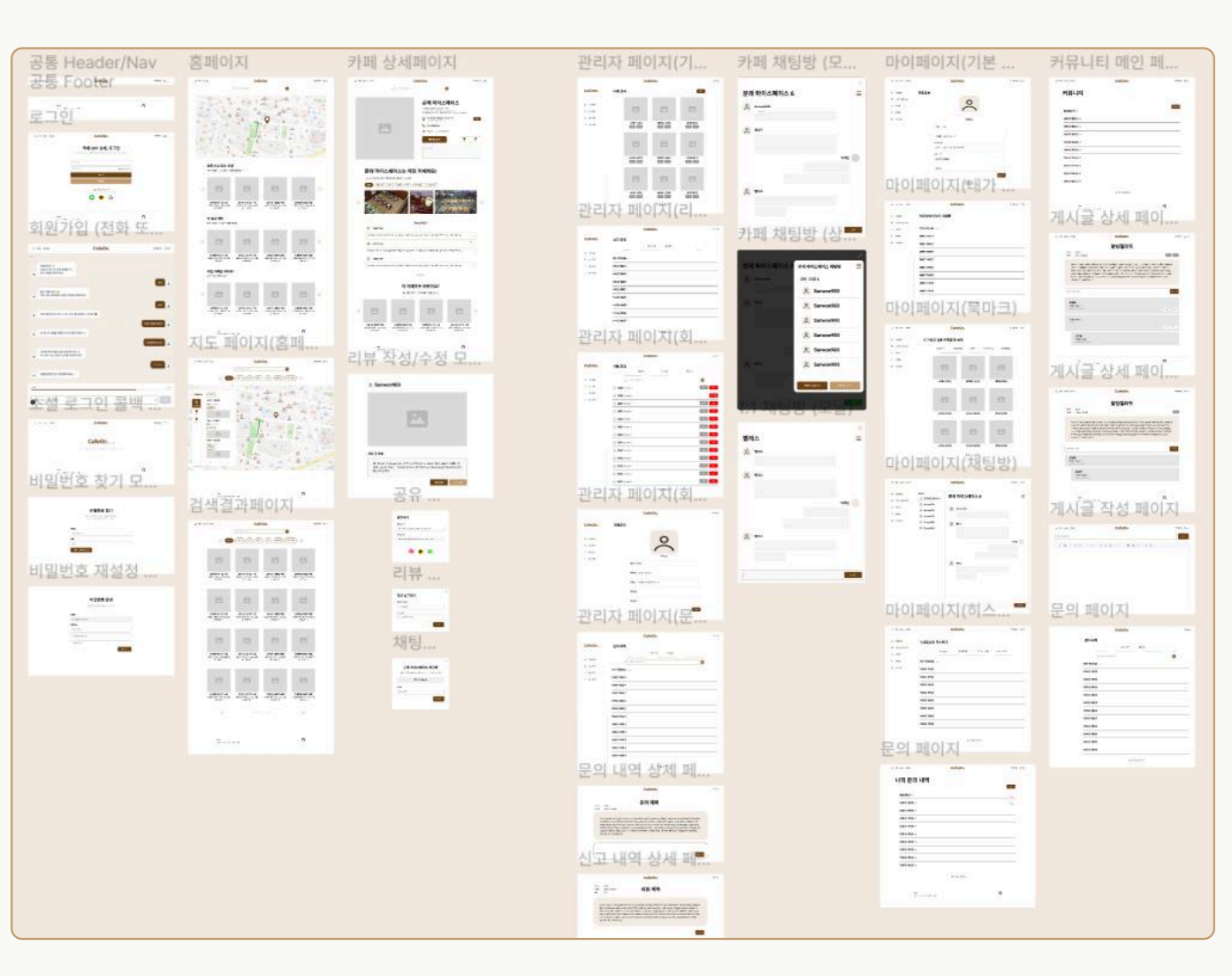
마이페이지 관리
프로필 수정(회원 정보)
북마크(5 카테고리) 확인 및 삭제
내가 작성한 문의글 목록
활동 히스토리(내가 쓴 글/좋아요 목록)
참여 중인 채팅방 목록 → 채팅방 진입

커뮤니티 및 문의
홈/Nav → 커뮤니티 페이지 → 카테고리 선택(일반/질문/정보) → 게시글 상세 → 댓글/좋아요/신고
커뮤니티 페이지 → 게시글 작성(일반/질문/정보)
홈/Nav → QnA 페이지 → 문의하기(공개/비공개 및 내용) → 관리자 답변 확인

기능 명세서

사용자 중심 기능 확장:
단순 카페 정보 제공을 넘어, AI 리뷰 요약 및 실시간 채팅을 통한 '커뮤니티형 카페 탐색' 기능 설계
상태 기반 UI 기획: 로그인 여부, 권한(일반/관리자)에 따른 메뉴 노출 및 접근 제어 로직 상세화.

페이지	주요 페이지/모듈	기능 상세
인증/계정	회원가입	자동 채팅 형식 UI, 입력 유효성 검사, 에러 처리
	로그인	기본 형식 (ID/PW), 소셜 로그인 콜백 처리 화면 (로딩 후 리다이렉트)
	비밀번호 찾기	이메일 입력 UI
	비밀번호 재설정	새 비밀번호 입력 UI
마이페이지	마이페이지 메인	프로필, 탭/사이드바 내비게이션 UI
	회원정보 관리	정보 확인 및 수정 폼, 비밀번호 재설정
	회원 탈퇴	확인 모달/페이지 (Soft Delete 연동)
	내가 쓴 리뷰	작성/수정/삭제 기능 제공 리스트
	북마크 목록	5가지 옵션별 (나만의 아지트 등) 탭/필터링 UI, 삭제 기능
	참여 중인 채팅방	참여 목록 리스트
	나의 문의 내역	문의 글 수정 및 삭제 기능 제공
	사용자 히스토리	활동 내역 (내가 남긴 글/댓글, 좋아요 목록) 리스트
탐색	홈페이지	상단 검색바, 추천 카페 섹션 (요즘 뜨는, 찜 많은, 신상/가오픈) 컴포넌트
	검색 결과 페이지	검색바, 필터/정렬 영역 UI, 카페 리스트 (리뷰/별점 필터 추가)
	지도 페이지	사용자 위치 기반 지도 연동 (핀, 클러스터링), 사이드 카페 목록, 카테고리별 추천 리스트
카페 상세	기본 정보 표시	이름, 위치(지도 연동), 영업시간, 연락처, 사진
	AI 리뷰 요약	네이버 리뷰 요약 결과 표시 컴포넌트
	리뷰 목록	최신순/별점 순 정렬, 목록 UI
	리뷰 작성/수정	모달 UI
	북마크 저장	5가지 옵션 선택 UI
	소셜 기능	채팅방 공유, 외부 SNS 공유 버튼
	추천 카페	관련 태그 및 요즘 뜨는 카페 목록 컴포넌트
채팅	채팅방 목록	카페 그룹/DM 구분 리스트
	채팅방 상세	WebSocket 기반 실시간 채팅 UI (텍스트/사진), 날짜/시간/안 읽음 수 표시
	채팅방 기능	참여자 목록/알림 끄기/나가기 기능 UI
커뮤니티	커뮤니티 메인	카테고리 (일반/질문/정보) 필터링, 게시글 리스트
	게시글 상세	본문, 좋아요, 댓글/대댓글 UI
	게시글 작성/수정	에디터 UI
	신고하기	신고 옵션 선택 UI
QnA	문의 내역	전체 문의 목록, 공개/비공개 문의 작성/검색, 관리자 답변 표시
관리자	카페 관리	카페 CRUD 기능 목록 및 폼 UI
	회원 관리	전체/정상/정지 회원 목록 및 분류 UI
	신고 관리	미처리/처리 내역 탭, 상세 보기 및 처리 기능
	문의 내역	미처리/처리 내역 탭, 상세 보기 및 답변 작성 기능



화면 설계서

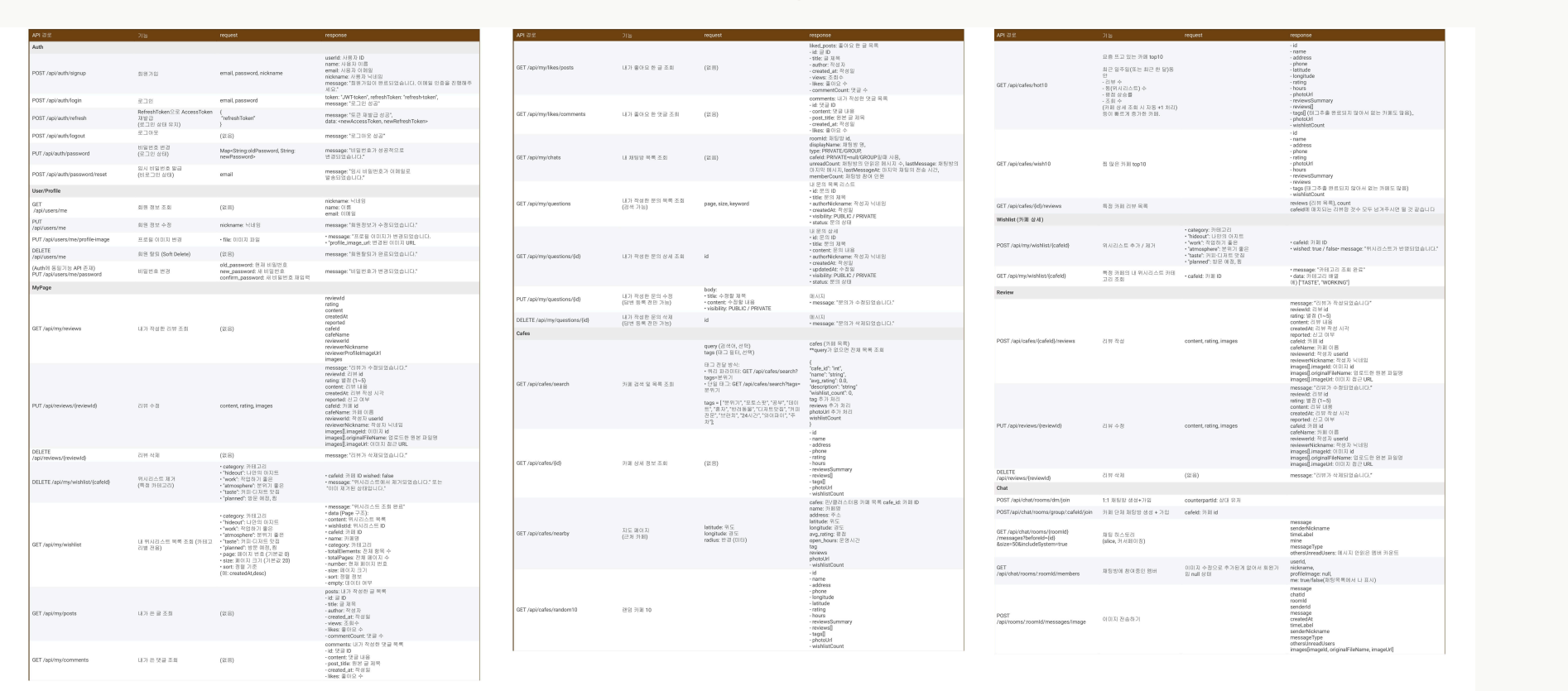
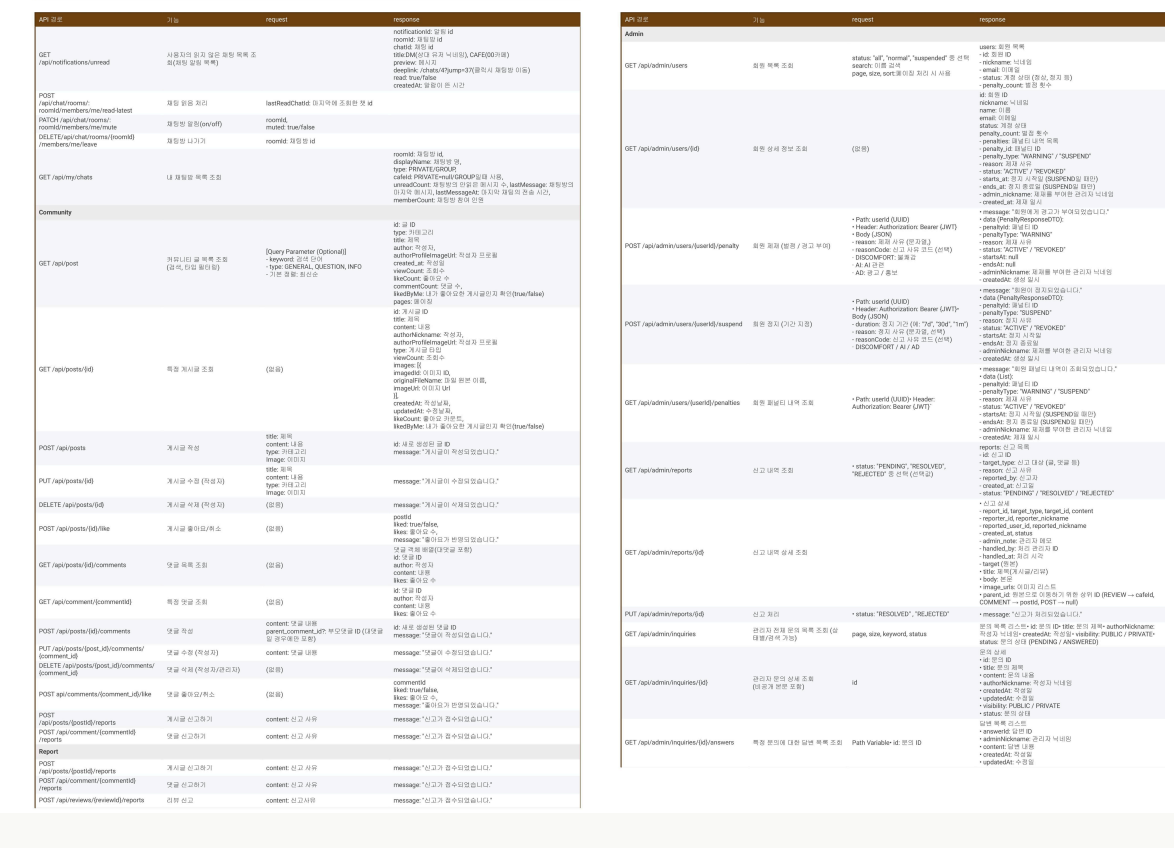
대화형 회원가입 UI
사용자가 우리 서비스에서 목표지점까지 도달하게 만드는 첫번째 단계인 회원가입을 단순 폼이 아닌 대화형 설계로 신규 유저의 심리적 진입 장벽을 낮춤

