

졸업작품 종합설계 결과보고서

과제명 : 냉장고와 함께하는 나만의 메뉴

2025. 12. 01.

팀원 : 하종수 (팀장), 박기준 , 정승훈 , 김강민

동서울대학교 컴퓨터소프트웨어학과

조 번호	3 - C 반- 8조		
졸업작품종합설계 요약문			
유형	<input type="checkbox"/> H/W 과제 <input type="checkbox"/> S/W 과제 <input type="checkbox"/> H/W + S/W 과제		
졸업작품명	냉장고와 함께하는 나만의 메뉴		
팀장	학번 / 성명	2106066 / 하종수	
팀원성명	학번	성명	역할 및 개발분야
	2106066	하종수	프로젝트 총괄(PM)로서 전체 일정 관리 및 팀원 간 의견 조율을 담당하고, UI/UX 디자인 및 프론트엔드 구현과 통합 테스트(QA)를 수행함
	2006094	박기준	전체 시스템 구조 설계 및 개발을 총괄하고, 레시피 데이터 구축과 안드로이드 앱 및 백엔드 전반을 구현함
	2106089	정승훈	프로젝트 기획 및 문서화(보고서/자료)를 지원하고, 앱 기능 수동 테스트 및 버그 리포팅을 통한 품질 관리(QA)를 수행함
	2006103	김강민	프로젝트 초기 기획 단계에서 요구사항 분석 및 기능 명세서를 작성하고, 시스템의 기초 UI 구조와 사용자 시나리오 설계를 수행함
요약	<p>1. 연구 배경 및 목적 최근 1인 가구의 증가와 바쁜 현대인의 라이프스타일로 인해, 구매한 식재료를 잊고 방치하여 폐기하는 식재료 낭비 문제가 빈번하게 발생하고 있다. 또한, 매일 반복되는 “오늘 뭐 먹지?”라는 메뉴 선정의 고민은 많은 사용자에게 스트레스로 작용한다. 본 프로젝트는 이러한 문제를 해결하기 위해, 사용자의 냉장고 속 식재료를 체계적으로 관리하고 이를 바탕으로 최적의 레시피를 추천해 주는 안드로이드 애플리케이션 ‘요리GO조리GO’를 기획 및 개발하였다.</p> <p>2. 주요 구현 내용 및 기술적 특징 본 시스템은 안드로이드 MVP(Model-View-Presenter) 아키텍처를 기반으로 설계되어 UI와 비즈니스 로직을 철저히 분리하였으며, Jetpack Navigation을 도입하여 효율적인 화면 전환 구조를 확립하였다. 백엔드 인프라는 Google Firebase(Auth, Firestore, Cloud Functions)를 활용하여 서비스(Serverless) 환경으로 구축하였으며, 실시간 데이터 동기화와 안정적인 회원 관리 시스템을 구현하였다. 특히, 기존 데이터베이스 검색의 한계를 극복하기 위해 Algolia 검색 엔진과 Okta 형태소 분석기를 결합한 하이브리드 검색 시스템을 자체 구축하였다. 이를 통해 자연어 입력 시 핵심 키워드 추출과 초고속 검색이 가능하도록 고도화하였으며, Python 웹 크롤링을 통해 확보한 10만 건의 대용량 레시피 데이터를 탑재하여 상용 수준의 정보량을 확보하였다. 또한, Android WorkManager를 활용하여 앱이 종료된 상태에서도 유통기한 임박 식재료를 감지하고 푸시 알림을 전송하는 백그라운드 서비스를 구현하였다.</p> <p>3. 결과 및 기대효과 최종 구현된 시스템은 식재료 등록부터 검색, 추천, 알림에 이르는 전 과정을 유기적으로 연결하여 사용자 편의성을 극대화하였다. 이를 통해 사용자는 냉장고 파먹기를 실천하여 음식물 쓰레기를 줄이는 친환경적인 생활이 가능하며, 개인화된 레시피 추천을 통해 메뉴 선정에 소요되는 시간과 노력을 획기적으로 절감할 수 있을 것으로 기대된다.</p>		

목 차

1. 서론	1
1.1 배경 및 필요성	1
1.2 개발 목표	2
1.3 팀구성 및 추진일정	4
2. 본론	5
2.1 목표 시스템 개요	5
2.2 개발 및 운영 환경	7
2.3 설계내용	11
2.4 제작 과정 및 결과물	14
2.5 시연 결과 및 자체평가	14
2.6 개인별 공헌내용	14
3. 결론	20
참고문헌	20
부록. 소스코드 List	40
1. front-end 관련 Android MVP & Core Logic	40
2. back-end 및 Data Engineering 관련	42
3. DB 관련 (Firestore Data Modeling)	45

1. 서 론

1.1 배경 및 필요성

- 배경:** 1인 가구의 증가와 바쁜 현대인의 라이프스타일로 인해 식재료를 구매해두고 잊어 버려 폐기하는 일이 빈번하게 발생한다. 또한, “냉장고에 있는 재료로 무엇을 해 먹을지”에 대한 고민은 매일 반복되는 숙제와 같다.
- 필요성:** 기존 레시피 앱들은 대부분 단순히 요리법만 나열하는 형태로, 사용자가 현재 보유한 재료와 직접적인 연관성을 찾기 어렵다. 또한 실제 시장조사 결과, Google Play 스토어 기준으로 음식 재고 관리/유통기한 알림을 제공하는 앱과 레시피 추천 앱이 각각 별도의 서비스로 분리되어있어 (만개의 레시피, 유통기한 알리미), 한 곳에서 통합적으로 관리하기 불편하다. 따라서 사용자가 자신의 냉장고 상태(재료 재고, 유통기한 등)를 직관적으로 관리하고 이를 바탕으로 당장 실현 가능한 레시피를 자동으로 추천받을 수 있는 통합 플랫폼의 필요성이 더욱 커지고 있다.

1.2 개발 목표

- 개발 범위** 본 프로젝트는 식재료 관리 및 레시피 추천을 핵심으로 하는 안드로이드 네이티브 애플리케이션 개발을 목표로 하며, 구체적인 개발 범위는 다음과 같다.

기존 기능의 고도화

- 기존 수준:** 단순한 키워드 매칭(DB LIKE 연산) 방식은 정확도가 낮고, ‘맛있는 김치 찌개’와 같은 복합어 검색 시 결과가 누락되는 한계가 있음.
- 개발 목표:** 대용량 데이터의 초고속 검색을 위해 외부 검색 엔진인 Algolia를 연동하고, Okt 형태소 분석기를 추가로 도입한다. 이를 통해 입력어에서 동의어 처리 조사와 형용사를 배제하고 핵심 명사만 추출하여 검색 정밀도를 높인다.

시스템 안정성 강화

- 기준 수준:** 일반적인 알람 방식은 앱이 종료되거나 기기가 절전 모드에 진입하면 알람이 누락될 위험이 있음
- 개발 목표:** 안드로이드 시스템이 보장하는 WorkManager API를 활용하여 백그라운드 작업을 구현한다. 앱 실행 여부와 관계없이 24시간 주기로 유통기한 임박 재료를 푸시 알림을 전송한다.

아키텍처 및 데이터 동기화

- 기준 수준:** 클라이언트가 모든 데이터를 직접 처리하거나, UI 코드에 비즈니스 로직이 섞여 있어 유지보수가 어려움.
- 개발 목표:** MVP 디자인 패턴을 적용하여 뷰와 로직을 철저히 분리하고, Firebase Cloud Functions를 활용한 아키텍처를 도입한다. 데이터 변경 시 검색 인덱스와 알람 대기열이 자동으로 동기화되는 이벤트 기반 시스템을 구현하여 클라이언트 부하를 최소화한다.

2. 기대효과

- 사용자 측면:** 냉장고 속 잊혀진 재료의 유통기한을 사전에 인지하여 음식물 쓰레기를 줄이고, 보유 재료 기반의 레시피 추천을 통해 “오늘 뭐 먹지?”라는 매뉴 선정의 고민 시간을 단축할 수 있다.
- 기술적 측면:** MVP 패턴 적용을 통해 코드의 유지보수성을 높이고, Firebase 와 외부 검색 엔진(Algolia), GitHub 협업등 협업에서 사용되는 최신 라이브러리와 기술 스택을 경험 함으로써 안드로이드 개발 역량을 강화한다.

1.3 팀구성 및 추진일정

1) 업무분담

구분	학번	성명	역할 및 개발분야
팀장	2106066	하종수	<ul style="list-style-type: none">프로젝트 총괄(기획/일정) 및 통합 테스트(QA) 수행안드로이드 UI/UX 디자인 및 프론트엔드(XML/View) 구현
팀원	2006094	박기준	<ul style="list-style-type: none">전체 시스템 구조(Client-Server) 설계 및 개발 총괄레시피 데이터 수집(크롤링/전처리) 및 DB 설계안드로이드 앱 전반(MVP, 검색, 알림) 및 백엔드
팀원	2106089	정승훈	<ul style="list-style-type: none">프로젝트 기획 및 문서화: 보고서 작성 지원 및 자료 정리품질 관리(QA): 앱 기능 수동 테스트 및 버그 리포팅
팀원	2006103	김강민	<ul style="list-style-type: none">프로젝트 기획(아이디어/요구사항) 및 기능 명세서 작성시스템 초기 설계(UI 구조) 및 사용자 시나리오 구상

2) 추진일정

연 구 내 용	연 구 기 간						
	3월	4월	5월	6월	9월	10월	11월
자료 조사 및 프로젝트 계획, 요구사항 분석	●	●	●				
시스템 설계 (DB/UI/아키텍처)			●	●			
안드로이드 앱 및 서버 구현					●	●	●
시스템 통합 테스트 및 안정화							●

2. 본 론

2.1 목표 시스템 개요

1) 전체 시스템 구성도

본 시스템은 유지보수성과 확장성을 고려하여 MVP(Model-View-Presenter) 패턴을 적용 하였으며, 물리적인 서버 구축 비용을 절감하고 운영 효율을 높이기 위해 Google Firebase 기반의 서비스 아키텍처를 채택하였다.

1. Client Layer (Android Mobile)

- 가. H/W: Android OS 8.0 (Oreo) SDK 26 이상이 탑재된 모바일 디바이스.
- 나. Architecture: MVP 패턴을 도입하여 UI(View) 와 비즈니스 로직(Presenter), 데이터 처리 (Model)를 명확히 분리함으로써 코드의 재사용성과 테스트 용이성을 확보하였다.

다. Major Components

- 1) UI Layer: Jetpack Navigation Component를 도입하여 메인 부분 단일 액티비티 구조 하에 Fragment 간의 유연하고 효율적인 화면전환을 관리한다. Activity 와 Fragment로 구성되며, 사용자의 입력을 받아 Presenter에 전달한다.
- 2) Presentation Layer: 사용자 이벤트를 처리하고 Model 데이터를 요청하며, 결과를 가공하여 View를 갱신한다.
- 3) Data Layer (Repository): Firestore, Algolia 등 외부 데이터 소스와 통신하며 데이터의 CRUD를 담당한다.
- 4) Background Worker: Android WorkManager를 활용하여 앱 실행 여부와 관계없이 24 시간 주기로 유통기한 임박 재료를 체크하고 알림을 생성한다.

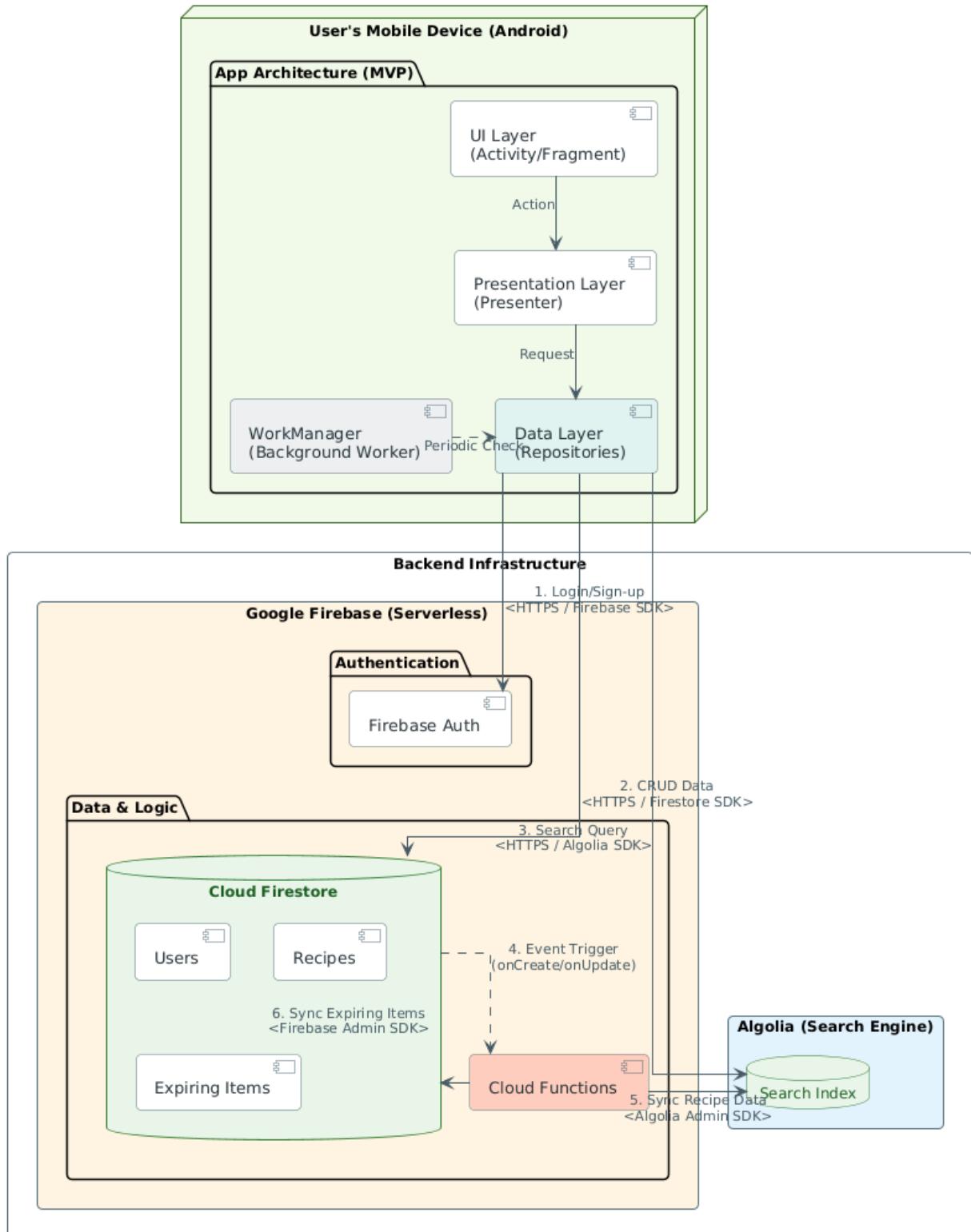
2. Backend & Infrastructure Layer

- 가. Authentication: Firebase Auth를 통해 이메일 및 Google 소셜 로그인을 처리하며, 안전한 세션 관리와 사용자 식별을 수행한다.
- 나. Database: Cloud Firestore (NoSQL) 를 사용하여 유저 데이터(냉장고 재료, 즐겨찾기)와 10만개의 레시피 데이터를 실시간으로 동기화 및 저장한다.
- 다. Business Logic: Firebase Cloud Functions (TypeScript)를 통해 데이터베이스의 변경사항 (트리거)을 감지하여 검색 인덱스 동기화 및 알림 데이터 정제등의 서버 측 로직을 수행 한다.