실시간 현장감 구현
WebSocket을 연동한 채팅 UI로 '현재 카페 혼잡도'나 '메뉴 정보' 등 휘발성 강한 정보를 유저 간 실시간으로 교류하도록 유도

탐색 효율 극대화
-'작업하기 좋은', '디저트 맛집' 등 5가지 북마크 시스템을 설계하여 유저의 재방문 유도
-방대한 리뷰 데이터를 직접 읽지 않아도 AI 요약 정보를 상세 페이지 최상단에 배치하여 정보 습득 속도 개선

반응형 고려
다양한 디바이스 환경을 고려하여 그리드 시스템 기반의 레이아웃 배치 및 컴포넌트 단위 설계

API 명세서

핵심 API	특징 및 적용 기술
GET /cafes/search	서비스 진입점 및 핵심 탐색 기능 다중 쿼리 파라미터 활용 〉 필터링 확장성
GET /cafes/{id}	페이지 렌더링의 기반 데이터 데이터 로딩 상태(Loading/Error) 전역 관리
POST /chat/join	실시간 인터랙션의 시작점 Deadlock 대응 자동 재시도 로직
POST /chat/messages	가장 높은 빈도의 사용자 상호작용 사용자 의사결정의 핵심 데이터
GET /cafes/{id}/reviews	지수 백오프 기반 안정성 확보 대용량 텍스트 데이터의 비동기 스트리밍 및 가공

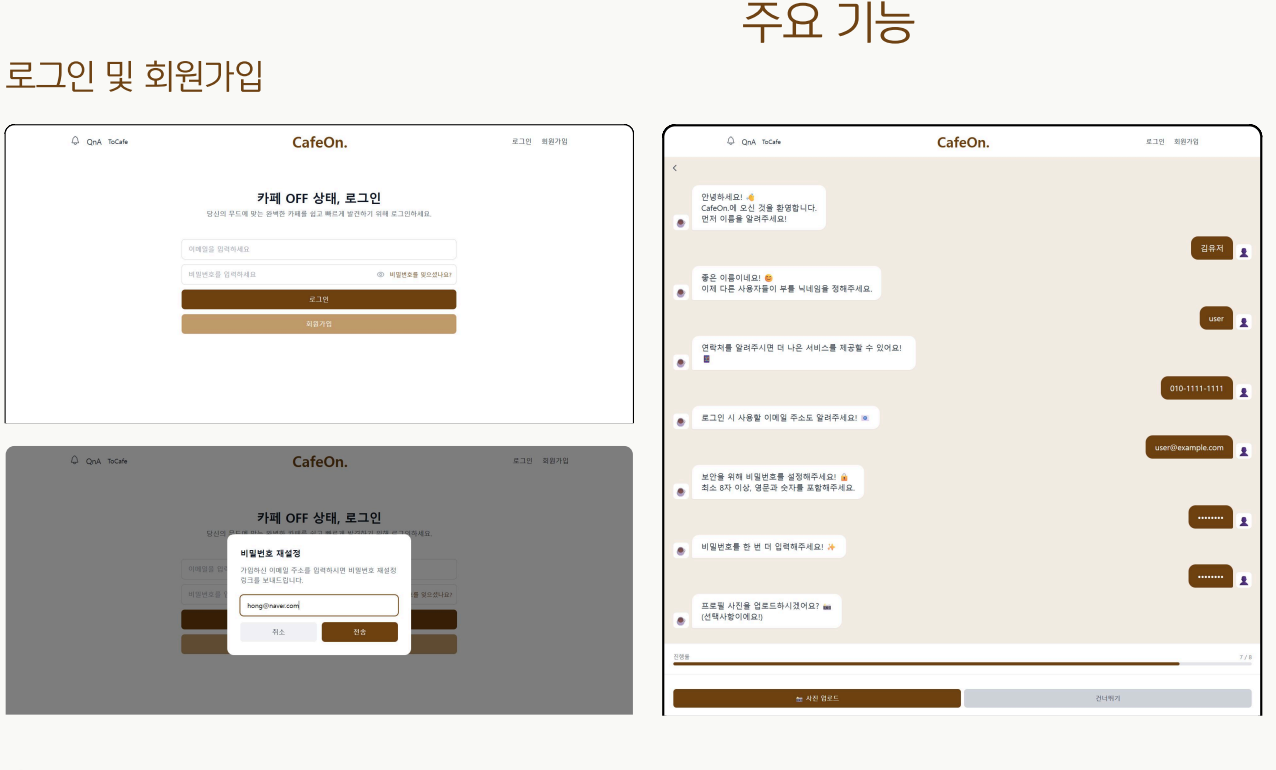
핵심 라우팅 명세서

구분	경로	페이지	권한	비즈니스 로직 및 핵심 기술
탐색	/	홈페이지	Public	위치 기반 카페 추천 및 메인 큐레이션
탐색	/map	지도 탐색	Public	지도 API 클러스터링 및 실시간 필터링
상세	/cafes/:cafeId	카페 상세	Public	AI 리뷰 요약 연동 및 북마크 로직
채팅	/cafes/:cafeId	실시간 채팅	Auth	WebSocket 기반 실시간 참여자 동기화
커뮤니티	/community	커뮤니티	Public	게시글 CRUD 및 카테고리별 동적 렌더링
계정	/mypage	마이페이지	Auth	Zustand(Persist) 기반 사용자 활동 데이터 관리
문의	/qna	문의 게시판	Public/ Auth	전체 목록 조회 및 개인별 1:1 문의 작성/상세
관리	/admin/*	관리자 도메인	Admin	RBAC(역할 기반 제어) 적용, 서비스 리소스 관리

* 카페 상세 페이지 내에서 React 상태 기반의 모달 컴포넌트로 채팅 기능을 구현하여, 별도의 페이지 이동 없이 즉시 채팅방에 접근할 수 있도록 설계

주요 기능

로그인 및 회원가입



기능 개요

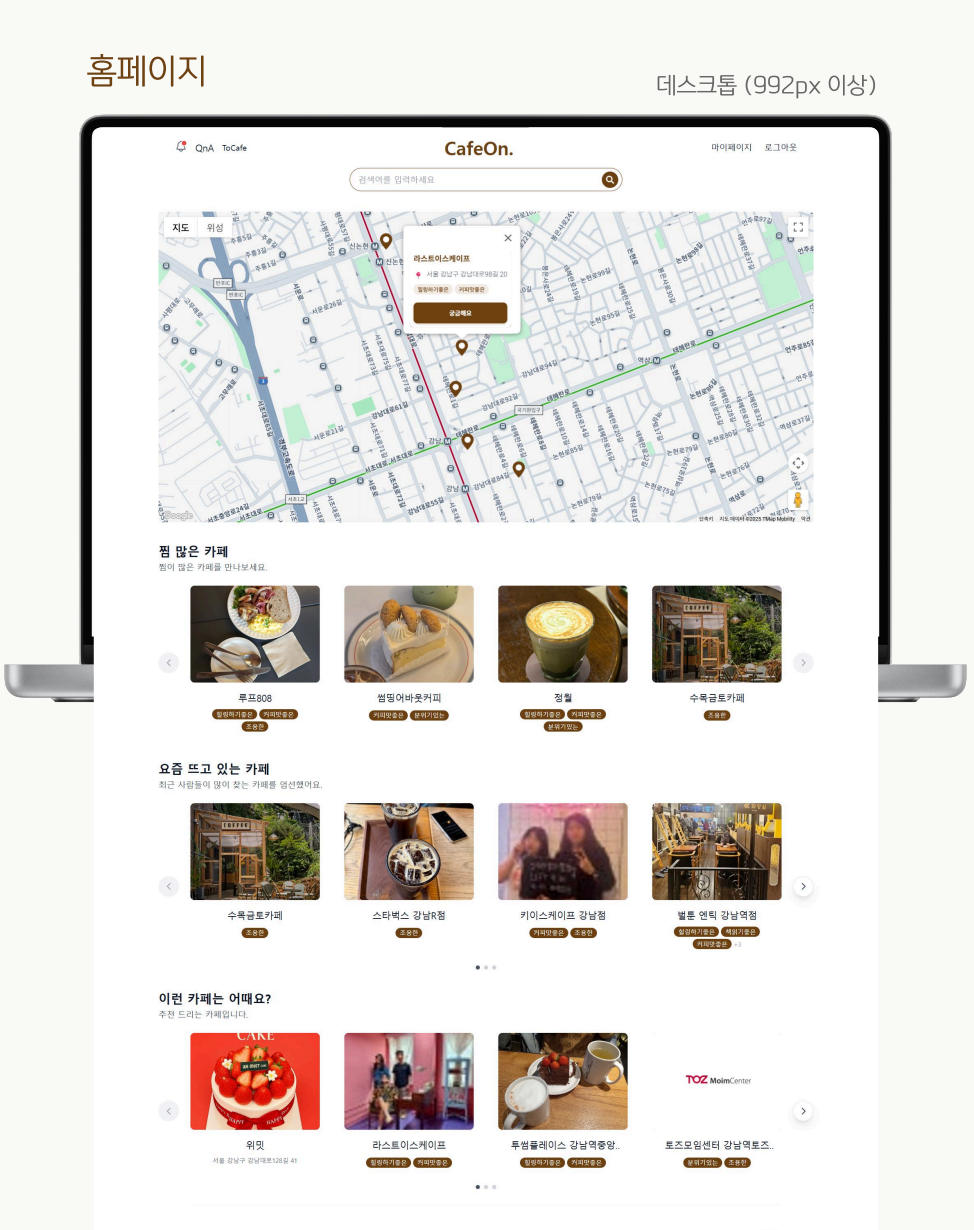
회원가입
이메일·비밀번호·이름·닉네임·연락처를 입력해 /api/auth/signup으로 가입하고, 에러는 normalizeError로 처리해 화면에 통일된 메시지로 보여준다.

로그인
이메일·비밀번호로 /api/auth/login 호출 후 전달받은 JWT(access/refresh)를 Zustand persist로 localStorage에 저장하고, 로그인 상태·사용자 정보를 전역에서 사용한다.

구현 포인트
Zustand persist로 토큰·사용자 정보를 localStorage에 보관하고, axios 인터셉터에서 Bearer 토큰 부착·401/403 시 refresh 토큰 자동 갱신 후 재요청을 처리한다.

홈페이지

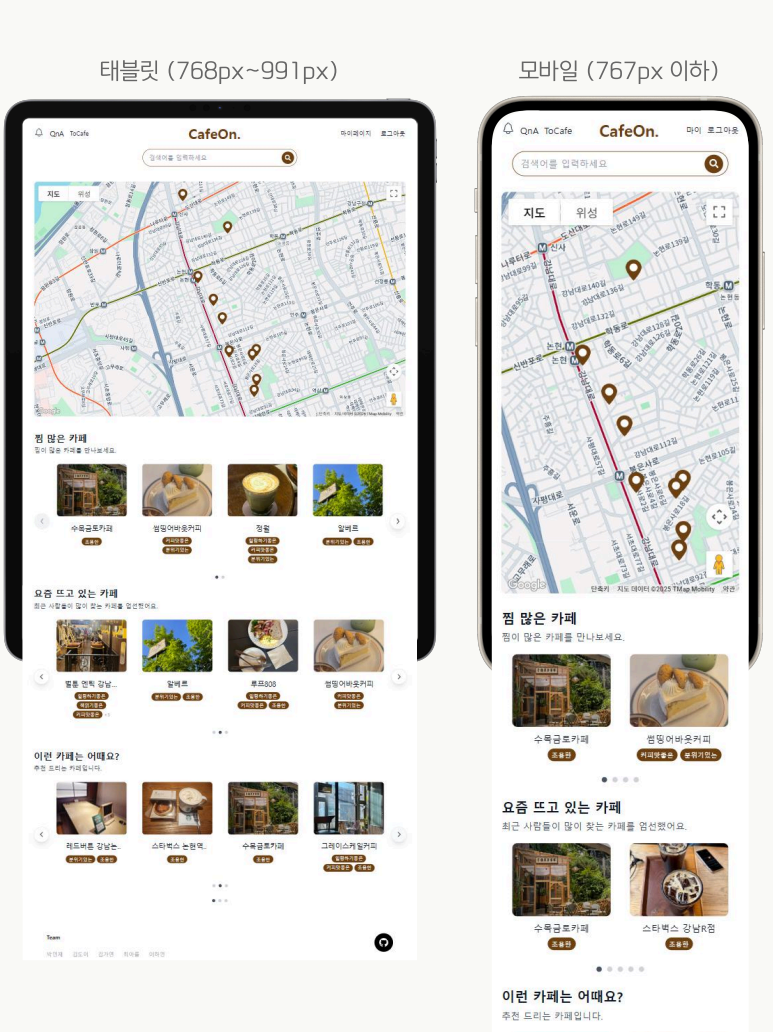
데스크톱 (992px 이상)



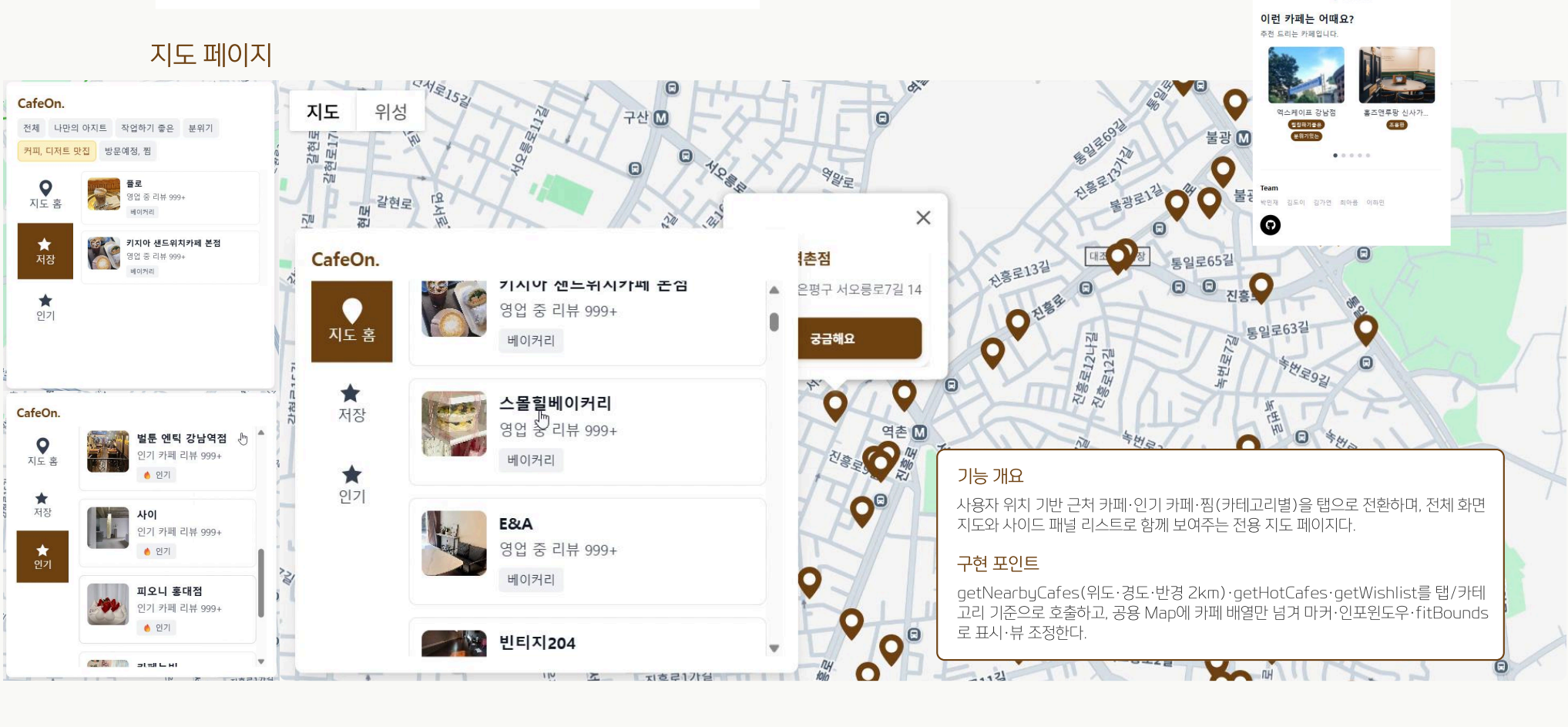
기능 개요
검색 이동용 SearchBar, 서울 구 단위 랜덤 중심 지도, "찜 많은 / 요즘 뜨는 / 랜덤 추천" 카페 캐러셀 세 섹션으로 메인을 구성한다.

구현 포인트
random10·hot10·wish10 API를 Promise.all로 병렬 조회하고, Google Maps + seoulDistricts로 지도 중심과 마커를 두며, CafeCarousel은 브라우저 너비에 따라 2/3/4열로 나누는 반응형 페이지네이션으로 렌더한다.

태블릿 (768px~991px) 모바일 (767px 이하)

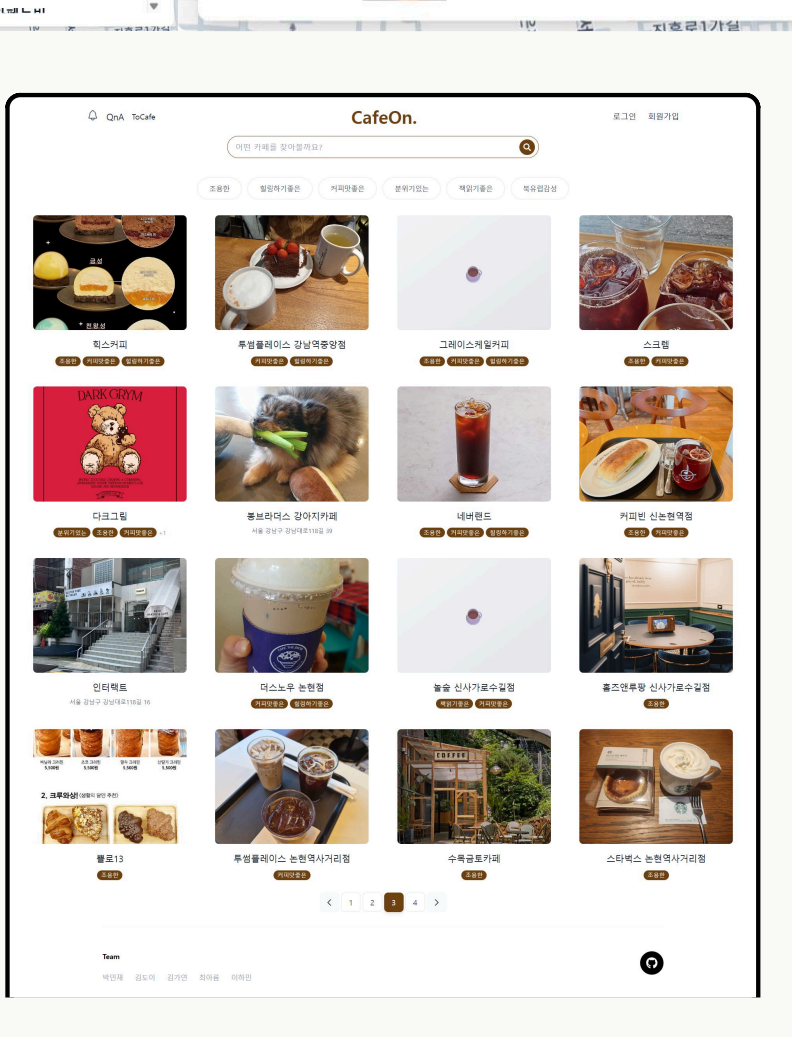


지도 페이지



기능 개요
사용자 위치 기반 근처 카페·인기 카페·찜(카테고리별)을 탭으로 전환하며, 전체 화면 지도와 사이드 패널 리스트로 함께 보여주는 전용 지도 페이지이다.

구현 포인트
getNearbyCafes(위도·경도·반경 2km)·getHotCafes·getWishlist를 탭/카테고리 기준으로 호출하고, 공용 Map에 카페 배열만 넘겨 마커·인포윈도우·fitBounds로 표시·뷰 조정한다.



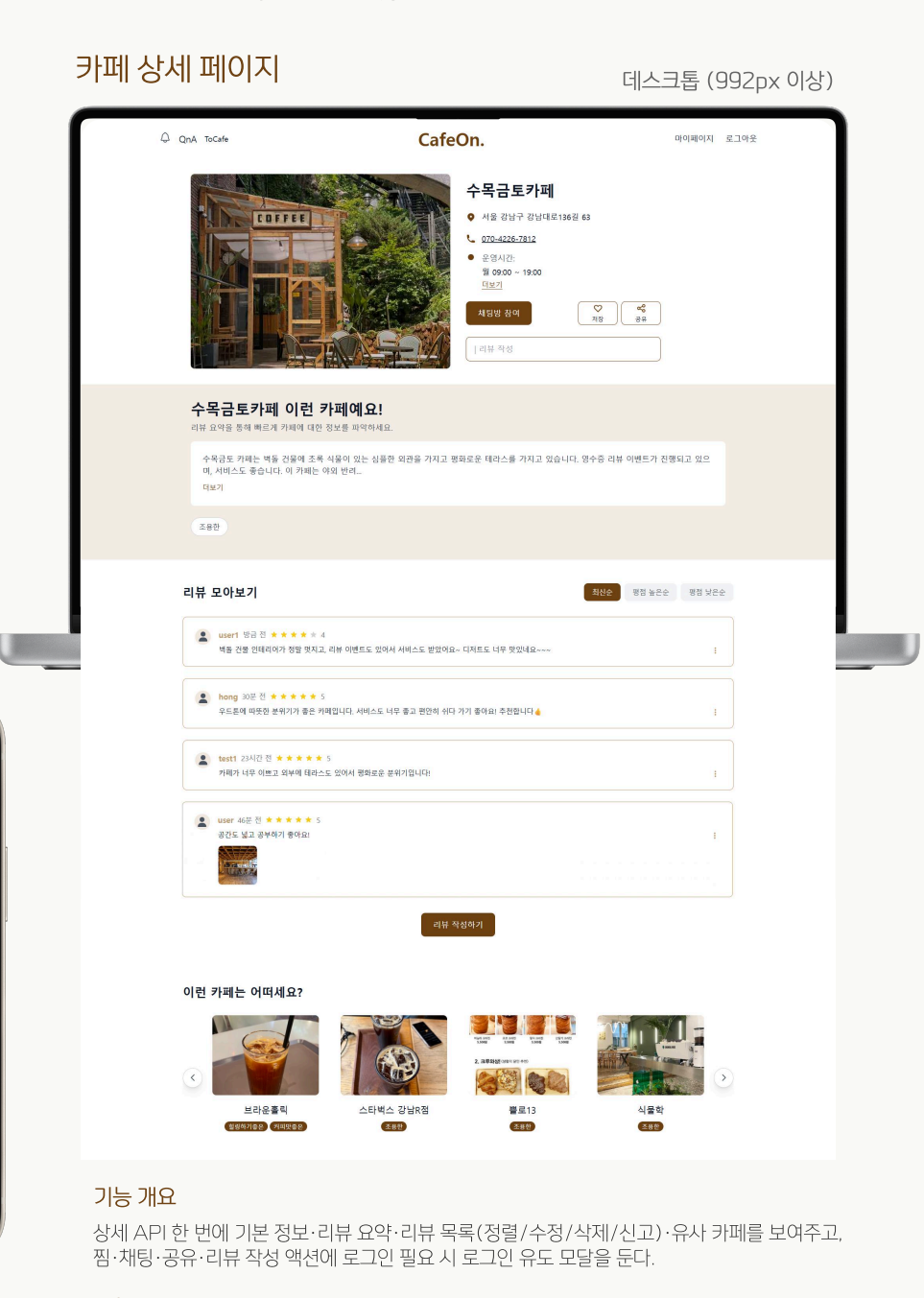
검색 페이지

기능 개요
URL·검색창 검색어와 태그 필터(결과 기반 상위 10개)로 카페 검색 API를 호출하고, 이미지 있는 카페만 반응형 그리드·클라이언트 페이지네이션으로 보여준다.

구현 포인트
searchCafes(query, tags)로 검색/필터하고, 결과에서 태그 빈도 상위 10개를 CategoryFilter에 넘겨 태그 필터로 재검색한다. 반응형 itemsPerPage + slice 페이지네이션과 페이지 전환 시 상단 스크롤을 적용했다.

카페 상세 페이지

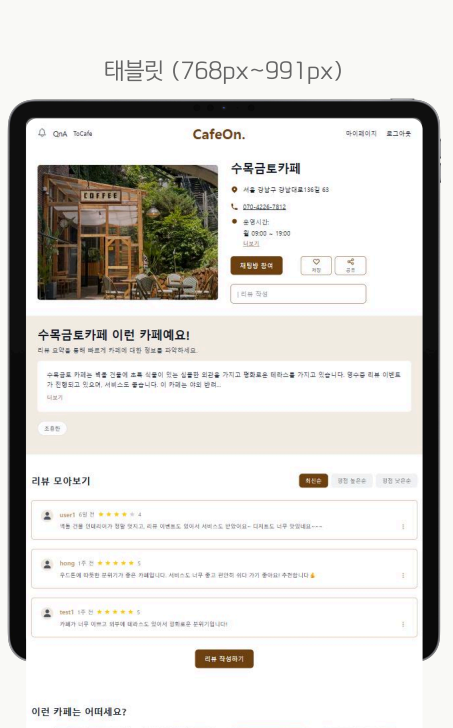
데스크톱 (992px 이상)



기능 개요
상세 API 한 번에 기본 정보·리뷰 요약·리뷰 목록(정렬/수정/삭제/신고)·유사 카페를 보여주고, 찜·채팅·공유·리뷰 작성 액션에 로그인 필요 시 로그인 유도 모달을 띄운다.

구현 포인트
getCafeDetail 응답을 mapApiToCafeDetail로 변환해 네 개 섹션에 나누어 넘기고, 리뷰는 initialReviews/getCafeReviews·refreshTrigger·정렬로 관리한다. 찜은 getWishlistCategories·WishlistModal, 채팅/리뷰는 인증 확인 후 모달 분기·useEscapeKey로 모달 닫기 기능을 두었다.

태블릿 (768px~991px) 모바일 (767px 이하)

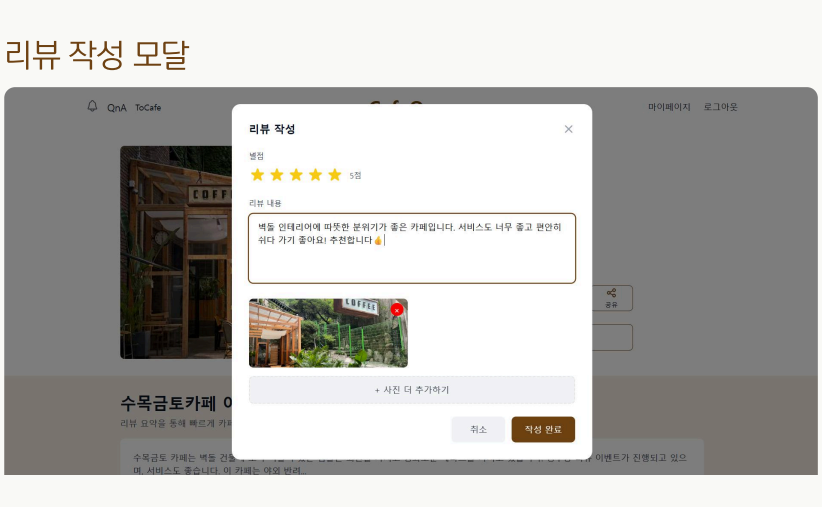


반응형 웹 구현

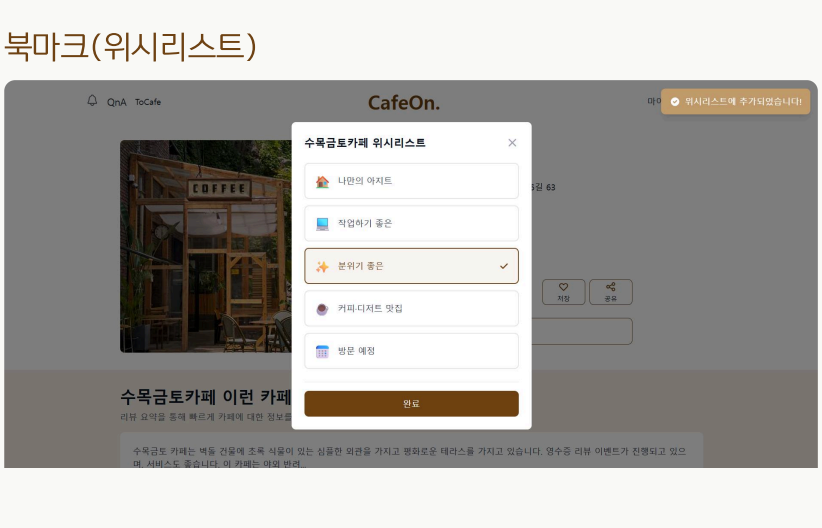
기능 개요
Tailwind sm/md/lg로 패딩·타이포·간격·버튼 크기를 조절하고, 정보 섹션은 lg에서 2열 그리드, 유사 카페는 640px 기준 1/4칸 전환, 리뷰 헤더·정렬은 모바일 세로·sm 이상 가로로 배치한다.