3. Search Engine Layer

- 가. Algolia: 대용량 레시피 데이터의 초고속 전문검색(Full-text Search)을 담당한다. Firestore 의 쿼리 성능 한계를 보완하고, 오타 보정 및 랭킹 알고리즘을 통해 정확도 높은 검색 결과를 클라이언트에 제공한다.

요리GO조리GO - System Architecture & Deployment Diagram



2.2 개발 및 운영환경

1) 개발 환경

구분	항목	상세 내용	비고
H/W	PC	Intel Core i5 이상, RAM 16GB 이상	Android Studio 구동 및 에뮬레이터 테스트
S/W	OS	Windows 10 / 11 (64bit)	개발용 운영체제
	IDE	Android Studio (Narwhal 2025.1.4 또는 최신 버전)	안드로이드 클라이언트 개발 및 디버깅
		Visual Studio Code	Cloud Functions (백엔드) & Python(데이터전 처리) 코드 작성
Language	Client	Java 11	안드로이드 앱 주요 비 즈니스 로직 구현
		Kotlin (DSL)	Gradle 빌드 스크립트 (build.gradle.kts) 작성
	Server	TypeScript (v5.7.3)	Cloud Functions 백엔 드 로직 구현
		Node.js (v20)	백엔드 런타임 환경
Libraries	Core	Android SDK (API Level 36)	안드로이드 앱 개발 프 레임워크 (minSdk 26)
	Network & Data	Firebase SDK, Gson (v2.13.2)	데이터베이스 연동 및 JSON 데이터 파싱
	UI & Image	Material Design 3, Glide (v5.0.5)	UI 컴포넌트 활용 및 이미지 로딩/캐싱
	Search & NLP	Algolia Search Client, Open Korean Text	검색 엔진 연동 및 한 국어 형태소 분석
Tools	VCS	Git, GitHub	소스 코드 버전 관리 및 협업
	DB Tool	Firebase Console	Firestore 데이터베이스 및 Authentication 관리

2) 운영 환경

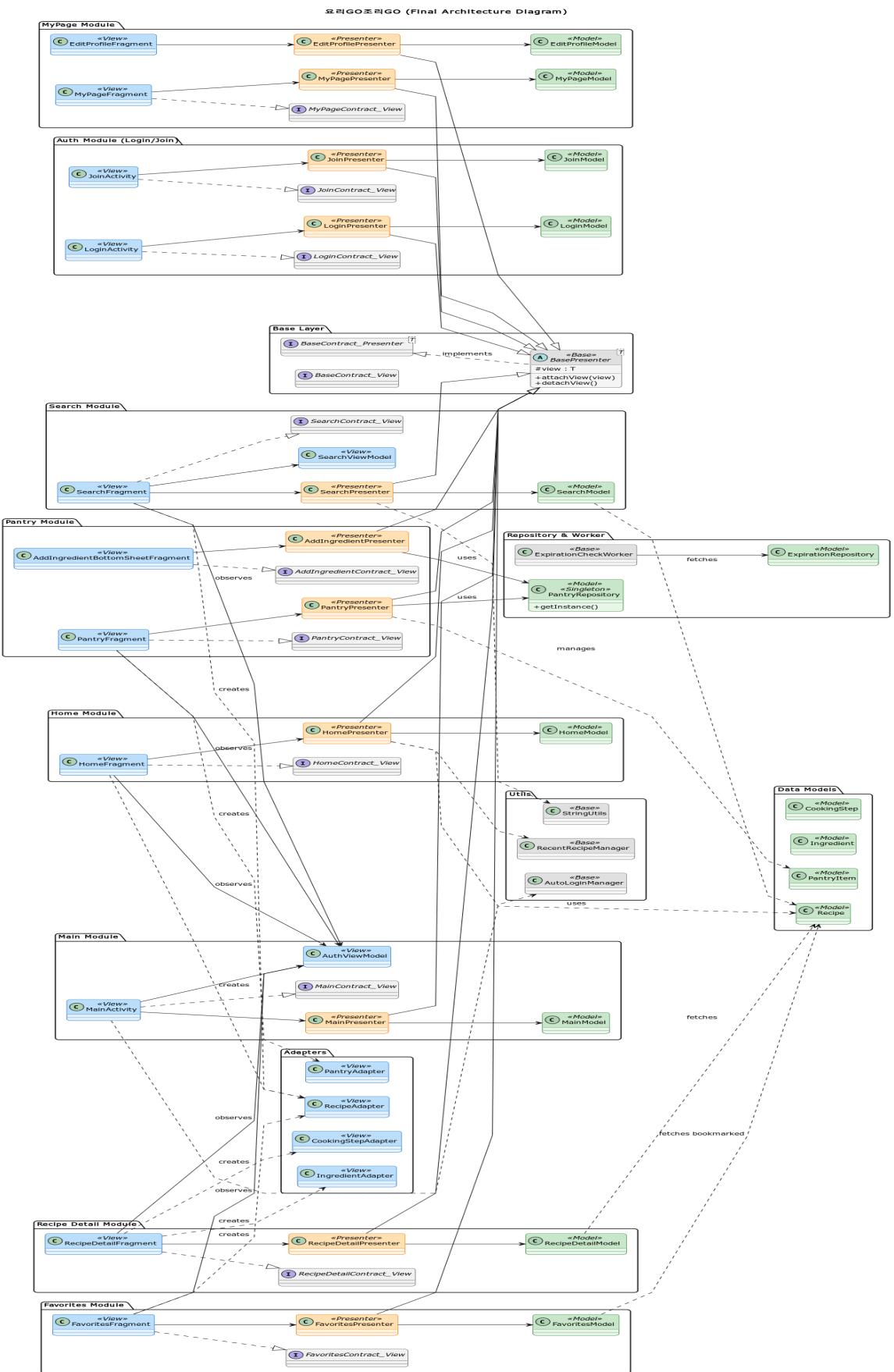
구분	항목	상세내용	비고
Client	Device	Android OS가 탑재된 스마트폰	삼성 갤럭시 S 시리즈, 구글 픽셀 등
(App)	OS	Android	minSdk = 26 설정에 따름 (안드로이드 8.0 Oreo) 이상
	Target	Android 15 (API 36) 호환	최신 안드로이드 운영체제 지원
	Network	LTE, 5G & Wi-Fi 네트워크	실시간 데이터 동기화 및 검색을 위해 인터넷 필수
Server	Cloud	Google Firebase	별도의 물리 서버 구축 없이 클라우드 환경에서 운영
Backend	Runtime	Node.js 20	Firebase Cloud Functions 실행 환경 (functions/package.json 설정)
	Auth	Firebase Authentication	사용자 인증 및 세션 관리 (이메일, Google 로그인)
Database	DB	Cloud Firestore	사용자 정보(냉장고, 즐겨찾기) 및 레시피 데이터(10만 건) 실시간 저장
Search	Engine	Algolia	대용량 레시피 데이터의 인덱싱 및 초고속 전문 검색 (Full-text Search) 처리

2.3 설계내용

본 프로젝트는 기획된 요구사항을 바탕으로 실제 구현된 시스템의 구조를 체계적으로 시각화 하였다. 시스템의 전체적인 기능 명세는 유스케이스 다이어그램으로, 소프트웨어 아키텍쳐는 클래스다이어그램으로, 데이터 구조는 ERD로, 그리고 핵심 기능의 동작 흐름은 시퀀스 다이어그램으로 각각 정의하였다.

1) 시스템 아키텍쳐 및 클래스 다이어그램

안드로이드 애플리케이션의 유지보수성과 확장성을 확보하기 위해 MVP(Model-View-Presenter) 아키텍쳐 패턴을 도입하여 설계하였다.



구조적 특징: UI를 담당하는 View 와 비즈니스 로직을 처리하는 Presenter , 데이터 통신을 담당하는 Model 을 철저히 분리하였다.

Base Layer: BaseContract 와 BasePresenter 추상 클래스를 정의하여 모든 모듈이 공통된 생명주기 관리 로직을 상속받도록 설계함으로써 메모리 누수를 방지하고 코드의 일관성을 유지하였다.

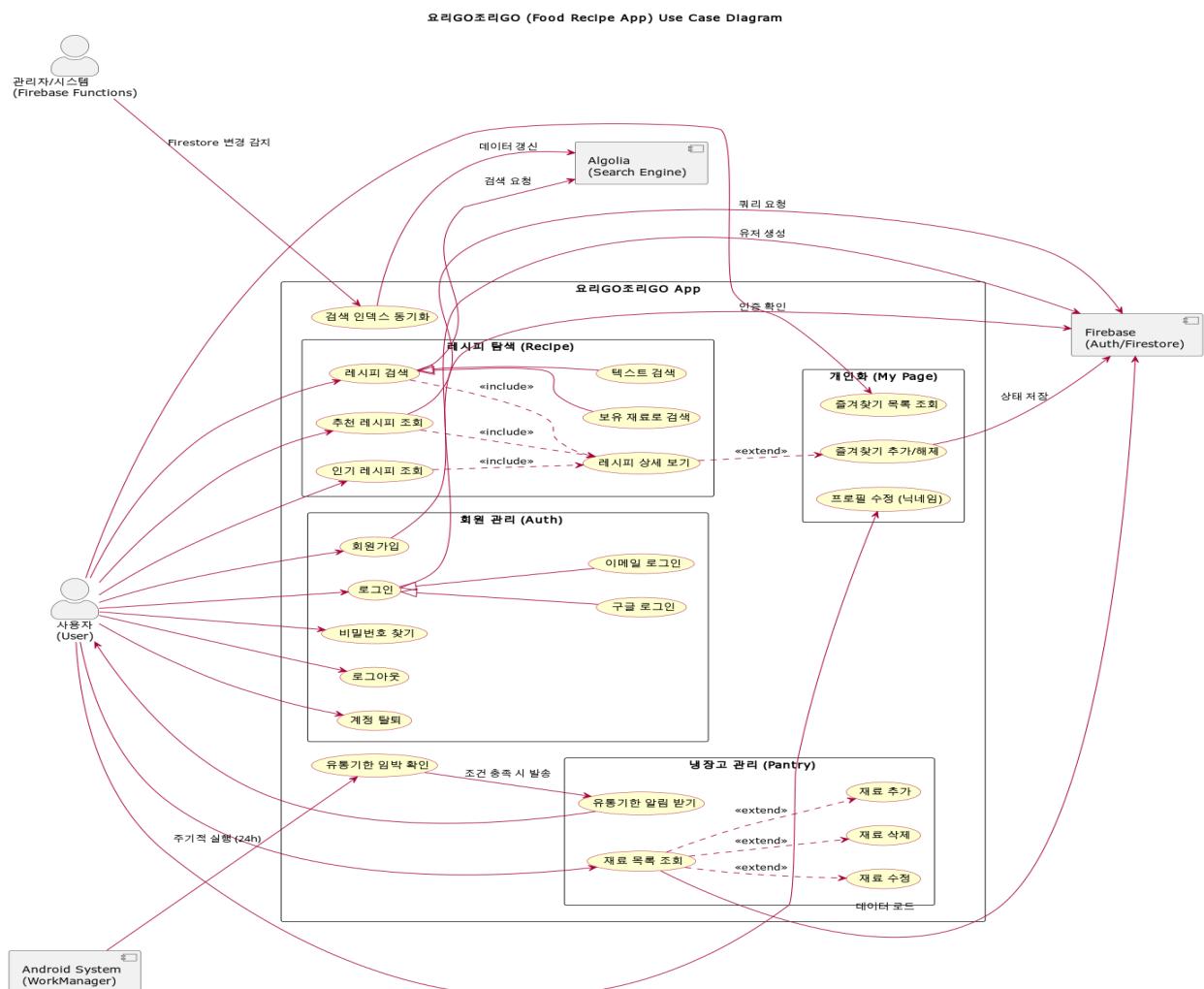
Repository Pattern: 데이터 접근 계층을 추상화하기 위해 PantryRepository 등을 두어 Presenter 가 구체적인 데이터소스 (Firestore , Algolia) 에 종속되지 않도록 설계했다. 물리적으로는 각 기능별 패키지(pantry, search) 등 내에 위치하지만, 논리적으로는 데이터처리 를 전담하는 Model 계층 의 역할을 수행한다.

State Management: ViewModel(AuthViewModel , SearchViewModel) 을 함께 도입하여, 화면 회전 이나 프래그먼트 전환시에도 사용자 인증 정보나 검색 결과 데이터가 소실되지 않고 안전하게 유지하도록 보완하였다.

Background Processing: UI와 무관하게 실행되어야 하는 유통기한 체크 로직은 Worker 클래스 (ExpirationCheckWorker)로 분리하여 시스템의 안전성을 높였다.

2) 시스템 기능 및 유스케이스 다이어그램

사용자와 시스템간의 상호작용을 정의하기 위해 주요 기능을 모듈별로 구조화 했다.