구현 포인트
grid-cols-1 lg:grid-cols-2·px-4 sm:px-6 md:px-8·text-* sm:text-*로 레이아웃·폰트 단계 적용. 유사 카페만 window.innerWidth와 resize로 itemsPerPage를 1/4로 바꾸고, 리뷰 정렬 영역은 flex-col sm:flex-row·flex-1 sm:flex-none·text-xs sm:text-sm으로 넓이·글자 크기를 반응형 처리했다.

리뷰 작성 모달



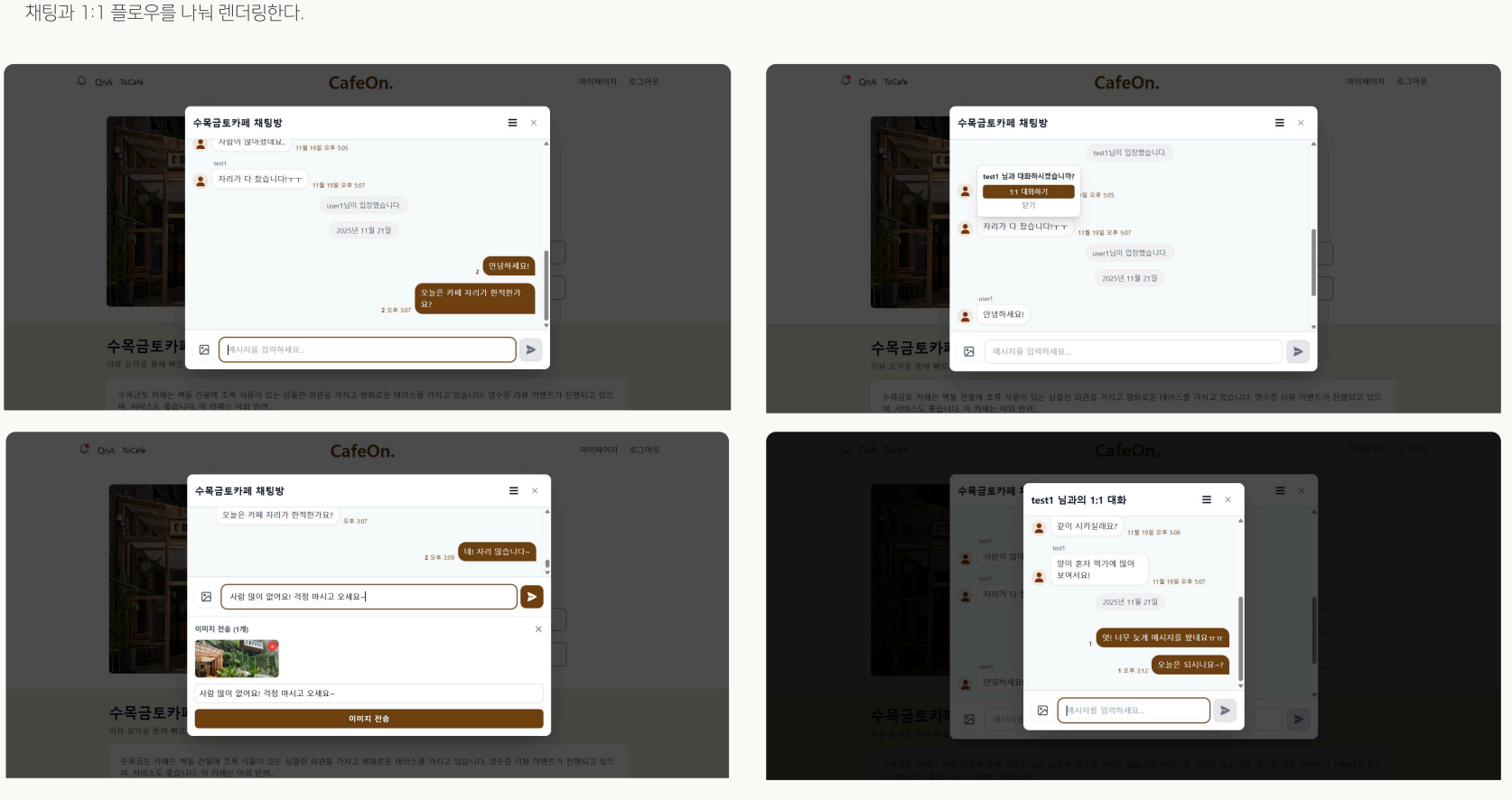
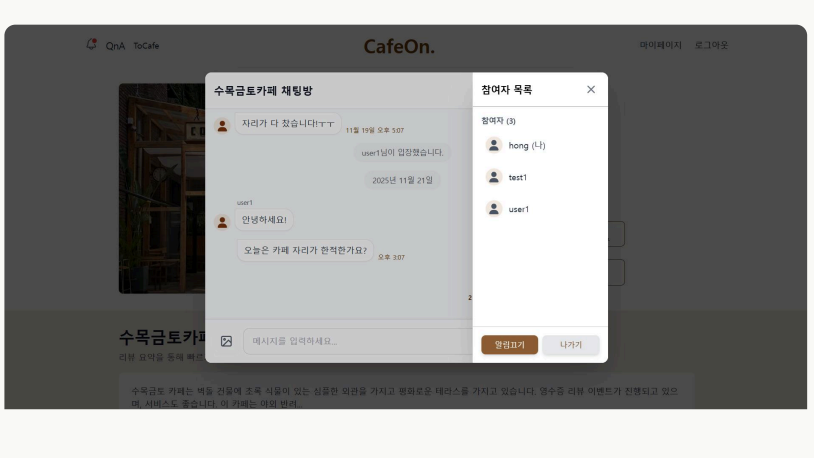
북마크(위시리스트)



카페 단체 채팅방

기능 개요
카페 상세의 채팅방 참여로 해당 카페 단체 채팅에 입장해, REST+STOMP로 실시간 메시지·참여자 목록·과거 대화·전송 메시지 시간·읽음·나가기/알림을 지원하고, 참여자 클릭 시 1:1 채팅으로 이어진다.

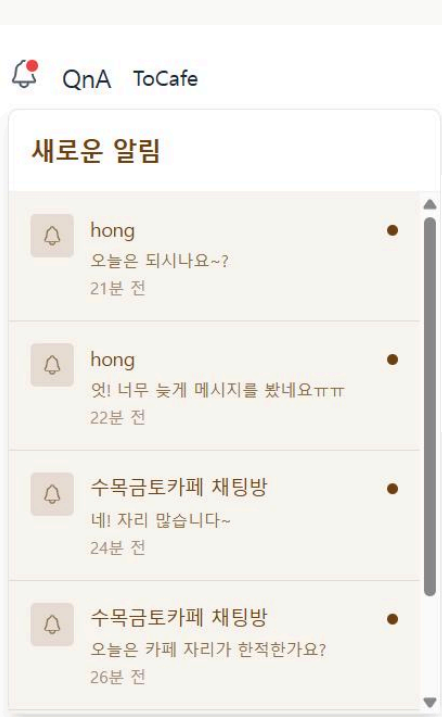
구현 포인트
joinCafeGroupChat으로 roomId 확보 후 chatStore 기반 useCafeChat에서 STOMP 구독·메시지/참여자/읽음/뮤트 훅을 묶고, CafeChatModal이 입장 한 번만 실행한 뒤 ChatMessageList·Input·ChatSidebar·ProfileMiniPopup·PrivateChatModal로 채팅과 1:1 플로우를 나눠 렌더링한다.

1:1 채팅방 (DM)

기능 개요
단체 채팅 참여자 프로필 → "1:1 대화하기"로 PrivateChatModal을 열어, 상대와 1:1 방을 생성·입장하거나 기존 방을 불러와 실시간 메시지·히스토리·알림·나가기를 지원하며, 마이페이지에서는 existingRoomId로 기존 DM 방 진입을 지원한다.

구현 포인트
createDmChat(counterpartId)로 POST /api/chat/rooms/dm/join 호출 후 roomId 확보하고, Duplicate 시 roomId 추출·dmChatMapping 저장으로 재입장 처리. useDmChat은 dmSessions[sessionKey]에 방별 상태를 두고 useStompConnection·useChatMessages 등과 연동하며, usePrivateChatFlow가 프로필 클릭→팝업→"1:1 대화하기"→PrivateChatModal(targetUser) 플로우를 담당한다.

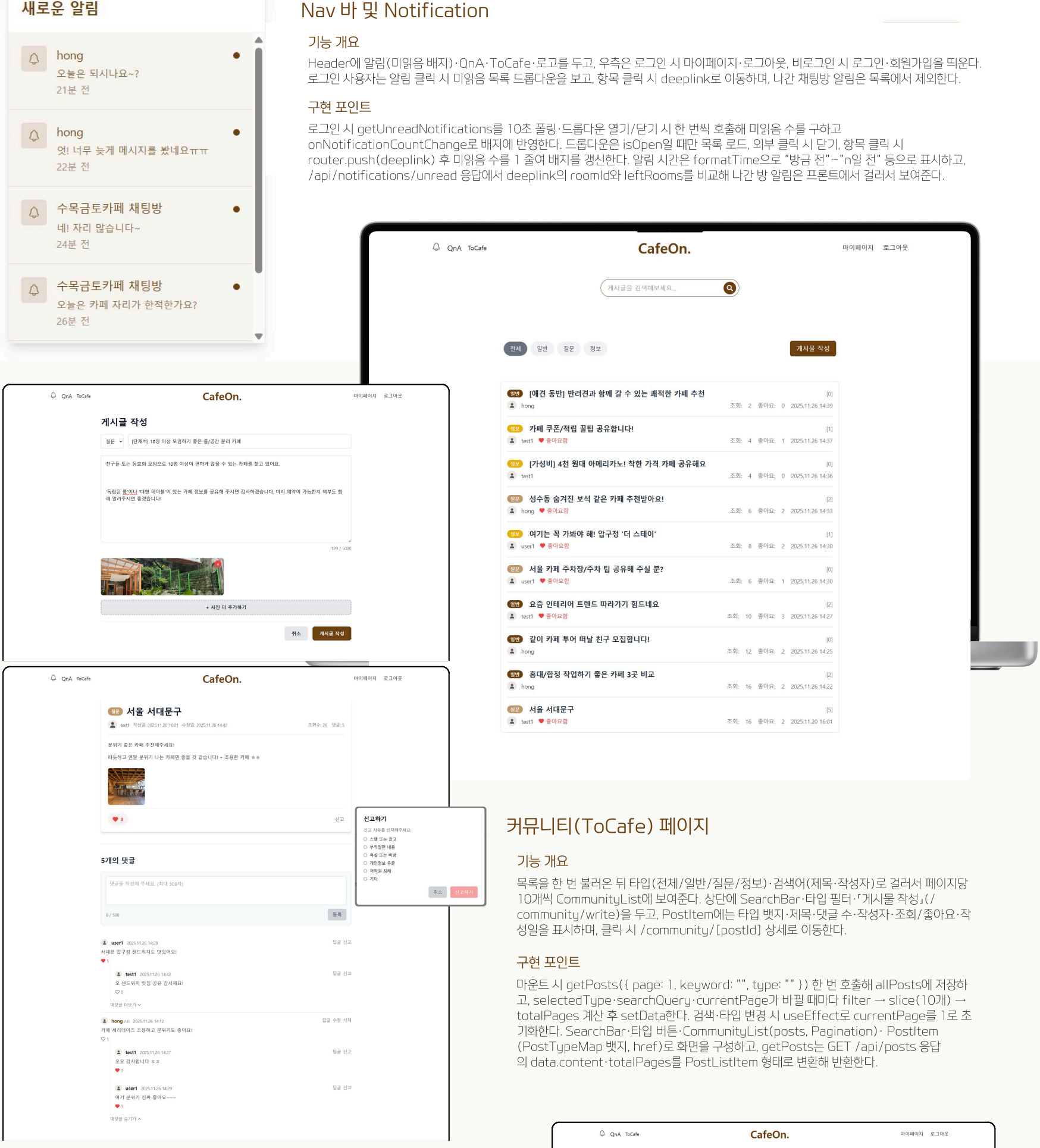


CafeOn. 마이페이지 로그아웃

Nav 바 및 Notification

기능 개요
Header에 알림(미읽음 배지)·QnA·ToCafe·로고를 두고, 우측은 로그인 시 마이페이지·로그아웃, 비로그인 시 로그인·회원가입을 띄운다. 로그인 사용자는 알림 클릭 시 미읽음 목록 드롭다운을 보고, 항목 클릭 시 deeplink로 이동하며, 나간 채팅방 알림은 목록에서 제외한다.