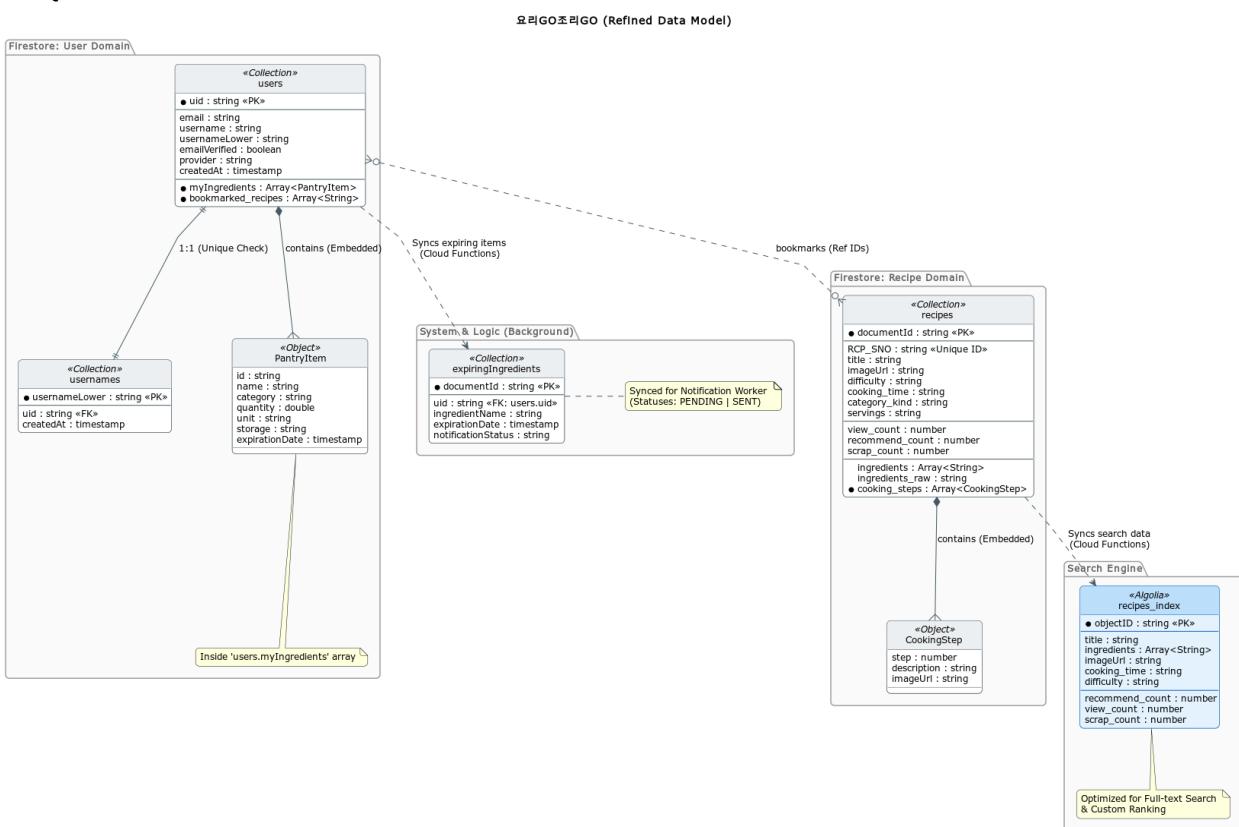
1. **사용자(User)**: 이메일 또는 구글 로그인 계정으로 로그인하여 레시피를 검색하고, 자신의 냉장고 재료를 관리하며, 개인화된 서비스(즐겨찾기) 등을 이용한다.

2. 시스템(System):

- 백그라운드 작업**: Android WorkManager 가 24시간 주기로 실행되어 유통기한 임박 재료를 탐색하고 알림을 발송한다.
- 데이터 동기화**: Firebase Functions 가 Firestore 의 데이터 변경을 감지하여 Algolia 검색 인덱스를 갱신함으로써 검색 데이터의 실시간성을 보장한다.

3) 데이터베이스 다이어그램 (ERD)

NoSQL 데이터베이스인 Cloud Firestore를 사용하여 유연하고 조회 성능에 최적화된 데이터 모델을 설계하였다.



User 컬렉션: 사용자별 고유 정보와 개인화 데이터를 저장한다. 특히 myIngredients (냉장고 재료)와 bookmarked_recipes (즐겨찾기) 를 하위 컬렉션이 아닌 문서 내 배열(Array) 필드로 설계하여, 단일 쿼리로 사용자 화면에 필요한 모든 정보를 빠르게 로딩할 수 있도록 최적화 하였다.

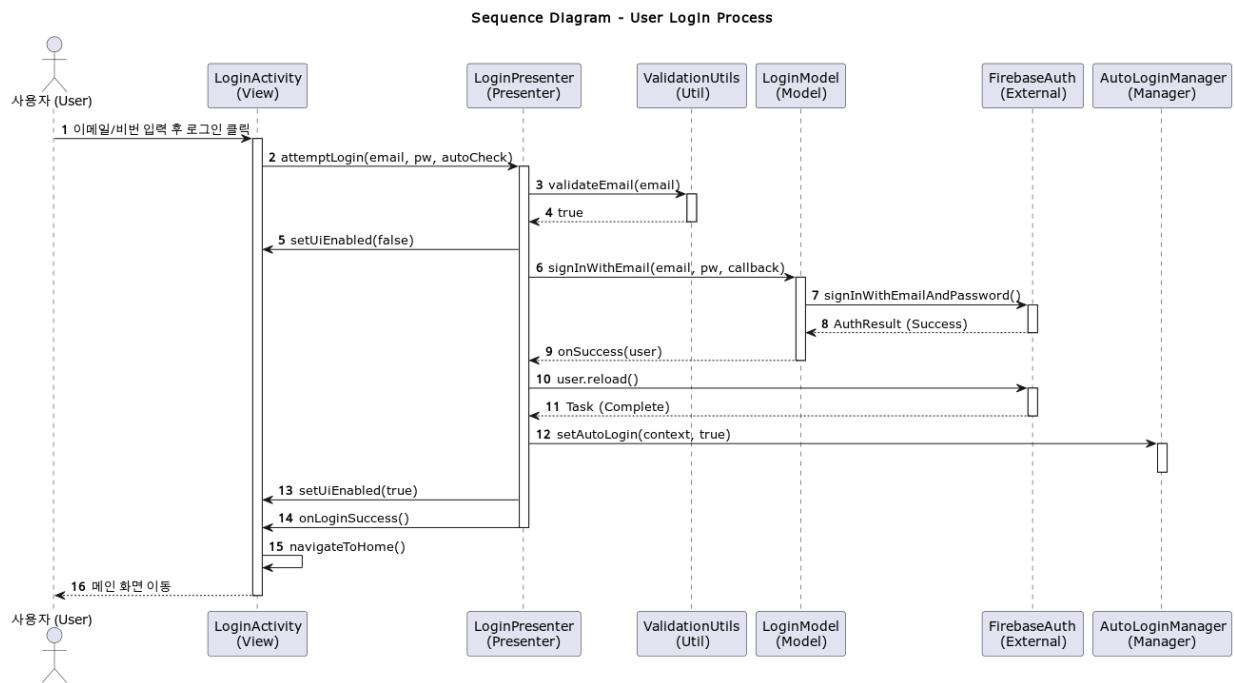
Recipes 컬렉션: 10만건의 만개의 레시피 무료데이터+ 웨크롤링한 cooking_steps (요리순서) 데이터를 저장하며, 요리 순서와 재료는(ingredients) 는 구조화된 객체 배열로 저장된다. 이 데이터는 Algolia 검색엔진과 동기화 하며 고성능 검색을 지원한다.

ExpiringIngredients 컬렉션: 알림 발송 효율성을 위해 별도로 설계된 컬렉션이다. Cloud Functions 트리거를 통해 사용자의 전체재료중 '유통기한 설정된 재료' 만 이 컬렉션으로 복제 및 동기화되며, 백그라운드 워커는 이 컬렉션만 조회하여 시스템 부하를 줄인다.

4) 주요기능 시퀀스 다이어그램

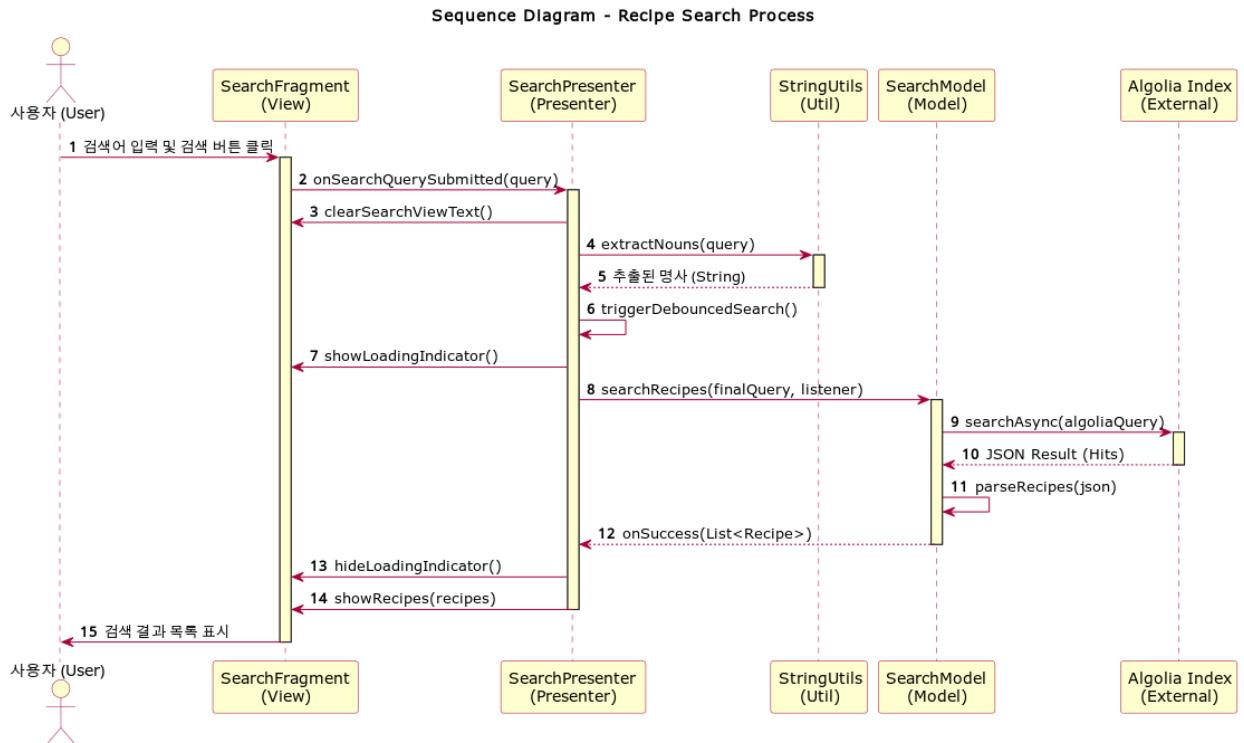
핵심 비즈니스 로직의 실행 흐름과 객체 간의 메시지 전달 과정을 시퀀스 다이어그램으로 정의했다.

1. 사용자 로그인 프로세스



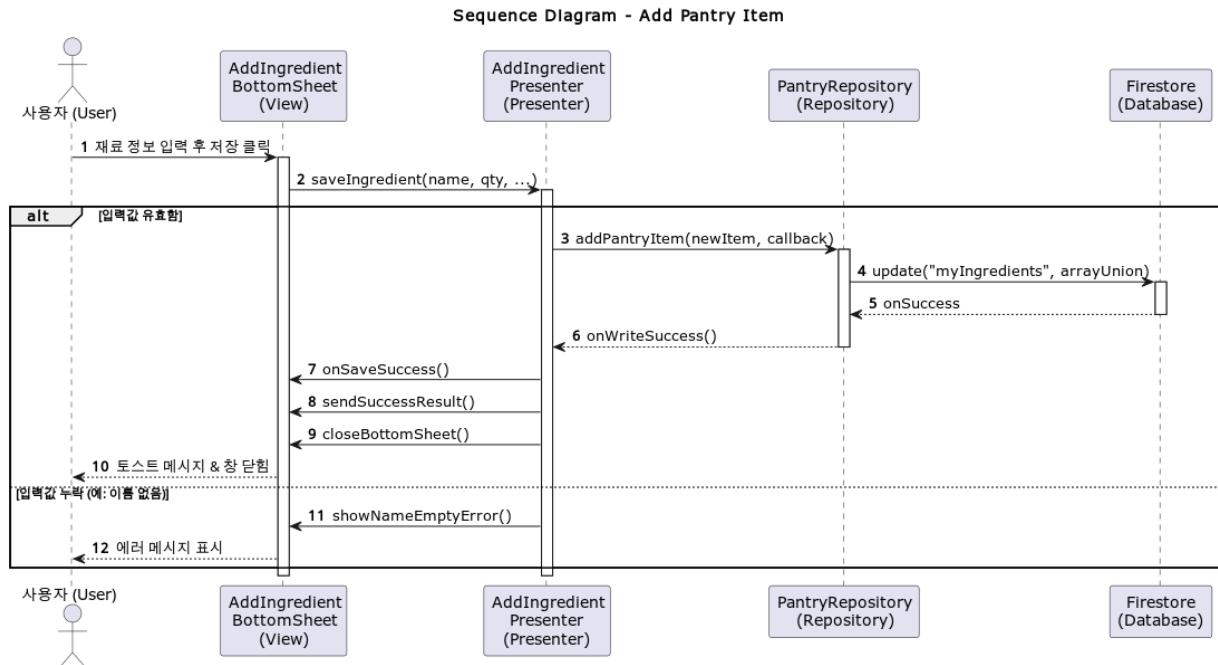
사용자가 로그인 정보를 입력하면 LoginPresenter 는 ValidationUtils를 통해 이메일 형식을 검증한다. 검증 통과시 LoginModel 이 FirebaseAuth 인증을 요청하고, 성공 응답을 받으면 AutoLoginManager를 통해 세션 정보를 로컬에 저장한 뒤 메인 화면으로 전환한다.

2. 자연어 기반 레시피 검색 프로세스



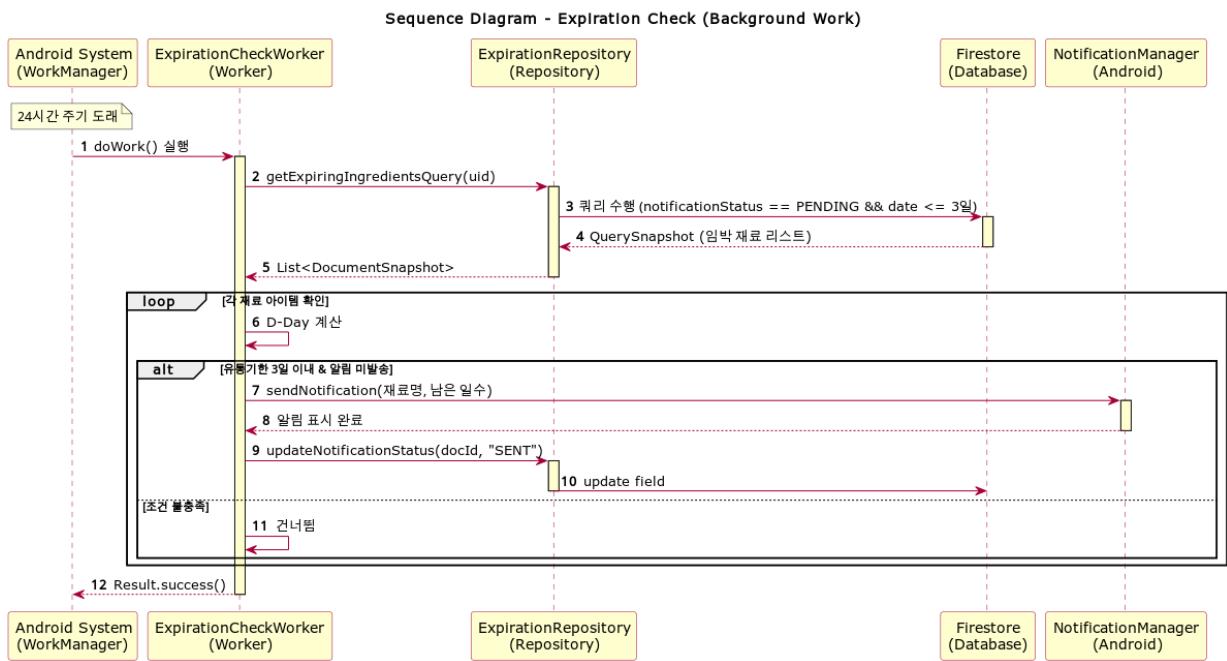
사용자가 “맛있는 김치찌개” 와 같이 자연어로 검색하면 SearchPreseneter 는 StringUtils(Okt 형태소 분석기)를 호출하여 불용어를 제거하고 핵심 명사(“김치찌개”)만 추출한다. 정제된 키워드는 SearchModel을 통해 Algolia Search API 로 전달되며, 오타 보정 및 랭킹이 적용된 검색 결과를 반환받아 UI에 표시한다.