구현 포인트
로그인 시 getUnreadNotifications를 10초 폴링·드롭다운 열기/닫기 시 한 번씩 호출해 미읽음 수를 구하고 onNotificationCountChange로 배지에 반영한다. 드롭다운은 isOpen일 때만 목록 로드, 외부 클릭 시 닫기, 항목 클릭 시 router.push(deeplink) 후 미읽음 수를 1 줄여 배지를 갱신한다. 알림 시간은 formatTime으로 "방금 전"~"n일 전" 등으로 표시하고, /api/notifications/unread 응답에서 deeplink의 roomId와 leftRooms를 비교해 나간 방 알림은 프론트에서 걸러서 보여준다.

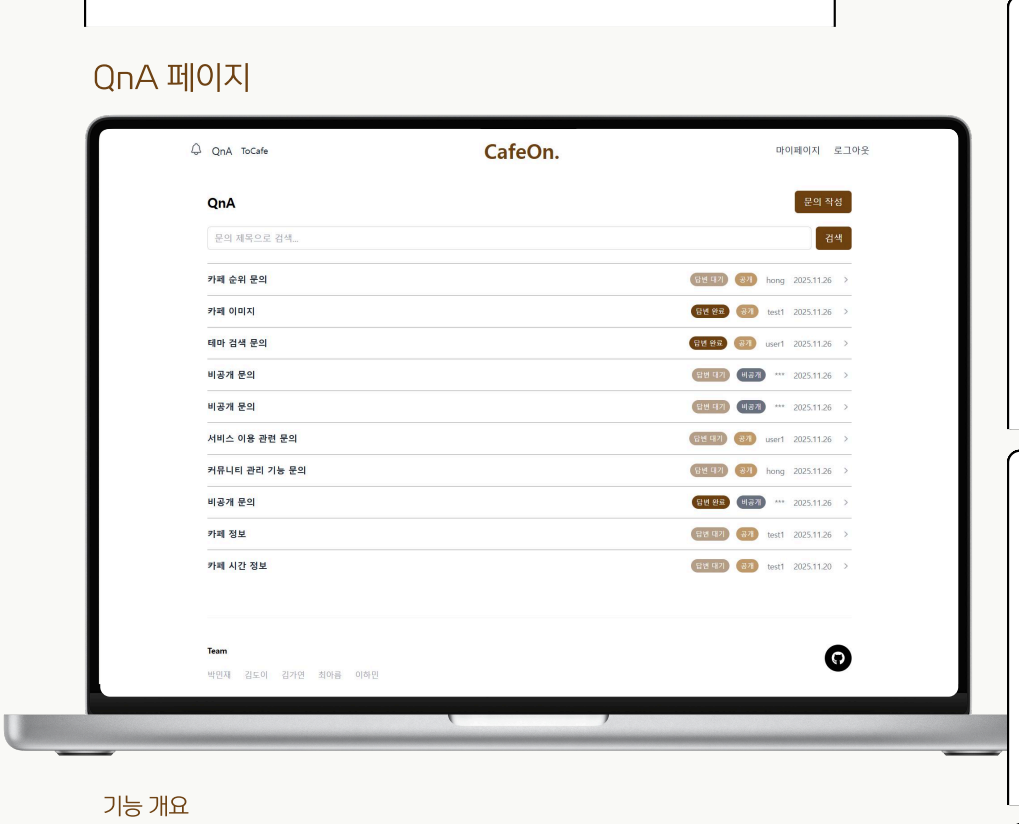
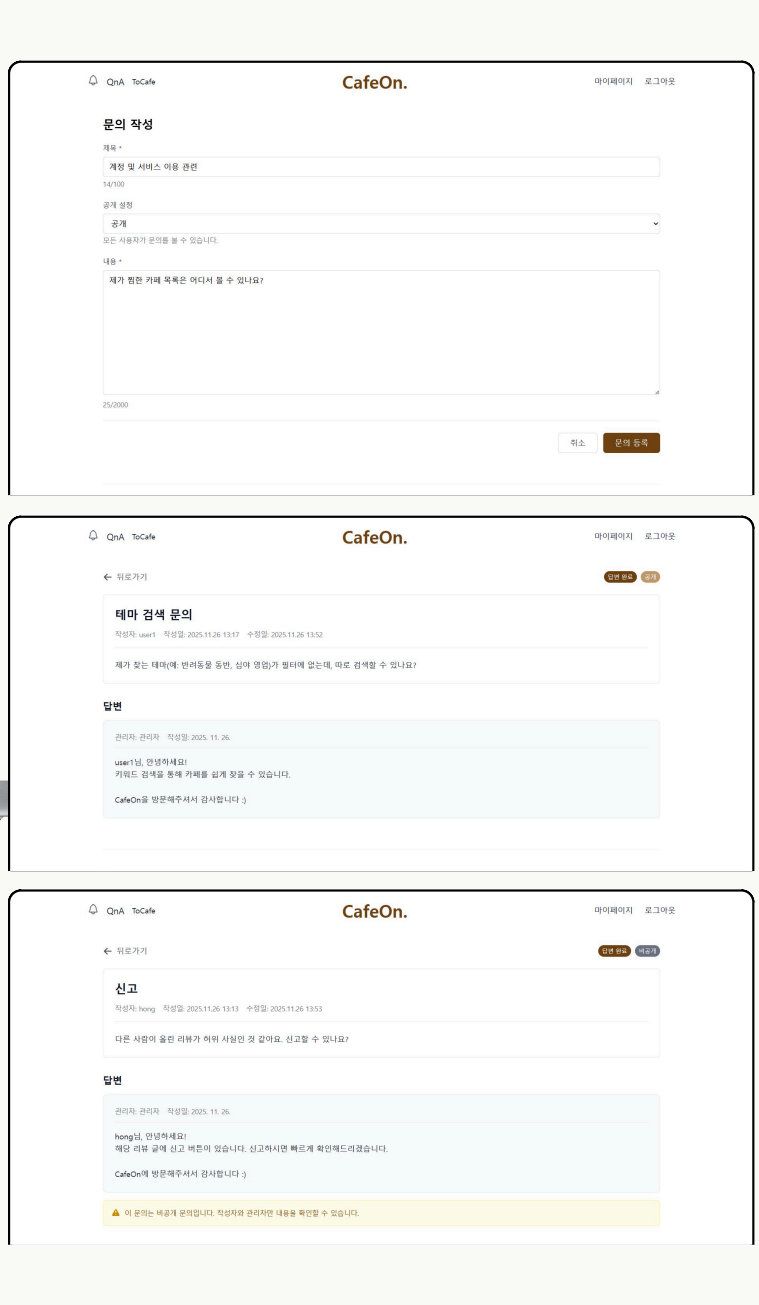


커뮤니티(ToCafe) 페이지

기능 개요
목록을 한 번 불러온 뒤 타입(전체/일반/질문/정보)·검색어(제목·작성자)로 걸러서 페이지당 10개씩 CommunityList에 보여준다. 상단에 SearchBar·타입 필터·'게시글 작성'(/community/write)을 두고, PostItem에는 타입 뱃지·제목·댓글 수·작성자·조회/좋아요·작성일을 표시하며, 클릭 시 /community/[postId] 상세로 이동한다.

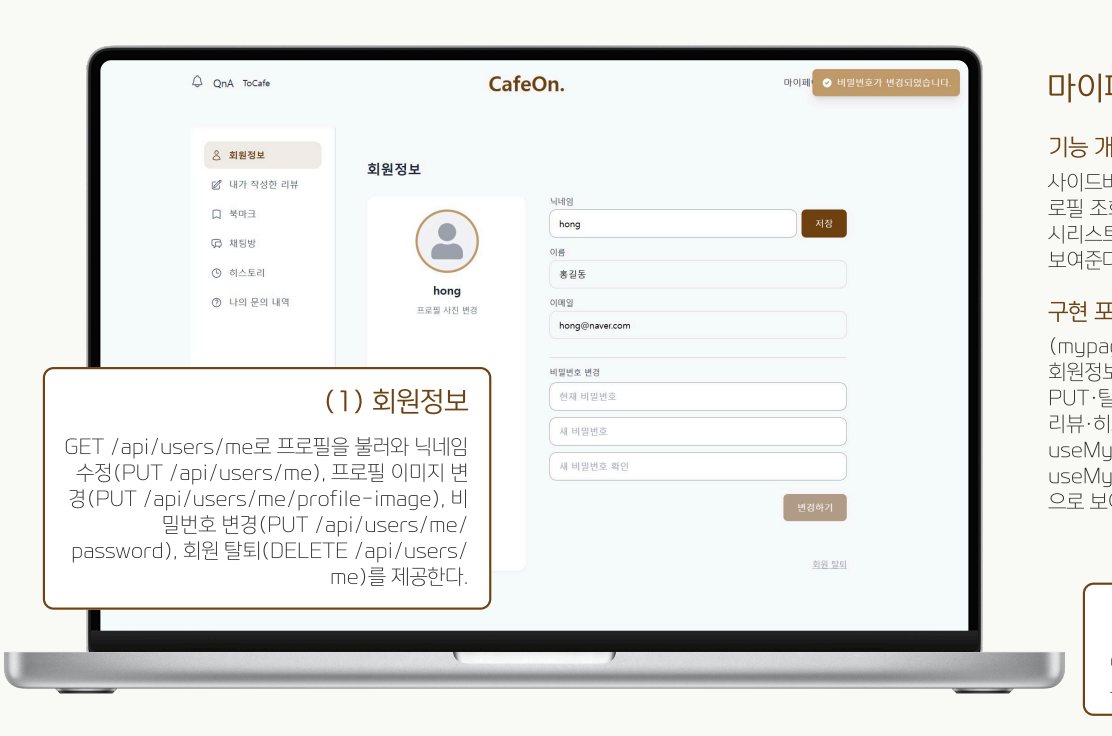
구현 포인트
마운트 시 getPosts({ page: 1, keyword: "", type: "" }) 한 번 호출해 allPosts에 저장하고, selectedType·searchQuery·currentPage가 바뀔 때마다 filter → slice(10개) → totalPages 계산 후 setData한다. 검색·타입 변경 시 useEffect로 currentPage를 1로 초기화한다. SearchBar·타입 버튼·CommunityList(posts, Pagination)·PostItem(PostTypeMap 뱃지, href)로 화면을 구성하고, getPosts는 GET /api/posts 응답의 data.content·totalPages를 PostListItem 형태로 변환해 반환한다.

QnA 페이지

기능 개요
목록(/qna)에서는 검색·문의 작성 버튼과 QuestionList(제목·답변 상태·공개/비공개·작성자·작성일)를 두고, 항목 클릭 시 /qna/[id] 상세로 이동한다. 비공개 문의는 목록에서 제목·작성자만 마스킹하고, 각 항목의 답변 상태는 배지로 표시한다. 작성(/qna/create)에서는 제목·내용·공개/비공개를 입력해 등록한 뒤, 성공 시 해당 문의 상세(/qna/[id])로 이동한다.

구현 포인트
목록 :useQuestionList(page, size, keyword)로 GET /api/qna/questions 호출 → PRIVATE은 제목·작성자 마스킹 후 setQuestions·setTotalPages.
상태 배지 :QuestionList에서 항목마다 fetchQuestionStatus(id)로 GET .../questions/{id} 호출 → questionStatuses[id]로 배지 표시.
상세 :useQuestionDetail(id)·useAnswerList(questionId)로 문의·답변 로드. useAnswerList는 공개이거나 (비공개+본인)이고 status === ANSWERED일 때만 호출.
작성 :createQuestion으로 POST /api/qna/questions → 반환 id로 router.push(/qna/${id}).



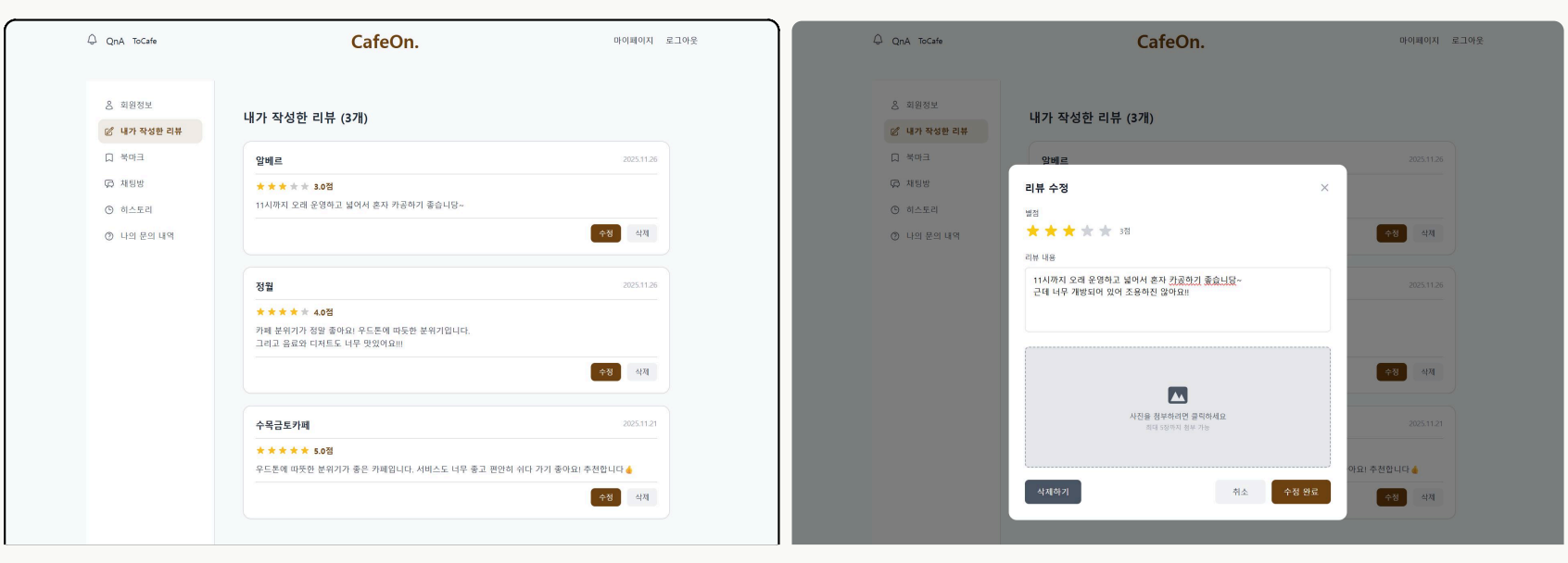
(1) 회원정보
GET /api/users/me로 프로필을 불러와 닉네임 수정(PUT /api/users/me), 프로필 이미지 변경(PUT /api/users/me/profile-image), 비밀번호 변경(PUT /api/users/me/password), 회원 탈퇴(DELETE /api/users/me)를 제공한다.