3. 냉장고 재료 추가 프로세스



사용자가 재료 정보를 입력하고 저장을 요청하면 AddIngredientPresenter 가 유효성을 검사한다. 이후 PantryRepository 를 통해 Firebase 의 users 컬렉션 문서 내 myIngredients 배열에 arrayUnion 연산을 수행하여 데이터를 추가한다. 데이터 저장이 완료되면 콜백을 통해 UI를 갱신하고 토스트 메시지를 출력한다.

4. 유통기한 알림 프로세스 (백그라운드)



사용자의 개입 없이 Android WorkManager 에 의해 24시간마다 ExpirationCheckWorker 가 백그라운드에 실행된다. Worker는 ExpirationCheckWorker를 통해 notificationStatus가 'PENDING' 상태이면서 유통기한이 3일 이내로 남은 재료를 쿼리한다. 조건에 맞는 재료가 발견되면 NotificationManager를 통해 시스템 푸시 알람을 발송하고, 해당 재료의 상태를 'SENT'로 업데이트 하여 중복 알람을 방지한다.

2.4 제작과정 및 최종 결과물

제작과정

3학년 1학기 ‘소프트웨어 공학’ 및 ‘졸업작품 종합설계’ 과목을 통한 체계적인 기획 단계를 거쳐, 2학기 부터 본격적인 구현 및 형상 관리를 진행하였다. 이 과정에서 초기 계획과 실제 구현 사이의 간극을 조정하며 시스템을 최적화하였다.

1단계: 요구사항 분석 및 설계 (3월~6월)

프로젝트 착수 초기에는 ‘냉장고 파먹기’, ‘냉장고를 부탁해’라는 핵심아이디어를 구체화하기 위해 관련 깃허브 조사와 시장 조사를 선행하였다. ‘소프트웨어 공학’ 과목과 연계하여 시스템의 안정성을 확보하기 위해 다음과 같은 산출물을 작성하고 설계를 확정하였다.

- **요구사항 명세:** 사용자의 식습관 및 프로필, 보유재료 기반 레시피 추천등 핵심 기능을 정의하고 유스케이스(Use Case)를 도출 하였다.
- **WBS 및 테스트 계획:** 프로젝트 일정을 WBS로 세분화하고, 단위/통합 테스트 시나리오를 사전에 계획하여 리스크를 관리 하였다.
- **UI/UX 초안 설계:** Figma를 활용하여 화면 흐름도(Flow Chart) 와 와이어 프레임을 작성 하였다.

2단계: 데이터 구축 및 아키텍처 설계(8월 말~9월)

여름방학 이후 본격적인 개발에 앞서, 핵심 자산인 레시피 데이터 확보와 개발 환경 구축에 집중하였다. 협업을 위해 GitHub 기능별(Feature) 브렌치 전략을 도입하고, Android Studio 와 Firebase 연동을 완료하였다.

- **데이터 소스 선정 및 가공:** 공공데이터 포털의 레시피 데이터와 KADX 무료레시피 데이터인 ‘만개의 레시피’를 비교 분석하였다. 그결과, 데이터의 다양성과 품질 면에서 우수한 ‘만개의 레시피’를 최종 선정하였다.
- **Python 크롤링 및 전처리:** 원본 데이터에 누락된 ‘요리순서(Cooking Steps)’ 필드를 보완하기 위해 Python을 활용한 웹 크롤링을 수행하였다. 수집된 비정형 데이터를 정제 하고 구조화하여 기존 20만건의 레시피 데이터셋을 10만 건의 고품질 레시피 데이터셋으로 구축하였다.
- **회원 시스템 구현 및 MVP 패턴 도입:**
 - 1) Firebase Auth를 연동하여 로그인, 회원가입, 비밀번호 찾기 등 핵심 회원 관리 기능을 우선적으로 구현하였다.
 - 2) 이 과정에서 Activity에 비즈니스 로직이 집중되어 코드가 비대해지는 문제를 식별하였다. 이를 해결하기 위해 아키텍처 패턴 도입을 검토하였으며, 팀원들의 러닝 커브와 유지보수 효율성을 고려했을 때 MVVM보다는 MVP(Model-View_Presenter) 패턴이 적합하다고 판단하였다.

3단계: 아키텍쳐 고도화 및 기능완성 (10월~11월)

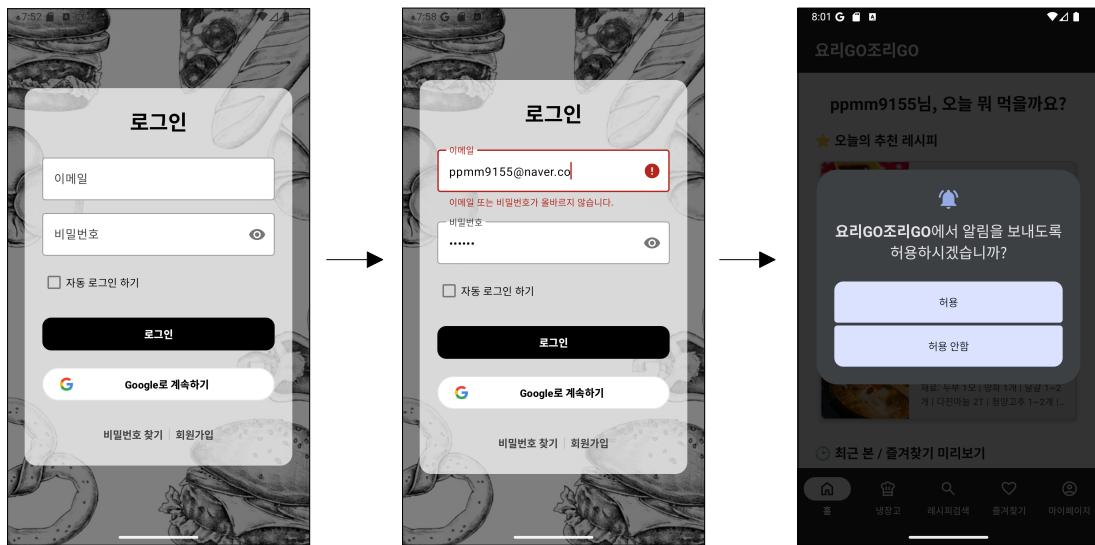
설계된 아키텍쳐 기반으로 시스템의 구조적 완성도를 높이고 사용자 편의 기능을 대폭 강화하였다. 특히 11월 중순 예정된 ‘캡스톤 디자인 전시회’ 시연을 목표로 개발 및 안정성화에 집중하였다.

- **내비게이션 구조 개선(Jetpack Navigation 도입):** 기존 Intent 기반의 화면 전환 방식은 액티비티가 많아 질수록 관리가 복잡해지고 메모리 리소스 소모가 크다는 단점이 있었다. 이를 해결하기 위해 메인 화면(MainActivity)에 Jetpack Navigation Componet를 도입하여 Single Activity Architecture를 구현하였다. 이를 통해 프레그먼트 간의 이동 경로(Navigation Graph)를 시각적으로 관리하고, 바텀 내비게이션 과의 연동성을 획기적으로 개선 하였다.
- **핵심 기능 구현(10월~11월초)**
 - 1) **레시피 검색:** Algolia 엔진과 Okt 형태소 분석기를 연동하여 자연어 검색 및 재료 기반 필터링 기능을 완성하였다.
 - 2) **나만의 냉장고:** 식재료의 추가-삭제(Swipe) - 수정(Bottom Sheet) 프로세스를 구현하고, Firestore 와 실시간 동기화되도록 처리 하였다.
 - 3) **유통기한 알림:** WorkManger 를 활용하여 앱 종료 상태에서도 24시간 주기로 유통기한 임박 재료를 체크하는 백그라운드 알림 시스템을 구축하였다.
 - 4) **개인화 서비스:** 즐겨찾기 및 마이페이지(프로필 수정,로그아웃 등) 기능을 통해 사용자 맞춤형 경험을 제공하였다.
- **시스템 안정화 및 최종 점검(11월12일~11월19일):** 11월19일~20일 캡스톤 디자인 전시회를 압두고 약 열흘간 집중적인 안정화 기간을 가졌다. 이 기간에는 신규 기능 개발을 동결하고, Android Studio Lint 통해서 정적코드검사하거나 사용자 시나리오 기반의 반복적인 수동 테스트를 병행하여 앱의 구동 안정성을 극대화하였다.

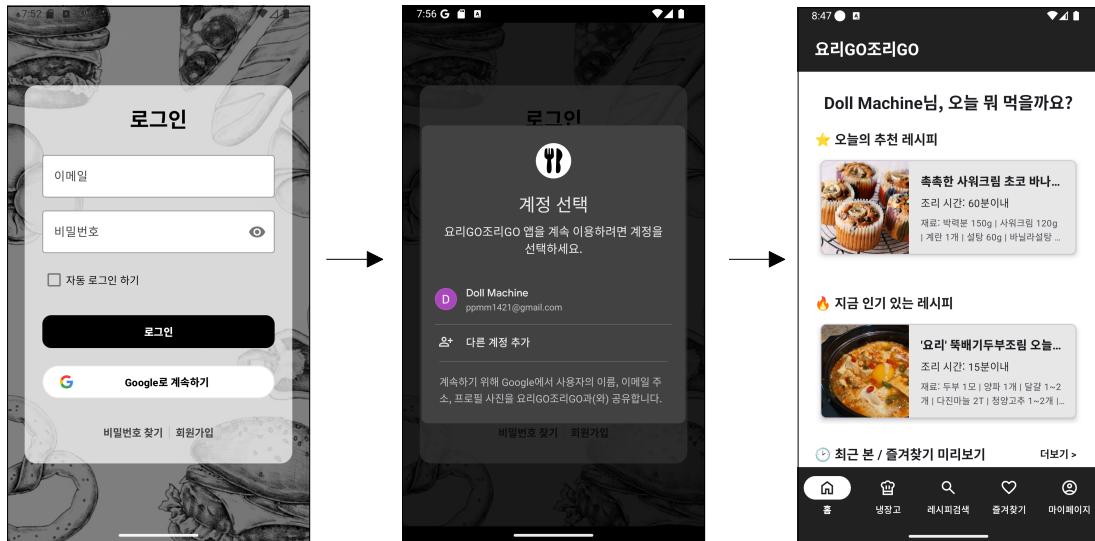
최종 결과물

제작 과정을 통해 완성된 ‘요리GO조리GO’ 애플리케이션의 주요 화면 구성과 핵심 기능은 다음과 같다. 사용자(UX)을 고려하여 직관적인 UI를 설계하였으며, 각 기능은 안드로이드 최신 라이브러리와 아키텍쳐 패턴을 적용하여 구현되었다.

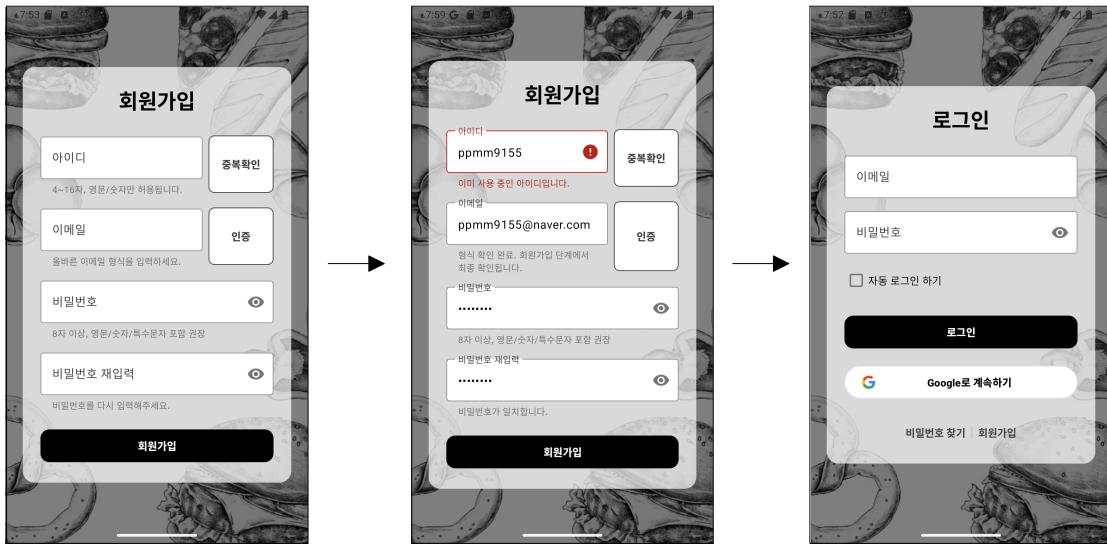
1) 로그인 및 회원가입: 앱 실행시 최초 진입 화면으로, 사용자 편의를 위해 이메일 로그인 외 Google 소셜 로그인을 지원한다. AutoLoginManager를 통해 자동 로그인을 처리하여 불필요한 재 로그인 과정을 생략하였다.



이메일/비밀번호 방식

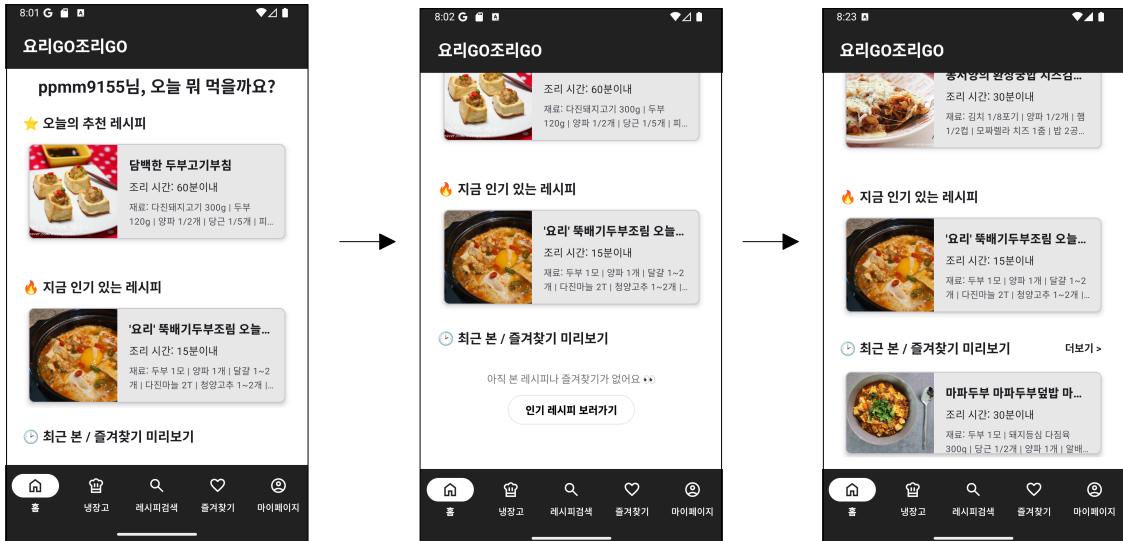


구글 로그인 방식



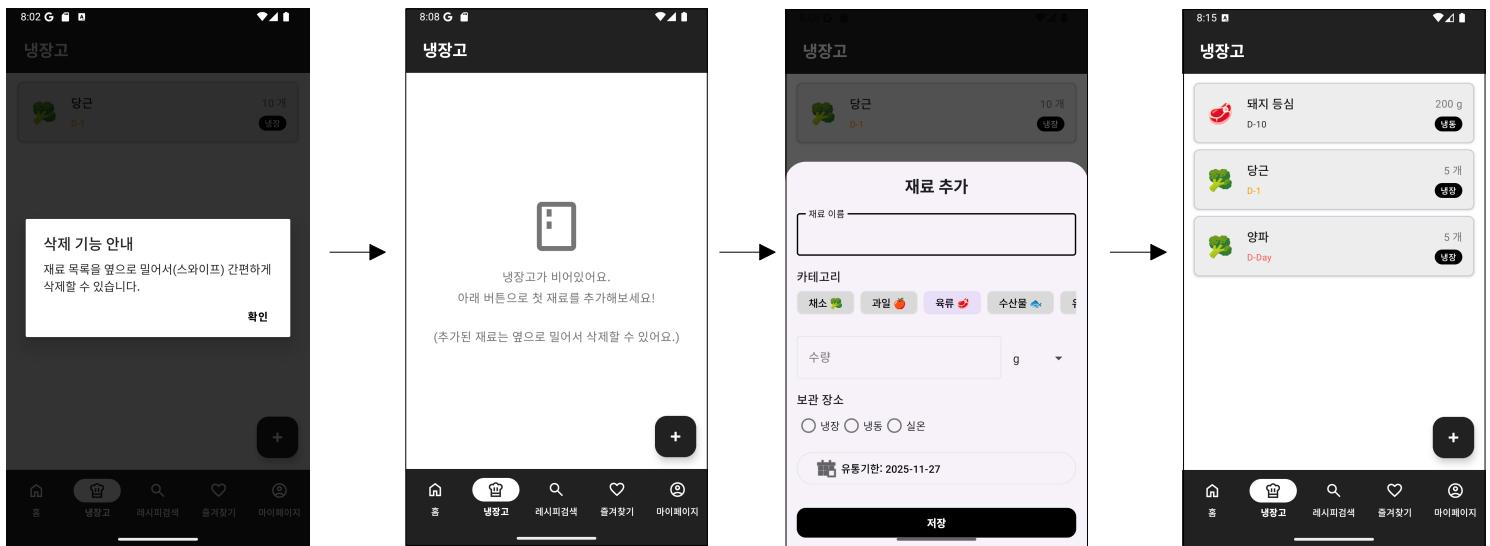
회원가입 방식

2) 메인 홈 화면: 사용자 닉네임을 포함한 개인화된 환영 메시지와 함께, 상황별 추천 레시피를 카드 형태로 제공한다. “오늘의 추천 레시피” (냉장고, 즐겨찾기 기반), “지금 인기 있는 레시피” (조회수, 즐겨찾기, 스크랩 수 기반), 상하 스크롤(NestedScrollView)과 좌우 스크롤(RecyclerView)을 결합하여 다양한 정보를 효율적으로 배치하였다.

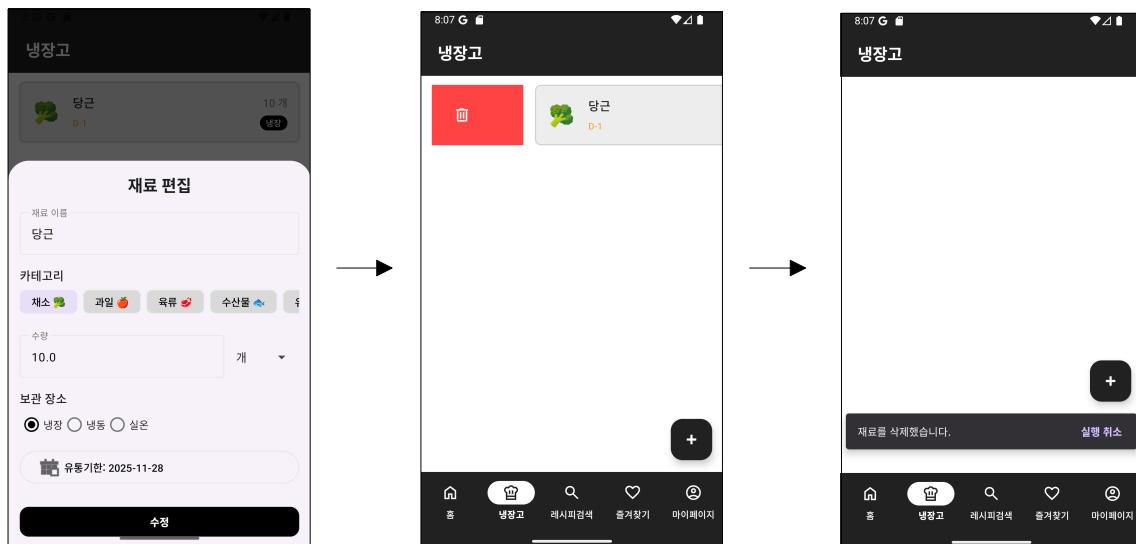


3) 나만의 냉장고: 사용자가 보유한 식재료를 냉장/냉동/실온으로 분류하여 식재료 관리를 한다. 특히 ItemTouchHelper를 적용하여, 리스트를 스와이프하는 동작만으로 재료를 직관적으로 삭제할 수 있다.

4) 재료 추가 및 수정: 화면 이동 없이 현재 화면 위에 뜨는 Bottom Sheet Dialog를 활용하여, 사용자의 작업 흐름을 끊지 않고 빠르게 재료 정보를 입력하거나 수정할 수 있다.

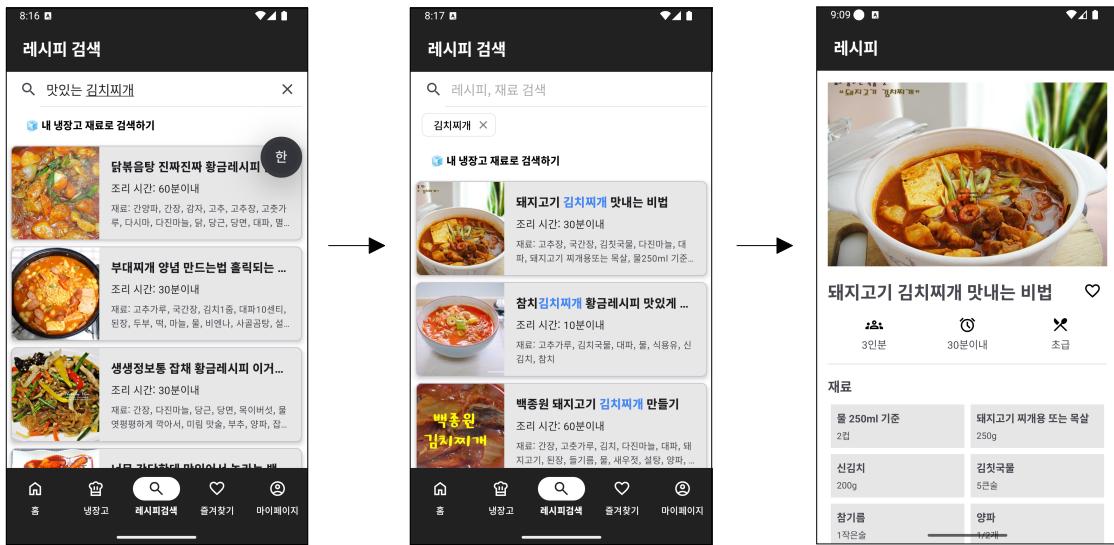


+ 버튼을 눌러서 재료 추가 및 해당 재료를 누르면 재료 편집이 가능하다.

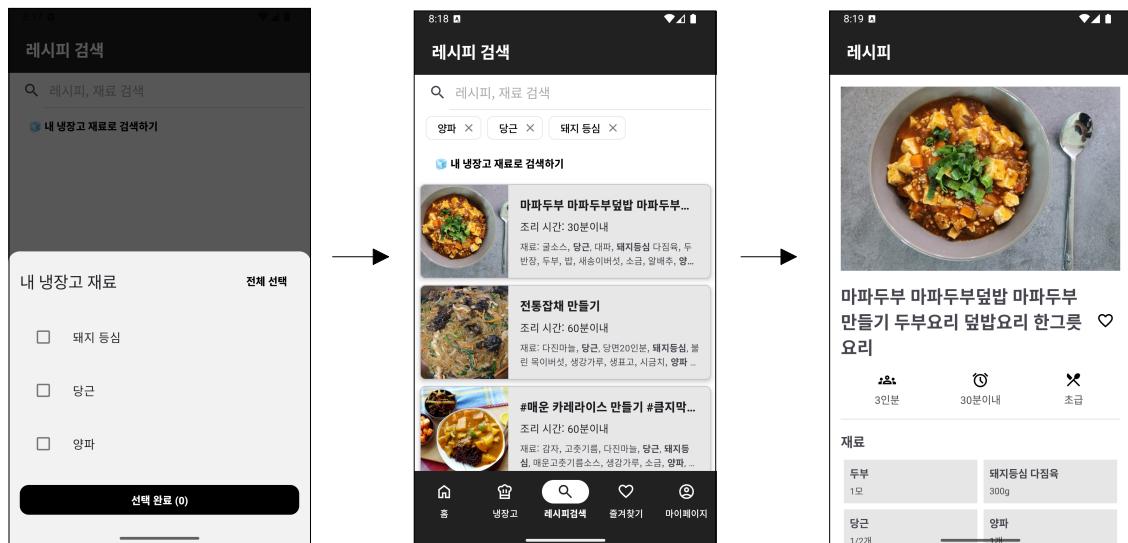


스와이프로 식재료 삭제 시 하단에 스낵바가 나와서 안내 문구 및 실행 취소가 가능하다.

5) 레시피 검색 및 상세 정보: Algolia 검색엔진과 Okt 형태소 분석기를 탑재하여, 자연어 검색 시에도 핵심 키워드만 추출하여 정확한 결과를 제공한다. ‘내 냉장고 재료로 검색하기’ 기능을 통해 보유 재료로 만들 수 있는 요리를 즉시 필터링 할 수 있다.

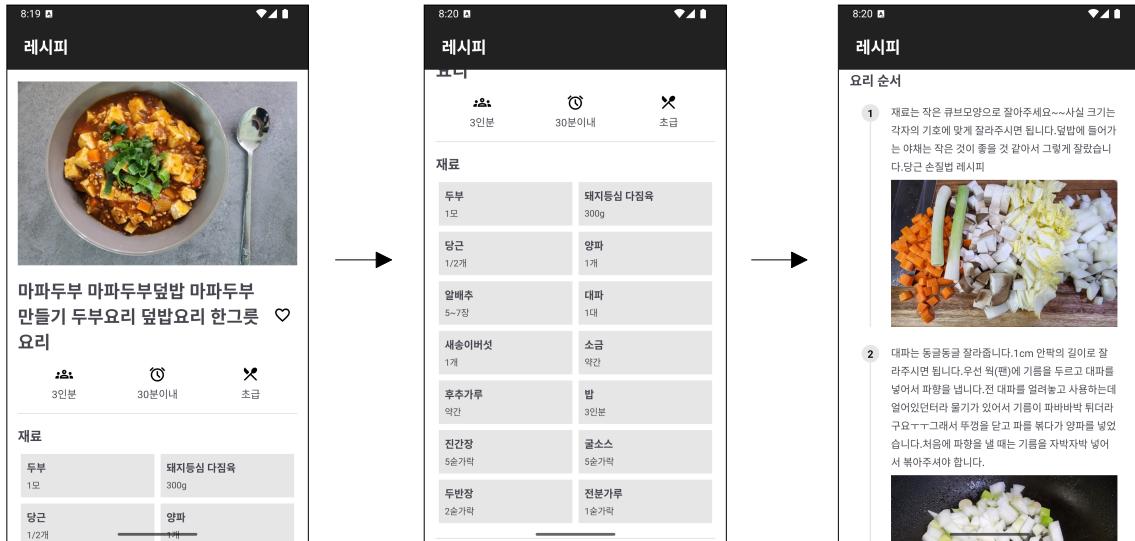


직접 입력 방식으로 레시피 검색 결과로 Algolia 하이라이팅 기능을 사용해 강조된 모습이다.