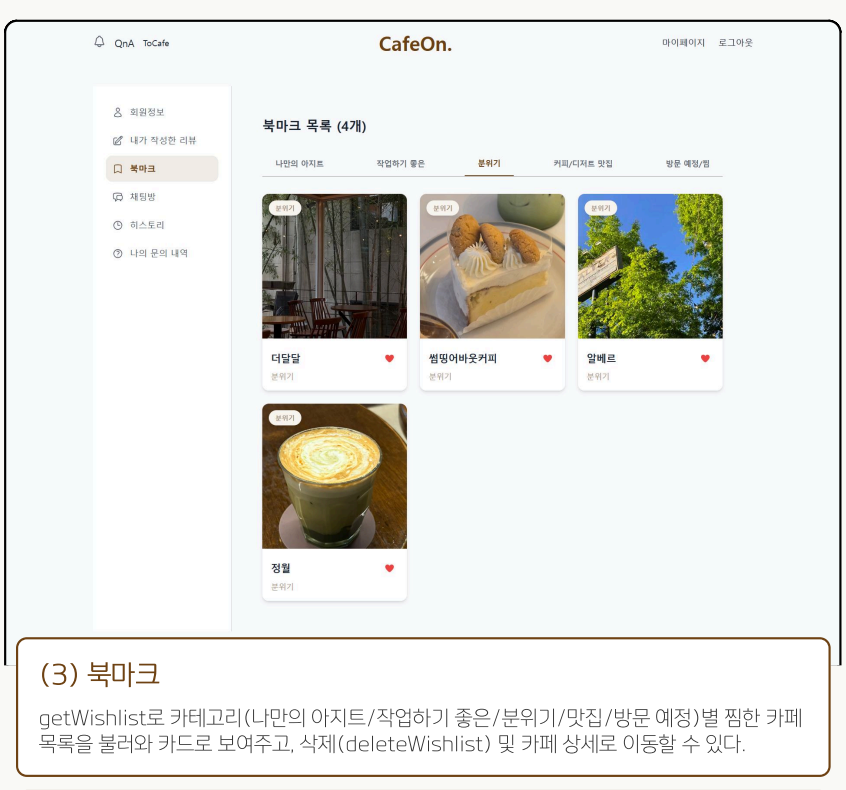
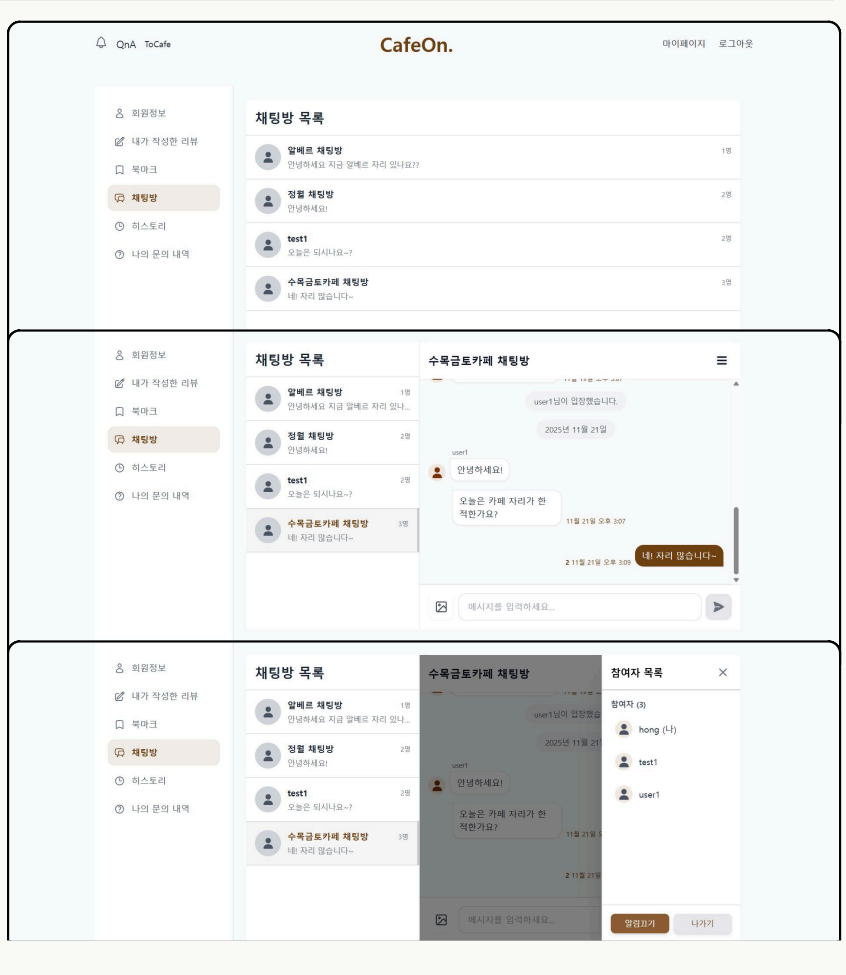
마이페이지

기능 개요
사이드바로 회원정보·리뷰·북마크·채팅·히스토리·문의 내역 고르고, 회원정보에서 프로필 조회/닉네임·프로필 이미지·비밀번호 수정·탈퇴를 하며, 나머지 메뉴는 각각 위시리스트·참여 채팅방·내 리뷰·내 글/댓글/좋아요·나의 QnA 목록을 API로 불러와 보여준다.

구현 포인트
(mypage) 레이아웃에 MypageSidebar(usePathname 기반 활성 메뉴)를 두고, 회원정보는 GET/PUT /api/users/me·프로필 이미지 FormData PUT·비밀번호 PUT·탈퇴 DELETE 후 AuthContext 갱신·리다이렉트로 처리한다. 북마크·채팅·리뷰·히스토리·QnA는 getWishlist· getMyChatRooms· getMyReviews· useMyPosts/useMyComments/useMyLikedPosts/useMyLikedComments·QnA 목록 훅 등으로 데이터를 채우고 탭·페이지네이션으로 보여준다.

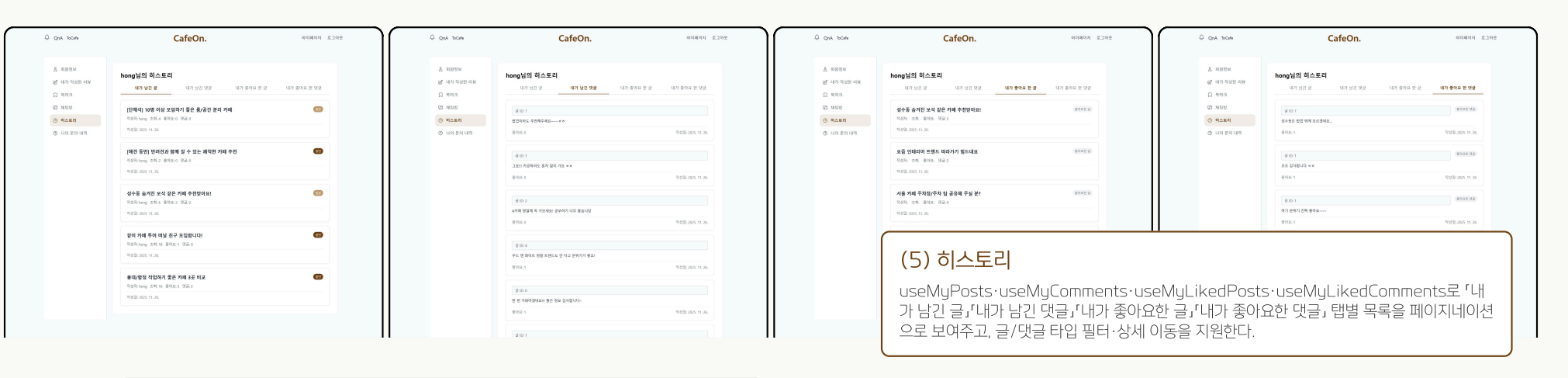
(2) 내가 작성한 리뷰
getMyReviews로 내가 쓴 리뷰 목록을 불러와 카페명·별점·내용·작성일을 보여주고, 수정(ReviewWriteModal)·삭제(deleteReview)를 할 수 있다.



(3) 북마크
getWishlist로 카테고리(나만의 아지트/작업하기 좋은/분위기/맛집/방문 예정)별 찜한 카페 목록을 불러와 카드로 보여주고, 삭제(deleteWishlist) 및 카페 상세로 이동할 수 있다.

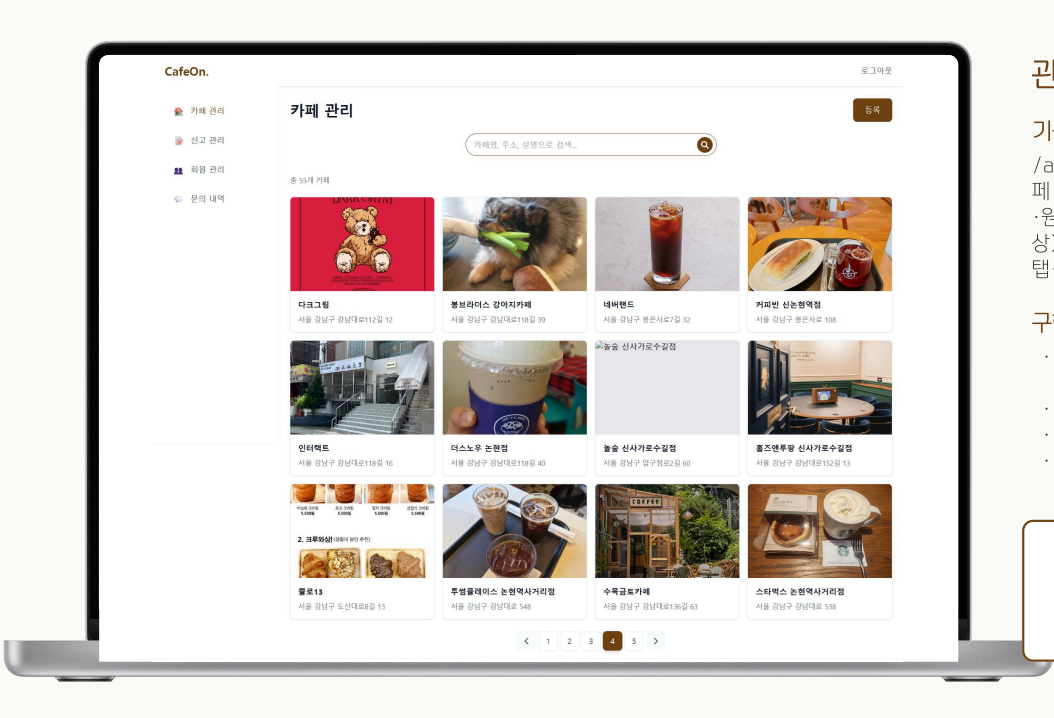
(4) 채팅방
getMyChatRooms로 참여 중인 채팅방 목록(단체·DM)을 보여주고, 방 선택 시 useCafeChat/useDmChat으로 채팅 화면을 띄우며 1:1 대화·나가기를 지원한다.



(5) 히스토리
useMyPosts·useMyComments·useMyLikedPosts·useMyLikedComments로 '내가 남긴 글','내가 남긴 댓글','내가 좋아요한 글','내가 좋아요한 댓글' 탭별 목록을 페이지네이션으로 보여주고, 글/댓글 타입 필터·상세 이동을 지원한다.

나의 문의 내역
문의 상세
문의 수정

(6) 나의 문의 내역
useQuestionList(또는 나의 문의 전용 API)로 내가 작성한 QnA 문의 목록을 불러와 제목·상태(답변 대기/완료)·작성일을 보여주고, 클릭 시 /mypage/qna/[id] 또는 /qna/[id] 상세로 이동한다.



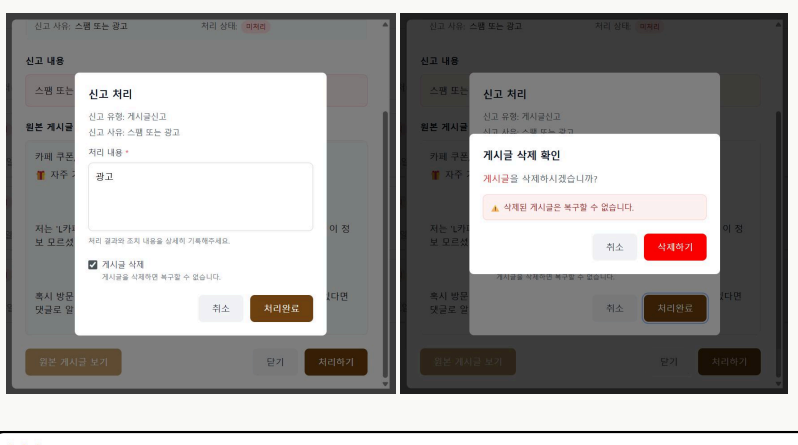
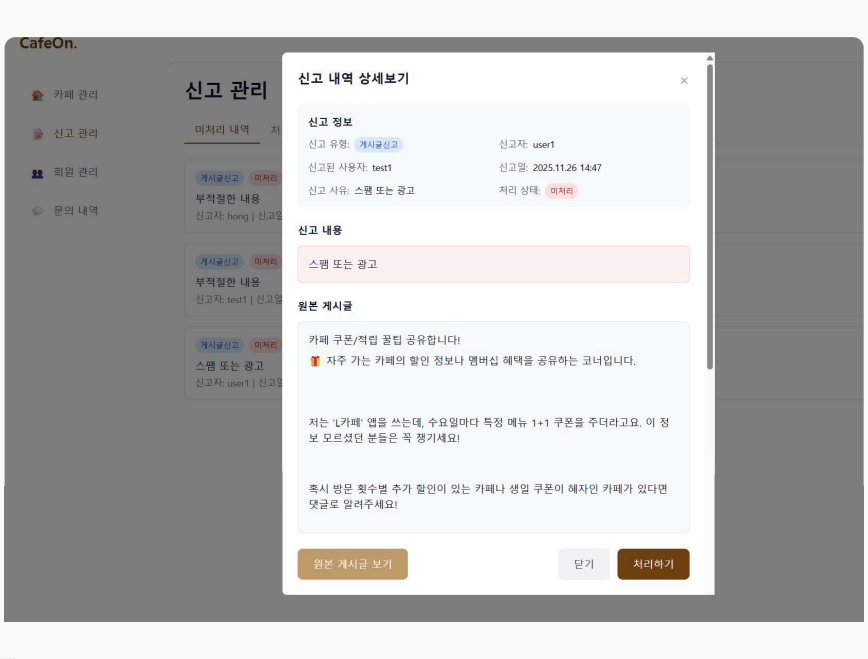
관리자 페이지