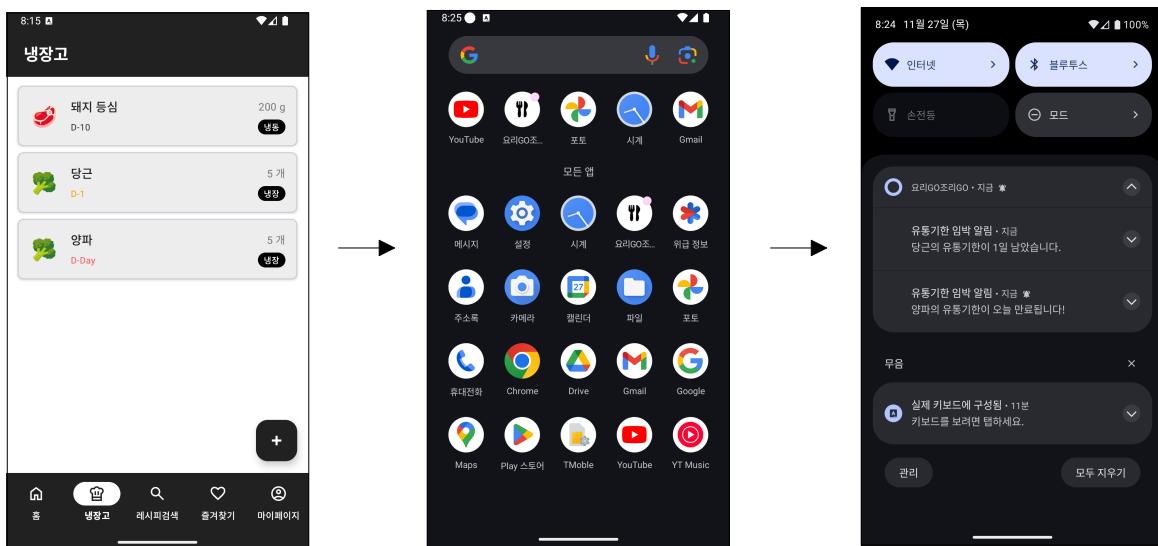


내 냉장고 재료로 검색 시에도 chip 형태로 “양파”, “당근”, “돼지 등심” 관련 재료나 레시피 제목이 검색 결과로 나온다.

6) 레시피 상세화면: 10만건 의 만개의 레시피 데이터 셋 으로 대표그림,레시피 이름,요리 난이도 및 요리시간 ,요리 순서등 을 제공한다.

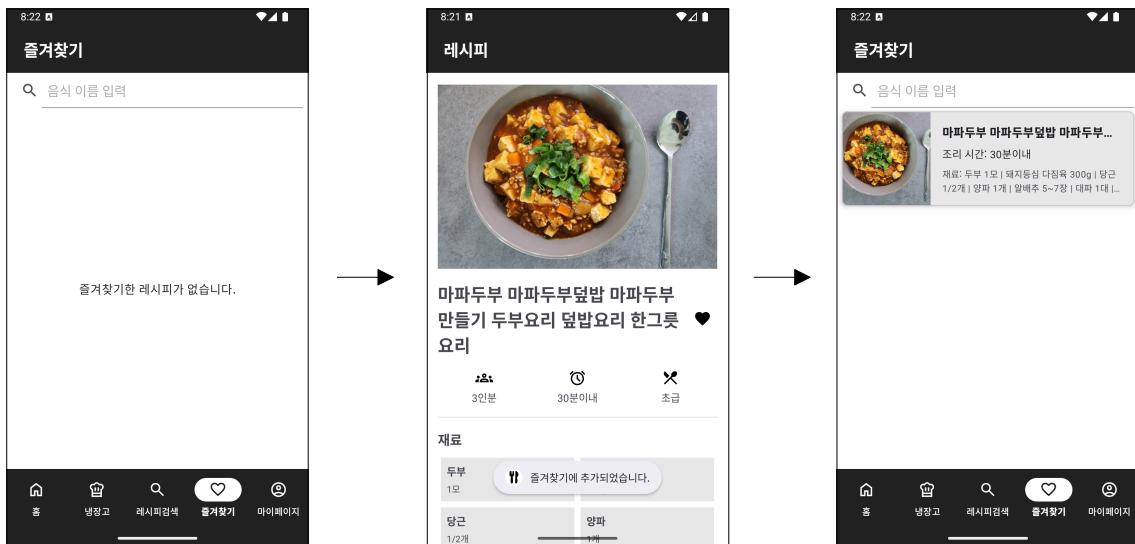


7) 백그라운드 유통기한 알림: 앱이 종료된 상태에서도 WorkManager 가 24시간 주기로 작동하여, 유통기한 3일 이내로 남은 식재료가 있을 경우 시스템 푸시 알림을 전송한다.

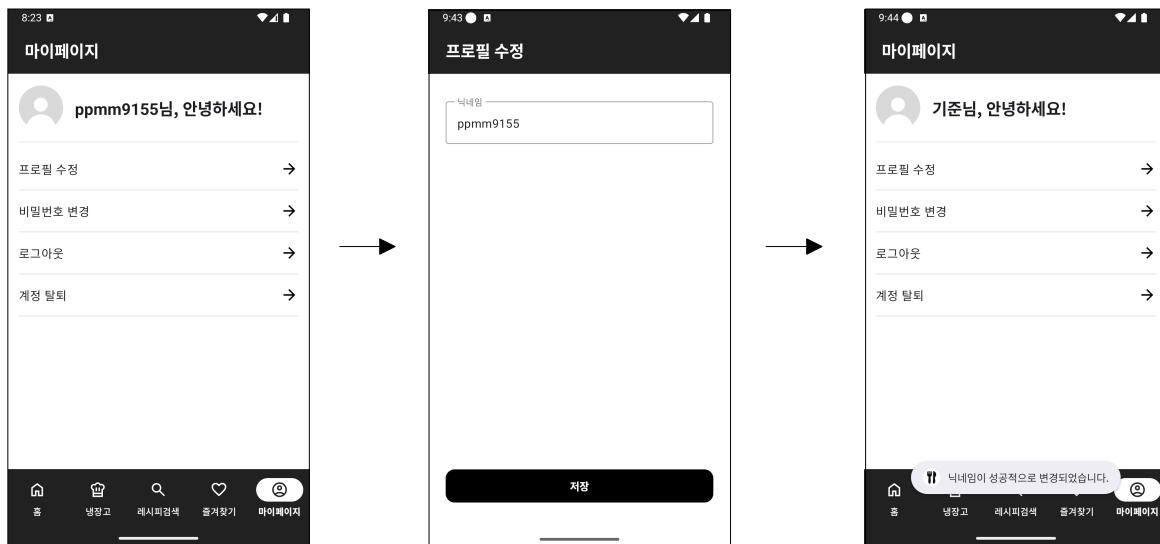


알람을 클릭 시 요리GO조리GO 홈 화면으로 이동하게 된다.

8) 즐겨찾기: 사용자가 마음에 드는 레시피를 보관하고 관리하는 개인화 공간이다. 단순한 로컬 저장이 아닌 Firebase 유저 DB와 레시피 DB의 즐겨찾기 count 가 증가, 감소 하며 Algolia 도 동시에 즐겨찾기 count 가 증가, 감소 해서 실시간 동기화가 되며 기기를 변경해도 데이터가 유지된다. 현재 즐겨찾기 검색 기능은 미구현 상태이다.



9) 마이 페이지 및 계정관리: 사용자의 프로필 정보를 확인하고, 계정을 관리하는 통합 설정 공간이다. 닉네임 수정뿐만 아니라 비밀번호 변경, 로그아웃, 계정 탈퇴 등 회원 생명주기와 관련된 모든 기능을 수행한다.



2.5 프로젝트 결과 시험 및 자체평가

1) 시험 내역

구분	테스트 항목	시나리오	예상 결과	결과
검색	자연어 검색 및 형태소 분석	입력값: 얼큰한 김치찌개 만드는 법 1. 검색 창에 자연어 문장 입력 2. 검색 버튼 클릭	1. Okt 분석기가 불용어를 제거하고 핵심 명사 “김치찌개”만 추출 2. Algolia 엔진이 해당 키워드가 포함된 레시피 리스트 반환	성공
검색	보유 재료 기반 필터링 검색	보유 재료: 돼지고기, 두부, 김치 1. ‘내 냉장고 재료로 검색’ 클릭 2. 검색된 결과 확인	1. 현재 사용자의 PantryItem 리스트를 자동으로 쿼리 필터로 변환 2. 위 재료들이 포함된 레시피만 필터링되어 노출	성공
냉장고	재료 추가 및 유효성 검사 제스처 삭제 및 실행 취소	입력값: (이름: 우유, 수량: 1.5, 단위:L, 유통기한: 2025-11-26) 1. BottomSheet에서 정보 입력 후 저장 2. 대상: 리스트의 첫 번째 아이템 3. 아이템을 왼쪽으로 스와이프 4. 나타난 스낵바의 ‘취소’ 버튼 클릭	1. Firestore 유저 DB에 실시간 저장 2. 화면 새로고침 없이 리스트에 즉시 추가됨 3. D-Day가 자동으로 계산되어 표시됨 4. 스와이프 시 빨간색 배경/휴지통 아이콘 노출 5. 삭제 후 ‘취소’ 클릭 시 데이터가 즉시 복구됨	성공
알림	백그라운드 유통기한 알림	조건: 유통기한이 3일 남은 재료 등록 1. 앱을 완전히 종료 2. 24시간 경과 또는 WorkManager 강제 트리거	1. 앱이 꺼져 있어도 시스템 푸시 알림 도착 2. 알림 클릭 시 앱의 메인 화면으로 진입	성공
동기화	즐겨찾기 실시간 방영	대상: 레시피 1. 상세 화면에서 하트(찜) 버튼 클릭 2. 하단 ‘즐겨찾기’ 탭으로 이동하여 확인 3. Algolia 대시보드에서 해당 레시피 확인	1. 클릭 즉시 하트 아이콘 색상 변경 2. 즐겨찾기 탭 리스트에 ‘마파두부’가 즉시 추가됨 (Firestore 실시간 리스너) 3. CloudFunctions 트리거에 의해 Algolia 인덱스의 recommend_count가 +1 증가되어 동기화됨	성공

특이 사항: 초기 테스트 시, Okt 형태소 분석기를 메인 스레드에서 초기화할 때 약 1.5초의 앱 멈춤 현상이 발생함.

조치 내용: Application 클래스의 onCreate() 시점에서 CompletableFuture를 활용하여 별도의 백그라운드 스레드에서 Okt 분석기를 미리 로딩하도록 비동기 처리하여 검색 지연 시간을 제거함.

2) 자체 평가

1. 목표 달성도

기획 단계에서 수립한 요구사항 명세, UI/UX 초기 초안 설계 대비, 핵심 기능인 식재료 관리, 자연어 기반 레시피 검색, 즐겨찾기, 마이페이지 등 모든 주요 기능이 98% 이상 구현 완료 하였다. 10만 건의 대용량 레시피 데이터를 구축하고 이를 실제 앱 서비스에 연동함으로써 상용 앱 수준의 데이터 규모를 확보하였다.

2. 기술적 우수성 및 차별점

- 지능형 검색 시스템 구축:** 기존 데이터베이스의 단순 문자열 매칭(LIKE 연산)과 Firestore 복합 쿼리의 제약을 극복하기 위해 Algolia 검색 엔진과 Okt 형태소 분석기를 결합한 하이브리드 검색 시스템을 구축하였다. 이를 통해 불용어 제거 및 핵심 키워드 추출이 가능해졌으며, 동의어 처리와 Algolia 하이라이팅 기술 등 검색 속도와 정확도를 획기적으로 개선하였다.
- 안정적인 시스템 아키텍처:** 안드로이드 MVP 패턴을 전면 도입하여 UI와 비즈니스 로직을 철저히 분리함으로써 코드의 유지보수성과 테스트 용이성을 높였다. 또한 WorkManager API를 활용하여 앱이 종료된 상태에서도 유통기한 임박 알림이 안정적으로 발송되도록 구현하여 안드로이드 백그라운드 정책을 준수하였다.

3. 개선점 및 향후 과제

- 기술 스택의 현대화:** 현재 프로젝트는 익숙함을 위해 Java와 러닝커브가 비교적 낮은 MVP 패턴을 채택하였으나, 향후 유지보수 효율성을 높이기 위해 Kotlin 언어로의 마이 그레이션을 고려한다. 또한 MVVM 패턴과 Jetpack Compose를 도입하여 UI 상태 관리를 일원화하고, 선언형 UI를 통해 보일러플레이트 코드를 획기적으로 줄이는 구조적 개선을 목표로 한다.
- 백엔드 인프라의 확장성 확보:** Firebase와 Algolia의 조합은 빠른 개발에 유리했으나, 복잡한 관계형 데이터나 대규모 트래픽 처리 시 비용 및 유연성 측면에서 한계가 있을 수 있다. 추후 Spring Boot나 Node.js 기반의 자체 REST API 서버를 구축하여 벤더 종속성을 해소하고 데이터 처리의 자율성을 높이는 방향으로 고도화할 필요가 있다.
- 검색 및 정렬 기능의 세분화:** 현재의 검색 시스템은 정확도 위주의 결과를 제공하고 있다. 향후 사용자의 다양한 니즈를 충족시키기 위해 유통기한 임박 순, 조리 시간순, 조회 수, 즐겨찾기 수 등 다차원적인 정렬 필터를 도입하여 검색 경험을 최적화할 예정이다.
- 커뮤니티 및 추천 시스템 고도화:** 단순 단방향 정보 제공을 넘어, 사용자가 레시피를 공유하는 커뮤니티 기능을 추가하고, 사용자 행동 로그 기반의 머신러닝 추천 모델을 도입하여 플랫폼의 지속 가능성 확보할 것이다.

- 식재료 등록 프로세스의 자동화:** 현재 텍스트 기반의 직접 입력 방식은 다수의 식재료를 등록할 때 사용자 피로도를 유발할 수 있다. 향후 Google ML Kit나 TensorFlow Lite를 활용한 객체 인식 및 바코드 스캔 기능을 도입하여, 카메라 촬영만으로 식재료 명과 유통기한 정보를 자동으로 추출(OCR)하고 등록하는 시스템으로 고도화할 계획이다.