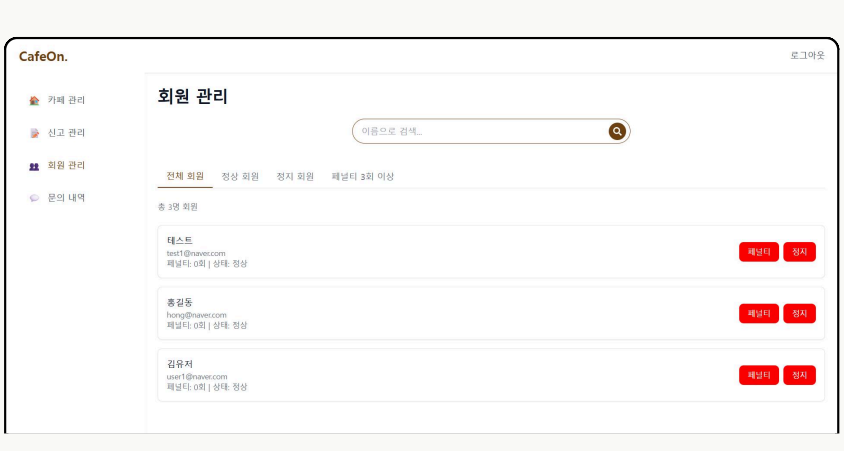
기능 개요
/admin은 사이드바로 카페 관리·신고 관리·회원 관리·문의 내역 네 구역을 둔다. 카페 관리는 전체 목록 조회·검색·페이지네이션, 신고 관리는 미처리/처리 탭·상세 보기·원본 삭제 여부 선택 후 처리 완료, 회원 관리는 탭(전체/정상/정지/패널티 3회 이상)·검색·패널티 부여·정지/해제·회원 상세(패널티 이력), 문의 내역은 미처리/처리 탭·상세 보기·관리자 답변 등록을 제공한다.

구현 포인트
- (admin) layout: 상단 헤더(로고·로그아웃) + 좌측 사이드바(카페/신고/회원/문의), usePathname()으로 활성 메뉴 강조.
- /admin 접속 시 router.replace("/admin/cafes")로 카페 관리로 리다이렉트.
- 모바일: sidebarOpen·햄버거·오버레이로 사이드바 열기/닫기.
- 각 하위 페이지: 인증 확인 후 API 호출, 에러·로딩 처리.

(1) 카페 관리
getAllCafes() 한 번 호출 후 searchTerm으로 이름/주소/설명 클라이언트 필터, 12개 단위 slice로 그리드·Pagination.

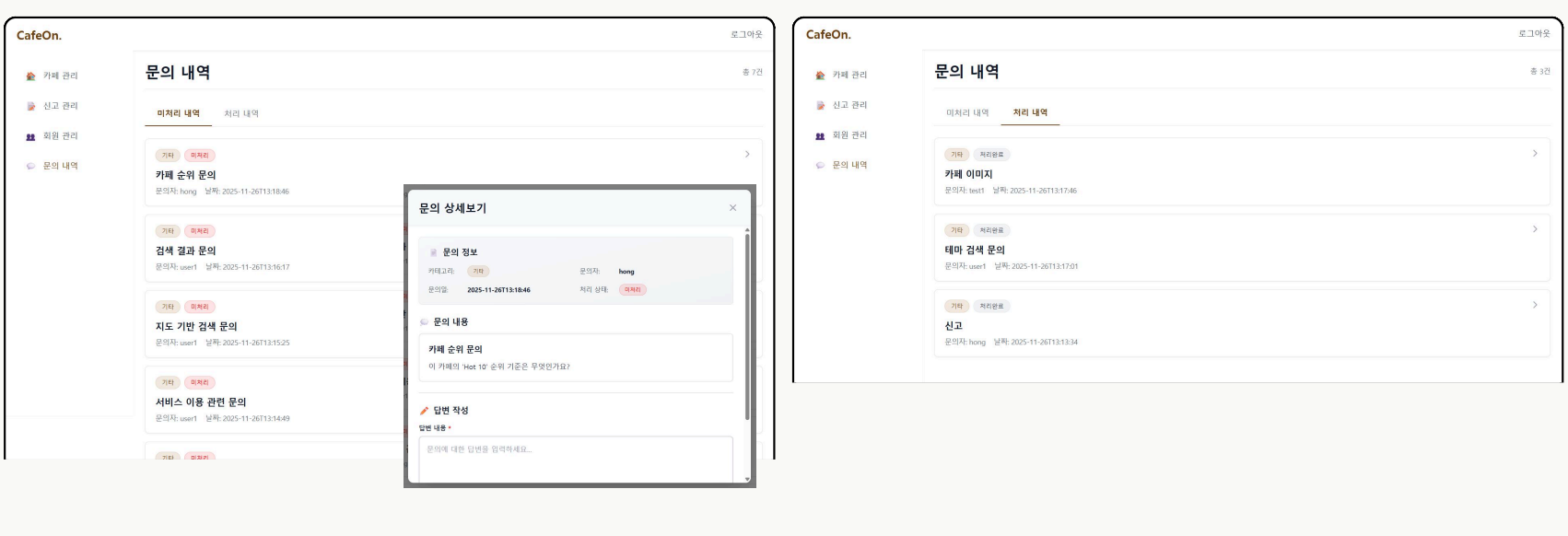
(2) 신고 관리
getAdminReports(PENDING/RESOLVED)로 탭별 목록, getAdminReportDetail(id)로 상세(원본 제목·본문·이미지). 처리 시 리뷰/게시글/댓글 관리자 코멘트 필수, "삭제" 체크 시 타입별 deleteReview/deleteCommentMutator/deletePostMutator 호출 후 updateAdminReport(id, { status: "RESOLVED" }).



(3) 회원 관리
getAdminMembers({ page, size, search, status })로 탭·검색·페이지네이션. "패널티 3회 이상"은 size 크게 두고 가져온 뒤 penaltyCount ≥ 3 클라이언트 필터. addAdminPenalty·suspendAdminUser 호출. 상세(/admin/members/[memberId])는 getAdminMemberDetail·getUserPenalties.

(4) 문의 내역
getAdminInquiries({ page, size, status: PENDING/ANSWERED })로 탭별 목록. 상세 모달은 getAdminInquiryDetail(id)·getAdminInquiryAnswers(id). 미처리일 때 createAdminInquiryAnswer(inquiryId, content)로 답변 등록 후 목록 재조회.



트러블 슈팅

1) API 예외 처리 구조 개선 및 무한 루프 방지

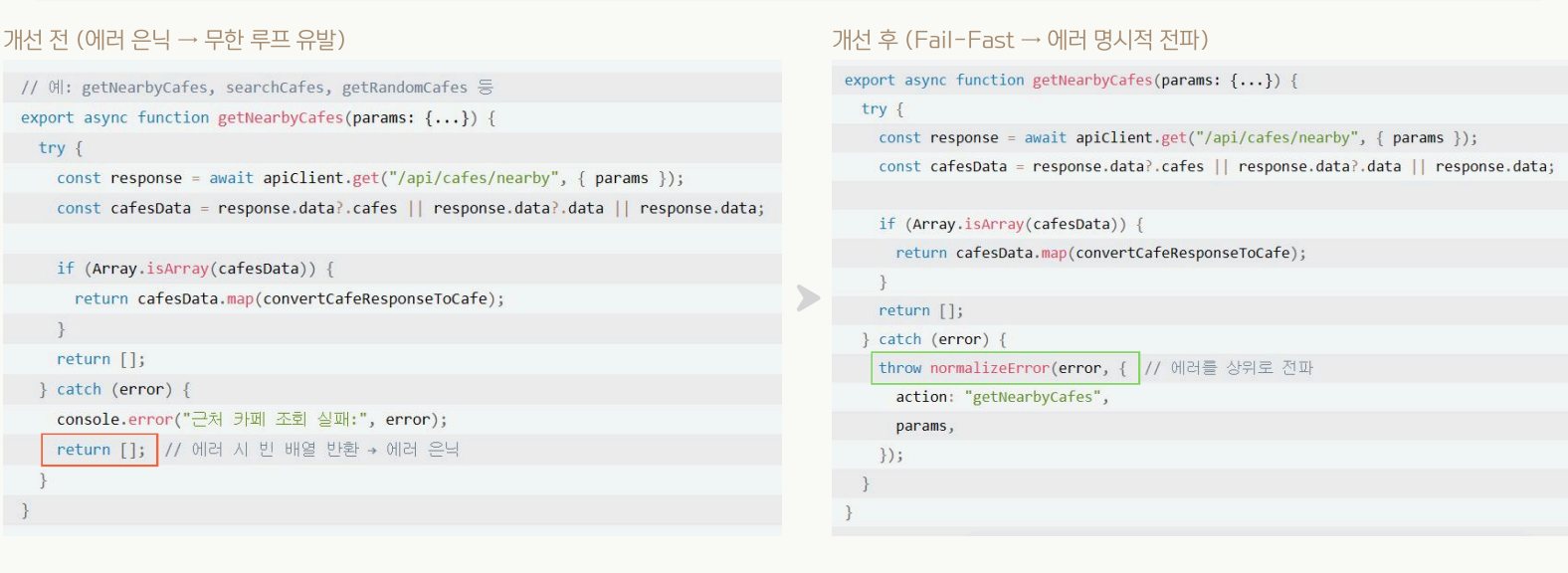
"에러를 숨기는 것이 아니라, 명시적으로 관리하여 시스템 자원을 보호하다."

문제 상황
네트워크 장애나 API 에러(404/500) 발생 시, API 함수가 빈 배열([])이나 가공된 기본값을 반환하는 **'에러 은닉'** 현상 발생.
→ 데이터가 비어있을 때 자동으로 재요청을 시도하는 UI 로직과 결합되어, 초당 수십 회의 **무한 재요청 루프**가 발생하여 서버 부하 급증.

원인 분석
안일한 예외 억제 방식이
컴포넌트의 선언적 렌더링 방식과 충돌하여 발생한 문제.

해결 방안
Fail-Fast 원칙 적용: 데이터를 위조해 반환하는 대신 throw normalizeError를 통해 상위 호출부로 에러를 명시적으로 전파.
에러 표준화 시스템 구축: AppError 클래스를 활용해 에러 발생 지점(Action)과 파라미터 메타데이터를 포함하도록 표준화하여 디버깅 효율성 제고.

성과
불필요한 네트워크 트래픽 차단 및 서버 가용성 확보, 잘못된 초기값으로 인한 오염된 데이터 렌더링 원천 차단
"에러를 감추기 위해 반환하던 기본값([])이 무한 루프의 원인임을 파악하고,
Fail-Fast 원칙에 기반한 명시적 에러 전파 구조로 개편하여 시스템의 안정성과 디버깅 효율을 높였다."



개선 전 (에러 은닉 → 무한 루프 유발)

```
// 예: getNearbyCafes, searchCafes, getRandomCafes 등
export async function getNearbyCafes(params: {...}) {
  try {
    const response = await apiClient.get("/api/cafes/nearby", { params });
    const cafesData = response.data?.cafes || response.data?.data || response.data;

    if (Array.isArray(cafesData)) {
      return cafesData.map(convertCafeResponseToCafe);
    }
    return [];
  } catch (error) {
    console.error("근처 카페 조회 실패:", error);
    return []; // 에러 시 빈 배열 반환 → 에러 은닉
  }
}
```

개선 후 (Fail-Fast → 에러 명시적 전파)

```
export async function getNearbyCafes(params: {...}) {
  try {
    const response = await apiClient.get("/api/cafes/nearby", { params });
    const cafesData = response.data?.cafes || response.data?.data || response.data;

    if (Array.isArray(cafesData)) {
      return cafesData.map(convertCafeResponseToCafe);
    }
    return [];
  } catch (error) {
    throw normalizeError(error, { // 에러를 상위로 전파
      action: "getNearbyCafes",
      params,
    });
  }
}
```

2) 실시간 상태 동기화 충돌 해결 (Race Condition)