2.6 개인별 공헌 내용

구분	학번	성명	주요 공헌내용
팀장	2106066	하종수	<ul style="list-style-type: none"> 주요 역할 팀 리더로서 프로젝트의 전반적인 방향성을 제시하고, 특히 사용자의 서비스 접근에 필수적인 인증(Authentication) 파트를 총괄 개발했습니다. 또한, 팀원의 코드 리뷰 및 병합을 책임지며 프로젝트의 코드 품질과 안정성을 확보하는 데 핵심적인 역할을 수행했습니다. 핵심 기능 개발 (인증 파트) 소셜 게스트 로그인: 구글 소셜 로그인 및 별도의 회원가입이 필요 없는 게스트 로그인 기능을 구현하여 사용자의 진입 장벽을 낮췄습니다. 계정 관리: 사용자가 비밀번호를 잊었을 경우 재설정할 수 있는 '비밀번호 찾기' 기능을 개발 사용자 피드백: 로그인 실패시, 실패 원인을 명확히 알려주는 토스트 메시지를 추가하여 ux를 개선 아키텍처 및 리팩토링 MVP 아키텍처 전환: 담당한 인증 관련 주요 화면(메인, 비밀 번호 찾기 등)을 MVP(Model-View-Presenter) 디자인 패턴으로 전환하여 코드의 유지보수성과 테스트 용이성을 향상 코드 표준화: 팀 전체의 코드 가독성을 높이기 위해 ID 명명 규칙을 카멜 케이스로 통일하는 리팩토링을 진행 UI/UX 개발 레이아웃 설계: 앱의 메인 레이아웃과 하단 네비게이션 바의 기본 구조를 설계하고 구현 디자인 구현: 사용자가 처음 마주하는 로그인 화면의 전체적인 레이아웃을 디자인하고 XML로 구현

구분	학번	성명	주요 공헌 내용
팀원	2006094	박기준	<ul style="list-style-type: none"> 주요 역할 프로젝트의 핵심 기능 대부분을 총괄하여 설계 및 개발 했으며, 안정적인 앱 아키텍처를 구축하고 고도화하는 역할을 수행했습니다. 특히 Algolia 검색 엔진 연동, WorkManager를 이용한 백그라운드 알림 등 기술적으로 복잡한 문제들을 해결하며 앱의 완성도를 비약적으로 향상시켰습니다. 또한, 프로젝트의 모든 과정을 상세히 기록하는 문서화 작업을 전담했습니다. 핵심 기능 개발 (앱 기능 총괄) 냉장고 관리: 사용자가 보유한 식재료를 추가/편집/삭제하는 CRUD 기능 전체를 구현. 특히, 연속 스와이프 삭제 시에도 앱이 충돌하지 않도록 안정성을 강화. 레시피 검색: Algolia 검색 엔진을 연동하고, 검색어 하이라이팅, 사용자 보유 재료 기반 조합 검색 등 고급 검색 기능을 구현. 레시피 상세 정보: 레시피의 상세 재료, 이미지, 요리 순서 등을 보여주는 상세 화면 전체를 개발. 즐겨찾기/홈: 사용자가 레시피를 즐겨찾기하는 기능을 구현하고, 이 데이터가 Firestore 및 Algolia에 실시간으로 동기화(추천 수 증감)되는 로직을 완성. 홈 화면에 최근 본/즐겨찾기 목록 미리보기를 구현. 아키텍처 및 시스템 설계 전사적 MVP 전환: 스플래시, 로그인, 회원가입 등 앱의 거의 모든 화면을 MVP 아키텍처로 전환하여 프로젝트의 구조적 일관성과 확장성을 확보. 중앙 인증 관리: AuthViewModel을 도입하여 훑어져 있던 인증 관련 로직을 중앙에서 관리하도록 개선, 코드 중복을 제거하고 안정성을 높임. 백그라운드 처리: Android WorkManager를 활용하여, 앱이 실행 중이지 않을 때도 유통기한 임박 재료를 감지하고 푸시 알림을 보내는 핵심 기능을 설계 및 구현. 보안 강화: Gradle Secrets Plugin을 도입하여 민감 정보인 Algolia API 키를 소스 코드로부터 안전하게 분리. UI/UX 개선 및 디버깅 사용성 개선: 전반적인 화면에 로딩 인디케이터를 추가하여 데이터 로딩 상태를 명확히 하고, 키보드와 하단 네비게이션 바가 겹치는 등 다수의 UI 버그를 수정. 안정성 확보: 자동 로그인 로직, 스플래시 화면 표시 로직, 회원가입/로그인 과정의 엣지 케이스 버그들을 수정하여 앱의 신뢰도를 향상.

			<ul style="list-style-type: none"> 프로젝트 관리 및 문서화 데이터 파이프라인: '만개의 레시피' 무료 데이터를 정제하고 Firestore/Algolia에 일괄 업로드하기 위한 Python 데이터 파이프라인 스크립트를 작성. Git/빌드 관리: .gitignore 설정, Gradle 의존성 관리 등 프로젝트의 초기 환경을 구축하고 안정화. 상세 문서화: 프로젝트의 모든 내용을 체계적으로 정리한 Readme.md를 작성하고 지속적으로 관리하여 팀의 협업 효율과 프로젝트의 가치를 높임.
--	--	--	---

구분	학번	성명	주요 공헌 내용
팀원	2106089	정승훈	<ul style="list-style-type: none"> 주요 역할 매니저 및 캡스톤 디자인 전시회 포스터 협업, 개발된 애플리케이션의 안정성을 확보하기 위해 품질 보증 프로세스를 전담하여 수행하였다. 사용자 관점에서 발생할 수 있는 다양한 시나리오를 기반으로 기능 테스트를 했다. 품질 관리 기능 및 시나리오 테스트: 로그인, 검색, 알림 등 핵심 기능에 대한 단위 테스트와 전체 흐름을 점검하는 통합 테스트를 수동으로 수행하여 앱의 구동 안정성을 검증함. 프로젝트 기획 및 문서화 문서화 지원: 프로젝트 계획서, 요구사항 분석서 등 프로젝트의 주요 단계별 산출물 작성을 지원하고, 캡스톤 디자인 포스터 작성에 협업함.

구분	학번	성명	주요 공헌 내용
팀원	2006103	김강민	<ul style="list-style-type: none"> 주요 역할 프로젝트 기획 및 초기 시스템 및 UI/UX 설계, 프로젝트 착수 단계에서 핵심 아이디어(앱 명칭 및 콘셉트)를 도출하고 이를 개발 가능한 형태의 요구사항 명세서로 구체화 하였다. 사용자 관점의 시나리오(Use Case) 와 초기 UI 구조를 설계 협업 하여 프로젝트의 청사진을 완성하였다. 서비스 기획 및 아이디어 구체화 앱 네이밍 및 브랜딩: 냉장고 파먹기와 요리 추천 이라는 핵심 기능을 직관적으로 표현하는 '요리GO조리GO' 서비스 명칭과 브랜드 콘셉트를 제안 및 확정함 요구사항 정의: '식재료 기반 레시피 추천', 등 핵심 기능을 정의하고, 이를 바탕으로 기능 명세서를 협업 하여 작성함 사용자 경험(UI/UX) 및 시스템 초기 설계 UI 구조 및 흐름도 설계: 사용자의 앱 진입 부터 핵심 기능 사용까지의 동선을 피그마를 이용해 고려하여 화면 흐름도를 협업하여 설계함

		사용자 시나리오 구상: '식재료 등록 -> 레시피 색검 -> 요리하기'로 이어지는 사용자 여정 시나리오를 구체화하여, 개발 단계에서 발생할 수 있는 예외상황을 방지함.
--	--	---

3. 결론 및 소감

본 프로젝트 '요리GO조리GO'는 현대인의 식재료 관리 및 메뉴 선정의 어려움을 기술적으로 해결하고자 시작되었다. 개발 과정에서 MVP 아키텍쳐 와 하이브리드 검색엔진 (Algolia+Okt), 백그라운드 시스템(WorkManager) 등 안드로이드의 핵심 기술들을 적용하였다. 초기 기획 단계의 기술적 난관들을 팀원 간의 협업과 엔지니어링 접근으로 극복하며, 단순한 앱 개발을 넘어 사용자 경험(UX)과 시스템 성능을 고려한 서비스를 완성할 수 있었다.

3.1 장애요인 및 극복 사례

- 형태소 분석기 로딩으로 인한 앱 멈춤:** 검색 정확도 향상을 위해 도입한 Okt 라이브러리의 초기화 시간이 길어 앱 실행 직후 UI가 멈추는 현상이 발생하였다. 이를 해결하기 위해 Application 클래스의 onCreate() 시점에서 비동기 처리를 적용하여 초기화 작업을 백그라운드 스레드로 분리함으로써, 앱 구동 속도에 영향을 주지 않으면서 검색 기능을 최적화 하였다.
- Firestore 쿼리 제약 사항 극복:** 보유 재료 기반 추천 기능 구현중, Firestore 의 IN 쿼리가 최대 10개의 조건만 허용한다는 제약 사항을 확인했다. 이를 해결하기 위해 전체 재료 리스트를 10개 단위로 분할 하고 병렬 쿼리를 수행한뒤 결과를 병합하는 알고리즘 을 구현하여 대량 데이터 조회 문제를 해결 하였다.
- 반복적인 MVP 코드의 구조적 개선:** MVP 패턴 도입 초기, 모든 Presenter에서 View의 연결과 해제를 처리하는 코드가 중복되는 비효율이 발생하였다. 이를 제네릭을 활용한 BaseContract 와 BasePresenter 추상 클래스로 리팩토링 하여 공통 로직을 모듈화 함으로써, 코드 생산성을 높이고 메모리 누수 위험을 원천적으로 차단하였다.

3.2. 성과 및 기대효과

1) 프로젝트 성과

- **상용 수준의 데이터 및 기능 확보:** Python 크롤링을 통해 구축한 10만 건의 대용량 레시피 데이터를 기반으로, 실제 마켓 출시가 가능한 수준의 완성도 높은 애플리케이션을 구현하였다. 기획 단계의 핵심 요구사항이었던 '냉장고 파먹기(보유 재료 기반 검색)' 기능을 100% 구현함으로써 식재료 관리와 레시피 추천의 유기적인 연동을 실현하였다.
- **안드로이드 최신 기술 내재화:** Jetpack Navigation, WorkManager, Material Design 3 등 최신 안드로이드 라이브러리를 적극 도입하고, MVP 아키텍처를 적용하여 유지보수가 용이하고 확장성 있는 소프트웨어 구조를 확립하였다.

2) 독창적 수행 내용

- **하이브리드 검색 엔진 구축:** 단순히 외부 API에 의존하거나 느린 DB 검색을 사용하는 대신, Algolia 검색엔진과 Okt 형태소 분석기를 결합한 독자적인 하이브리드 검색 시스템을 구축하였다.
- **서비스 기반의 실시간 동기화:** 별도의 물리 서버 없이 Firebase Cloud Functions를 활용하여, 사용자 행동(즐겨찾기, 재료 변경)에 따라 Firestore DB와 Algolia 검색 인덱스, 백그라운드 알림 대기열이 실시간으로 동기화 되는 백엔드 로직을 설계 및 구현하였다.

3) 기대효과

- **사용자 측면:** 사용자는 냉장고 속 잊혀진 식재료의 유통기한을 사전에 인지하여 음식물 쓰레기를 줄이는 친환경적인 생활을 실천할 수 있다. 또한 보유 재료에 기반한 맞춤형 레시피 추천을 통해 "오늘 뭐 먹지?"라는 매일의 메뉴 선정 고민을 단축시키고, 요리에 대한 진입 장벽을 낮추어 건강한 식생활을 유도한다.
- **기술적 측면:** 프로젝트를 통해 확보한 검색 엔진 연동 기술과 대용량 데이터 처리 노하우는 향후 콘텐츠 플랫폼 등 다양한 도메인의 모바일 서비스 개발에 적용 가능하다. 또한 MVP 패턴과 모듈화된 코드는 추후 기능 확장이나 유지보수 시 개발 비용을 절감하는데 기여할 것이다.

참고문헌 및 사이트

- [1] https://github.com/ppmm9155/Food_Recipe , “요리GO조리GO” , 깃허브 (Github)
- [2] [피그마 초기 UI/UX 디자인 프로토타입](#) , 피그마 (Figma)
- [3] [피그마 프로젝트 다이어그램 모음](#), 피그마 (Figma)
- [4] [탐색 : App architecture - Android Developers](#) , Android Developers
- [5] [위키백과 모델-뷰-프리젠테](#) , “모델-뷰-프리젠테”, 위키백과
- [6] [Firebase용 Cloud Functions 공식문서](#) , Firebase
- [7] [안드로이드 태스크 예약](#), Android Developers
- [8] <https://www.10000recipe.com/>, “만개의 레시피” (요리정보 참조) ,(주)만개의레시피

부록 : 소스코드 List 및 파일구조

```

Food_Recipe (Root)
├── app (Android Client)
│   ├── src/main/java/com/example/food_recipe
│   │   ├── base          # MVP 패턴의 공통 기능을 정의한 Base 클래스
│   │   └── BaseContract.java
│   │   └── BasePresenter.java
│   │
│   ├── model           # 데이터 모델 (DTO/VO)
│   │   ├── Recipe.java
│   │   ├── PantryItem.java
│   │   ├── Ingredient.java
│   │   └── CookingStep.java
│   │
│   ├── repository      # 데이터 소스 접근 계층
│   │   └── ExpirationRepository.java
│   │
│   ├── worker          # 백그라운드 작업 (WorkManager)
│   │   └── ExpirationCheckWorker.java
│   │
│   ├── utils            # 유ти리티 클래스
│   │   ├── StringUtils.java    # Okt 형태소 분석 래퍼
│   │   ├── AutoLoginManager.java # 자동 로그인 관리
│   │   └── ValidationUtils.java # 입력값 유효성 검사
│   │
│   ├── adapter          # RecyclerView 어댑터 모음
│   │   ├── RecipeAdapter.java
│   │   ├── PantryAdapter.java
│   │   └── ... (기타 어댑터)
│   │
│   /* --- 기능별 MVP 모듈 (Feature Packages) --- */
│   ├── main              # 메인 액티비티 & 공통 ViewModel
│   │   ├── MainActivity.java
│   │   └── AuthViewModel.java # 인증 상태 공유 뷰모델
│   ├── home              # 홈 화면 (추천/인기 레시피)
│   ├── search             # 검색 화면 (Algolia 연동)
│   │   ├── SearchFragment.java
│   │   ├── SearchPresenter.java
│   │   └── SearchViewModel.java # 검색 상태 보존
│   ├── pantry             # 냉장고 관리 (CRUD)
│   │   ├── PantryFragment.java
│   │   └── PantryRepository.java
│   ├── recipedetail       # 레시피 상세 정보
│   ├── favorites          # 즐겨찾기
│   ├── mypage              # 마이페이지 & 프로필 수정
│   ├── login               # 로그인 기능
│   └── join                # 회원가입 기능
│
│   └── src/main/res        # UI 리소스
│       ├── layout          # 화면 레이아웃 XML
│       ├── navigation/navgraph.xml # Jetpack Navigation 그래프
│       └── drawable, values, ...
│
└── functions (Backend Serverless)
    ├── src
    │   └── index.ts          # Firestore-Algolia 동기화 트리거
    ├── package.json
    └── tsconfig.json
    └──
└── data_pipeline (Data Engineering)
    ├── step1_csv_preprocessor.py # 1단계: CSV 전처리 및 골격 생성
    ├── step2_hybrid_crawler.py  # 2단계: 하이브리드 크롤링 (데이터 보완)
    ├── step3_data_refinement.py # 3단계: 데이터 정제 및 최적화
    └── step4_firestore_uploader.py # 4단계: DB 일괄 적재
    └──
└── documents             # 프로젝트 문서 및 산출물