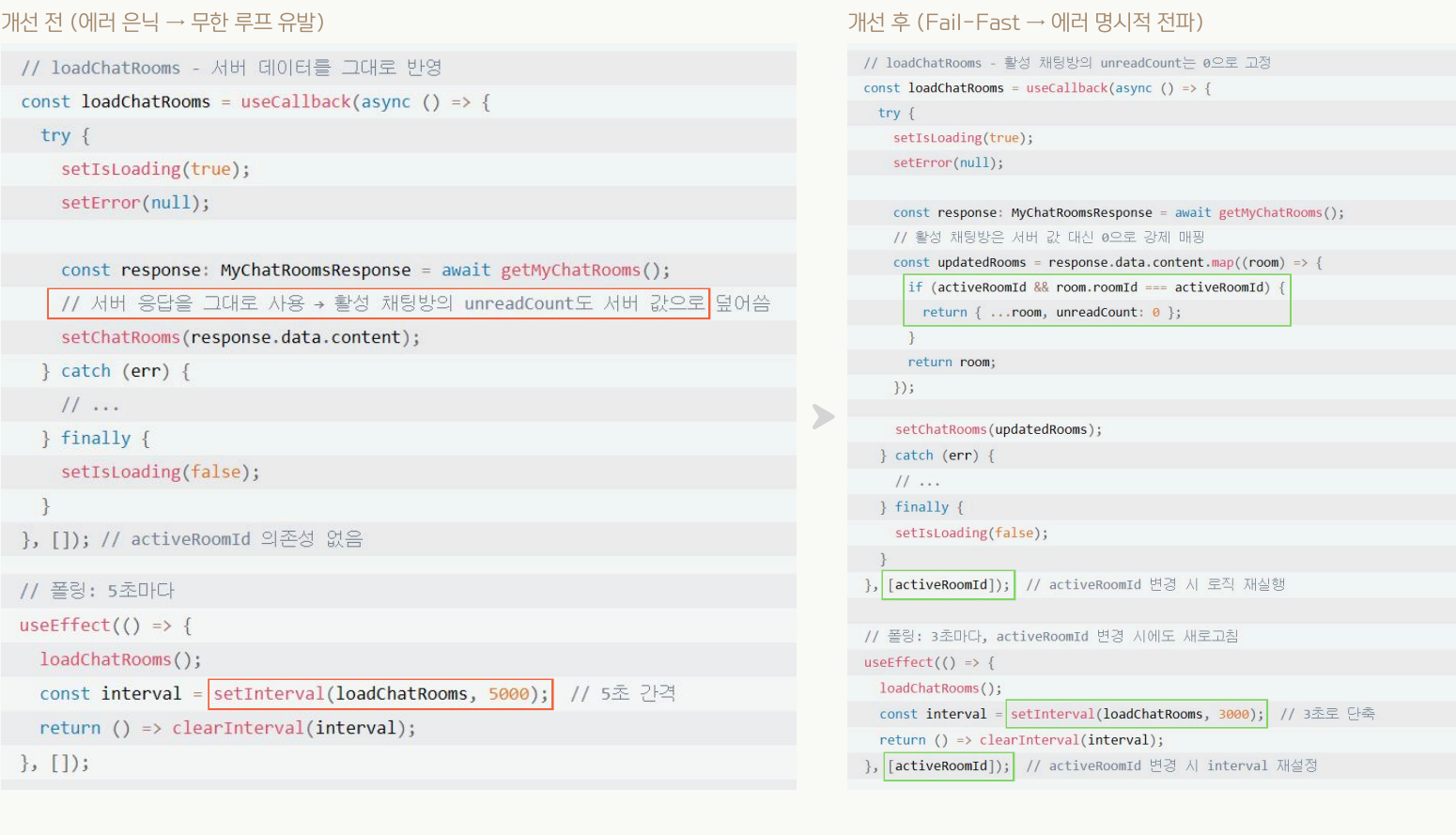
"로컬 상태와 서버 폴링 상태 간의 일관성 유지"

문제 상황
채팅 목록에서 메시지를 읽었음에도 불구하고,
안 읽은 메시지 수(unreadCount)가 다시 살아나거나
실시간으로 업데이트되지 않는 현상 발생.

원인 분석
상태 덮어쓰기: 사용자가 채팅방을 클릭해 로컬에서 unreadCount를 0으로 설정했으나, 직후 실행된 서버 폴링 데이터가 아직 업데이트되지 않은 서버 측 데이터를 가져와 로컬 상태를 이전 값으로 복구시킴.
→ 서버가 나의 속도를 못 따라오는 상황 (Race Condition)

해결 방안
폴링 로직 최적화: loadChatRooms 함수 내 서버 데이터를 수신할 때, 현재 활성화된 채팅방(activeRoomId) 경우 서버 값 대신 강제로 0을 유지하도록 매핑
주기적 갱신 전략 수정: 업데이트 간격을 5초에서 3초로 단축하고, 특정 조건문 없이 모든 채팅방의 실시간성을 보장하도록 개선.

성과
사용자 인터랙션(읽음 처리)과 서버 비동기 데이터 간의 데이터 일관성 확보 및 UX 향상.
"서버 폴링 데이터가 로컬의 사용자 인터랙션 상태를 덮어쓰는 레이스 컨디션을 발견하고,
활성화된 세션의 데이터를 강제 매핑하는 로직을 통해 클라이언트-서버 간 데이터 일관성을 확보했다."



개선 전 (에러 은닉 → 무한 루프 유발)

```
// loadChatRooms - 서버 데이터를 그대로 반영
const loadChatRooms = useCallback(async () => {
  try {
    setIsLoading(true);
    setError(null);

    const response: MyChatRoomsResponse = await getMyChatRooms();
    // 서버 응답을 그대로 사용 → 활성 채팅방의 unreadCount도 서버 값으로 덮어씀
    setChatRooms(response.data.content);
  } catch (err) {
    // ...
  } finally {
    setIsLoading(false);
  }
}, []); // activeRoomId 의존성 없음

// 폴링: 5초마다
useEffect(() => {
  loadChatRooms();
  const interval = setInterval(loadChatRooms, 5000); // 5초 간격
  return () => clearInterval(interval);
}, []);
```

개선 후 (Fail-Fast → 에러 명시적 전파)

```
// loadChatRooms - 활성 채팅방의 unreadCount는 0으로 고정
const loadChatRooms = useCallback(async () => {
  try {
    setIsLoading(true);
    setError(null);

    const response: MyChatRoomsResponse = await getMyChatRooms();
    // 활성 채팅방은 서버 값 대신 0으로 강제 매핑
    const updatedRooms = response.data.content.map((room) => {
      if (activeRoomId && room.roomId === activeRoomId) {
        return { ...room, unreadCount: 0 };
      }
      return room;
    });

    setChatRooms(updatedRooms);
  } catch (err) {
    // ...
  } finally {
    setIsLoading(false);
  }
}, [activeRoomId]); // activeRoomId 변경 시 로직 재실행

// 폴링: 3초마다, activeRoomId 변경 시에도 새로고침
useEffect(() => {
  loadChatRooms();
  const interval = setInterval(loadChatRooms, 3000); // 3초로 단축
  return () => clearInterval(interval);
}, [activeRoomId]); // activeRoomId 변경 시 interval 재설정
```

3) 식별자 체계 불일치 해결 (UUID Mapping)

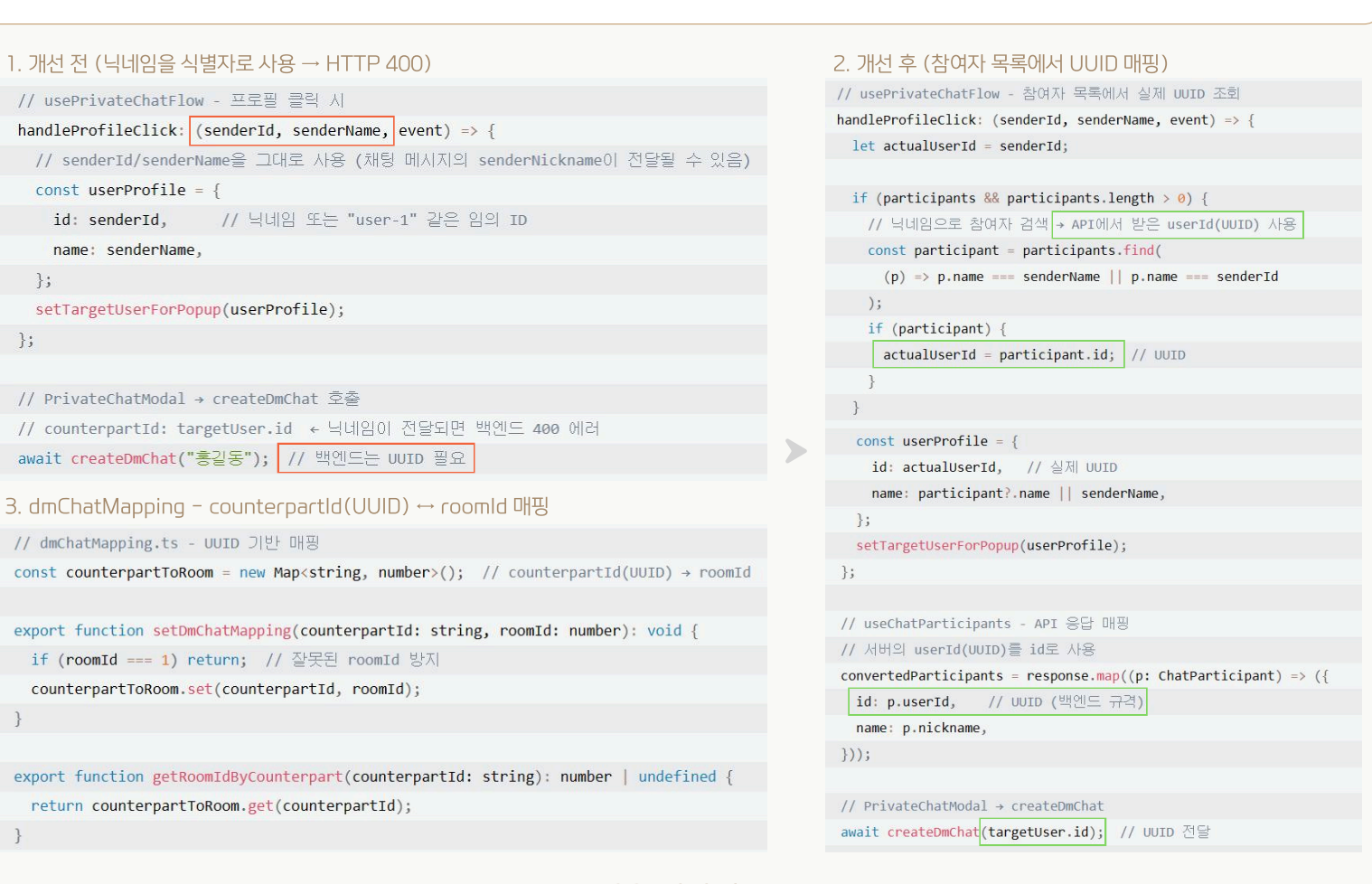
"UI용 정보와 서버 식별 데이터의 물리적 분리 및 매핑"

문제 상황
이름은 아는데 주소를 모르는 상황.
1:1 채팅 생성 시 백엔드에서 HTTP 400 에러가 빈번하게 발생하거나
메시지가 다른 채팅방으로 전송되는 문제 발생.

원인 분석
식별자 혼용: 프론트엔드에서 닉네임(String)이나 임의의 ID를 식별자로 사용했으나, 백엔드는 실제 DB의 UUID 규격을 요구함.
즉, UI 노출용 데이터와 통신용 데이터의 규격이 일치하지 않음.

해결 방안
데이터 매핑 아키텍처 구축: 채팅 참여자 목록 로드 시 닉네임과 UUID를 짝지어 관리하는 cafeToRoom 매핑 유틸리티 구현.
전역 상태 통합: Zustand와 Context API를 결합하여 인증 토큰 내의 실제 UUID를 추출, 모든 API 요청 시 닉네임이 아닌 매핑된 실제 식별자를 사용하도록 강제.

성과
프론트-백엔드 간 데이터 정합성 100% 확보 및 통신 장애 해결.
"UI용 닉네임과 서버용 UUID 간의 데이터 규격 불일치로 인한 통신 장애를 해결하기 위해, 식별자 매핑 아키텍처를 설계하여 데이터 정합성을 확보했다."



1. 개선 전 (닉네임을 식별자로 사용 → HTTP 400)

```
// usePrivateChatFlow - 프로필 클릭 시
handleProfileClick: (senderId, senderName, event) => {
  // senderId/senderName을 그대로 사용 (채팅 메시지의 senderNickname이 전달될 수 있음)
  const userProfile = {
    id: senderId,       // 닉네임 또는 "user-1" 같은 임의 ID
    name: senderName,
  };
  setTargetUserForPopup(userProfile);
};

// PrivateChatModal → createDmChat 호출
// counterpartId: targetUser.id → 닉네임이 전달되면 백엔드 400 에러
await createDmChat("홍길동"); // 백엔드는 UUID 필요
```

3. dmChatMapping - counterpartId(UUID) ↔ roomId 매핑

```
// dmChatMapping.ts - UUID 기반 매핑
const counterpartToRoom = new Map<string, number>(); // counterpartId(UUID) → roomId

export function setDmChatMapping(counterpartId: string, roomId: number): void {
  if (roomId === 1) return; // 잘못된 roomId 방지
  counterpartToRoom.set(counterpartId, roomId);
}

export function getRoomIdByCounterpart(counterpartId: string): number | undefined {
  return counterpartToRoom.get(counterpartId);
}
```

2. 개선 후 (참여자 목록에서 UUID 매핑)

```
// usePrivateChatFlow - 참여자 목록에서 실제 UUID 조회
handleProfileClick: (senderId, senderName, event) => {
  let actualUserId = senderId;

  if (participants && participants.length > 0) {
    // 닉네임으로 참여자 검색 → API에서 받은 userId(UUID) 사용
    const participant = participants.find(
      (p) => p.name === senderName || p.name === senderId
    );
    if (participant) {
      actualUserId = participant.id; // UUID
    }
  }

  const userProfile = {
    id: actualUserId,    // 실제 UUID
    name: participant?.name || senderName,
  };
  setTargetUserForPopup(userProfile);
};

// useChatParticipants - API 응답 매핑
// 서버의 userId(UUID)를 id로 사용
convertedParticipants = response.map((p: ChatParticipant) => ({
  id: p.userId,    // UUID (백엔드 규격)
  name: p.nickname,
}));

// PrivateChatModal → createDmChat
await createDmChat(targetUser.id); // UUID 전달
```

얻은 인사이트

에러 은닉의 위험
기본값 반환은 단기적으로 편하지만,
무한 루프·잘못된 렌더링 등 부작용을 키운다.

클라이언트-서버 상태 동기화
비동기 폴링과 로컬 상태가 충돌할 때,
"사용자 행동 우선" 규칙을 명확히 해야 한다.

식별자 규격 통일
UI용 데이터(닉네임)와 API용 식별자(UUID)를 분리하고,
매핑 레이어를 두는 것이 중요하다.

향후 개선 방향

단기 (1~2주)
에러 바운더리: AppError 기반으로 전역 에러 바운더리 도입, 사용자 친화적 메시지 표시
폴링 최적화: 채팅방 목록 폴링을 WebSocket/SSE로 전환해 실시간성·부하 개선 검토

중기 (1~2개월)
API 레이어 표준화: normalizeError 패턴을 모든 API 함수에 적용, 에러 타입·코드 체계 정리

장기 (3개월+)
타입 안전성 강화: API 응답·요청에 Zod 등으로 런타임 검증 도입

https://blog.naver.com/higkmin/224014381328
https://github.com/haminee01/Project_CafeOn.git
gkmin22@gmail.com