```

실제 소스코드 리스트

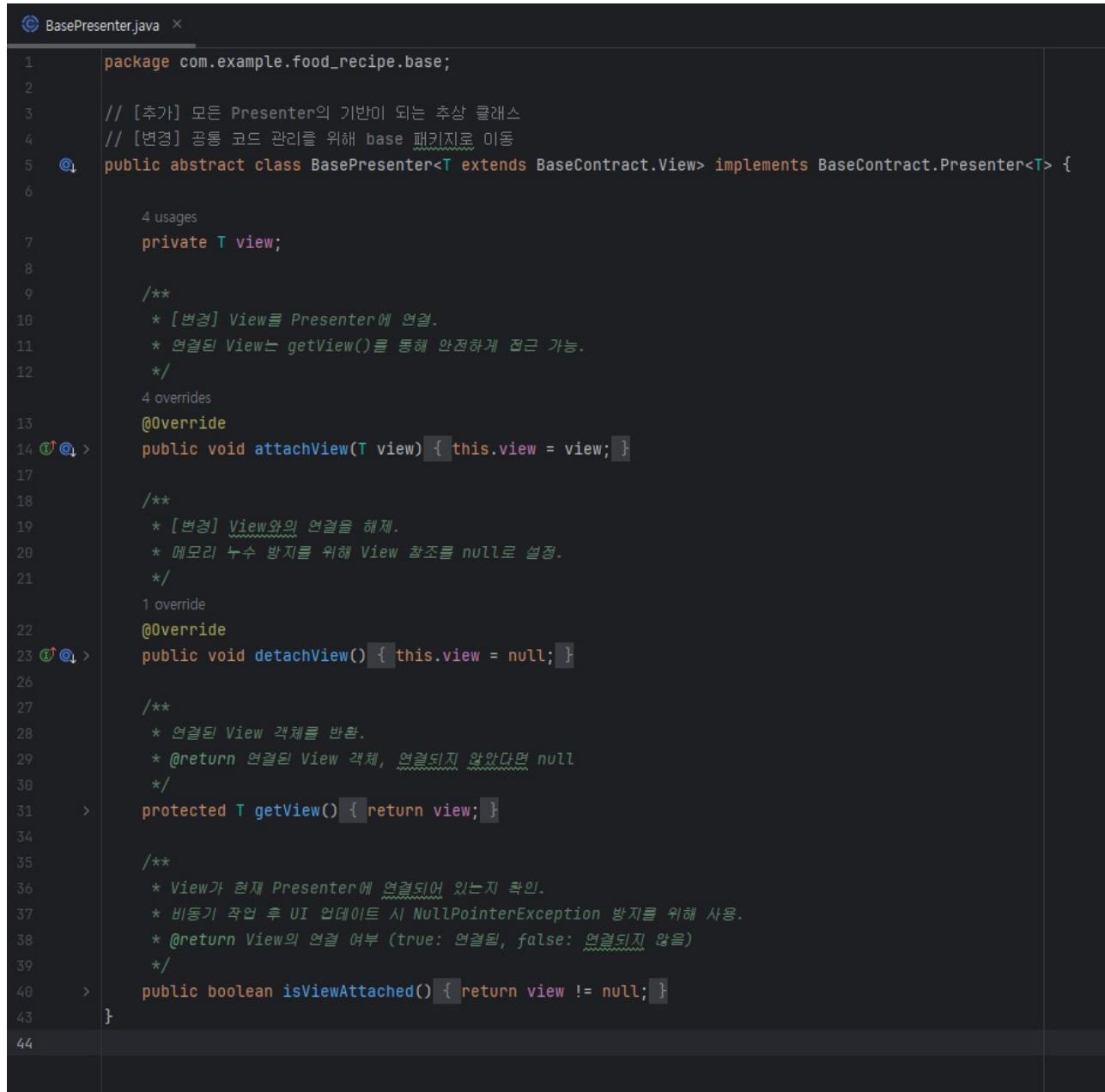


1. front-end 관련 Android MVP & Core Logic

• 아키텍처 기반 클래스 (BasePresenter.java)

설명: MVP 패턴의 View 연결/해제(attach/detach) 로직을 제네릭으로 추상화하여 메모리 누수를 방지하고 코드 재사용성을 높인 클래스.

경로: app/src/main/java/com/example/food_recipe/base/BasePresenter.java



```
1 package com.example.food_recipe.base;
2
3 // [추가] 모든 Presenter의 기반이 되는 추상 클래스
4 // [변경] 공통 코드 관리를 위해 base 패키지로 이동
5 public abstract class BasePresenter<T extends BaseContract.View> implements BaseContract.Presenter<T> {
6
7     private T view;
8
9     /**
10      * [변경] View를 Presenter에 연결.
11      * 연결된 View는 getView()를 통해 안전하게 접근 가능.
12      */
13     @Override
14     public void attachView(T view) { this.view = view; }
15
16     /**
17      * [변경] View와의 연결을 해제.
18      * 메모리 누수 방지를 위해 View 참조를 null로 설정.
19      */
20     @Override
21     public void detachView() { this.view = null; }
22
23     /**
24      * 연결된 View 객체를 반환.
25      * @return 연결된 View 객체, 연결되지 않았다면 null
26      */
27     protected T getView() { return view; }
28
29     /**
30      * View가 현재 Presenter에 연결되어 있는지 확인.
31      * 비동기 작업 후 UI 업데이트 시 NullPointerException 방지를 위해 사용.
32      * @return View의 연결 여부 (true: 연결됨, false: 연결되지 않을)
33      */
34     public boolean isViewAttached() { return view != null; }
35
36 }
```

- 검색 비즈니스 로직 (SearchPresenter.java)

설명: 레시피 재료 및 이름을 받아 Okt 형태소 분석기로 명사를 추출하고, Algolia 검색 모델을 호출하여 결과를 반환하는 검색 기능의 핵심 구현체.

경로: app/src/main/java/com/example/food_recipe/search/SearchPresenter.java

```

1 package com.example.food_recipe.search;
2
3 import android.os.Handler;
4 import android.os.Looper;
5 import android.util.Log;
6 import com.example.food_recipe.base.BasePresenter;
7 import com.example.food_recipe.model.Recipe;
8 import com.example.food_recipe.utils.StringUtils;
9 import java.util.ArrayList;
10 import java.util.List;
11 import java.util.concurrent.CompletableFuture;
12
13 1 usage
14 public class SearchPresenter extends BasePresenter<SearchContract.View> implements SearchContract.Presenter {
15
16     4 usages
17     private final SearchContract.Model model;
18     15 usages
19     private final SearchViewModel viewModel;
20
21     4 usages
22     private final Handler searchHandler = new Handler(Looper.getMainLooper());
23     4 usages
24     private Runnable searchRunnable;
25     1 usage
26     private static final long SEARCH_DELAY_MS = 300;
27
28     1 usage
29     public SearchPresenter(SearchViewModel viewModel) {
30         this.model = new SearchModel();
31         this.viewModel = viewModel;
32     }
33
34     @Override
35     public void start() {
36         if (viewModel.searchResult.getValue() == null || viewModel.searchResult.getValue().isEmpty()) {
37             loadInitialRecipes();
38         }
39     }
40
41     3 usages
42     private void loadInitialRecipes() {
43         if (!isViewAttached()) return;
44         getView().showLoadingIndicator();
45         model.fetchInitialRecipes(new SearchContract.Model.OnRecipesFetchedListener() {
46             @Override
47             public void onSuccess(List<Recipe> recipes) {
48                 if (!isViewAttached()) return;
49                 getView().hideLoadingIndicator();
50                 if (recipes == null || recipes.isEmpty()) {
51                     getView().showInitialView();
52                 } else {
53                     viewModel.setSearchResult(recipes);
54                 }
55             }
56         });
57     }
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
895
896
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
```

```

49
50     @Override
51     public void onError(String message) {
52         if (!isViewAttached()) return;
53         getView().hideLoadingIndicator();
54         getView().showError(message);
55     }
56 }
57
58 1 usage
59 @Override
60 public void search(String query) {
61     if (!isViewAttached()) return;
62
63     if (query == null || query.trim().isEmpty()) {
64         loadInitialRecipes();
65         return;
66     }
67
68     getView().showLoadingIndicator();
69     model.searchRecipes(query, new SearchContract.Model.OnRecipesFetchedListener() {
70     @Override
71     public void onSuccess(List<Recipe> recipes) {
72         if (!isViewAttached()) return;
73         getView().hideLoadingIndicator();
74         if (recipes == null || recipes.isEmpty()) {
75             getView().showEmptyView("검색 결과가 없습니다.");
76         } else {
77             viewModel.setSearchResult(recipes);
78         }
79     }
80     @Override
81     public void onError(String message) {
82         if (!isViewAttached()) return;
83         getView().hideLoadingIndicator();
84         getView().showError(message);
85     }
86 });
87
88 1 usage
89 @Override
90 public void onSearchQuerySubmitted(String query) {
91     if (!isViewAttached()) return;
92     CompletableFuture.supplyAsync(() -> StringUtils.extractNouns(query)) CompletableFuture<String>
93         .thenAccept( String extractedNouns -> {
94             searchHandler.post(() -> {
95                 if (!isViewAttached()) return;
96                 if (extractedNouns != null && !extractedNouns.isEmpty()) {
97                     List<String> currentChips = new ArrayList<>(viewModel.searchChips.getValue());
98                     boolean isChanged = false;
99                     String[] nouns = extractedNouns.split( regex: " " );
100                     for (String noun : nouns) {
101                         if (!currentChips.contains(noun)) {
102                             currentChips.add(noun);
103                         }
104                     }
105                     viewModel.setSearchChips(currentChips);
106                 }
107             });
108         });
109 }

```

```

102             isChanged = true;
103         }
104     }
105     if (isChanged) {
106         viewModel.setSearchChips(currentChips);
107         triggerDebouncedSearch();
108     }
109 }
110 );
111 } CompletableFuture<Void>
112 .exceptionally( Throwable ex -> {
113     searchHandler.post(() -> {
114         if (isViewAttached()) {
115             getView().showError("검색어 분석 중 오류가 발생했습니다.");
116         }
117         Log.e( tag: "SearchPresenter", msg: "Error extracting nouns", ex);
118     });
119     return null;
120 });
121
122     getView().clearSearchViewText();
123 }
124
1 usage
125 @Override
126 public void onPantryImportButtonClicked() {
127     if (!isViewAttached()) return;
128     getView().showLoadingIndicator();
129     model.fetchPantryItems(new SearchContract.Model.OnPantryItemsFetchedListener() {
130         @Override
131         public void onSuccess(List<String> items) {
132             if (!isViewAttached()) return;
133             getView().hideLoadingIndicator();
134
135             if (items == null || items.isEmpty()) {
136                 // [수정] 모호한 문자열 메시지를 보내는 대신, 명확한 메소드를 직접 호출합니다.
137                 getView().showEmptyPantrySnackbar();
138             } else {
139                 getView().showPantryImportBottomSheet(new ArrayList<>(items), new ArrayList<>(viewModel.searchChips.getValue()));
140             }
141         }
142         @Override
143         public void onError(String message) {
144             if (!isViewAttached()) return;
145             getView().hideLoadingIndicator();
146             getView().showError(message);
147         }
148     });
149 }
150
1 usage
151 @Override
152 public void onPantryIngredientsSelected(ArrayList<String> ingredients) {
153     List<String> currentChips = new ArrayList<>(viewModel.searchChips.getValue());

```

```

152 ① @
153     public void onPantryIngredientsSelected(ArrayList<String> ingredients) {
154         List<String> currentChips = new ArrayList<>(viewModel.searchChips.getValue());
155         boolean isChanged = false;
156         for (String ingredient : ingredients) {
157             String normalizedIngredient = StringUtils.normalizeIngredientName(ingredient);
158             if (!currentChips.contains(normalizedIngredient)) {
159                 currentChips.add(normalizedIngredient);
160                 isChanged = true;
161             }
162         }
163         if (isChanged) {
164             viewModel.setSearchChips(currentChips);
165             triggerDebouncedSearch();
166         }
167     }

168     1 usage
169 ① @Override
170     public void onChipClosed(String chipText) {
171         List<String> currentChips = new ArrayList<>(viewModel.searchChips.getValue());
172         if (currentChips.remove(chipText)) {
173             viewModel.setSearchChips(currentChips);
174             triggerDebouncedSearch();
175         }
176     }

177     2 usages
178 ① @Override
179     public void onPantrySelectionCancelled() {
180         if (!isViewAttached()) return;
181         List<Recipe> currentRecipes = viewModel.searchResult.getValue();
182
183         if (currentRecipes == null || currentRecipes.isEmpty()) {
184             loadInitialRecipes();
185         } else {
186             viewModel.setSearchResult(currentRecipes);
187         }
188     }

189     3 usages
190     private void triggerDebouncedSearch() {
191         if (searchRunnable != null) {
192             searchHandler.removeCallbacks(searchRunnable);
193         }
194         searchRunnable = this::performSearch;
195         searchHandler.postDelayed(searchRunnable, SEARCH_DELAY_MS);
196     }

197     1 usage
198     private void performSearch() {
199         String finalQuery = String.join(" ", viewModel.searchChips.getValue());
200         search(finalQuery);
201     }
202 }

```

- 백그라운드 알림 워커 (ExpirationCheckWorker.java)

설명: 앱이 종료된 상태에서도 WorkManager를 통해 주기적으로 유통기한 임박 재료를 감지하고 시스템 푸시 알림을 발송하는 백그라운드 서비스.

경로: app/src/main/java/com/example/food_recipe/worker/ExpirationCheckWorker.java

```

35      2 usages
36  public class ExpirationCheckWorker extends Worker {
37
38      12 usages
39      private static final String TAG = "ExpirationCheckWorker";
40      3 usages
41      private final ExpirationRepository repository;
42      7 usages
43      private final Context context;
44      2 usages
45      private static final String CHANNEL_ID = "EXPIRATION_REMINDER_CHANNEL";
46      1 usage
47      private static final String CHANNEL_NAME = "유통기한 알림";
48      1 usage
49      private final AtomicInteger notificationId = new AtomicInteger( initialValue: 1 );
50
51      no usages
52
53      @NotNull
54      @Override
55      public Result doWork() {
56          Log.d(TAG, msg: "WorkManager 작업 실행: 유통기한 확인 로직을 시작합니다.");
57
58          // [추가] Worker 입력 데이터에서 사용자 UID를 가져옵니다.
59          String uid = getInputData().getString( key: "uid" );
60
61          // [추가] UID가 없으면 작업을 중단합니다.
62          if (uid == null || uid.isEmpty()) {
63              Log.e(TAG, msg: "사용자 UID가 없어 작업을 중단합니다. 로그인 상태를 확인해주세요.");
64              return Result.failure();
65          }
66          Log.d(TAG, msg: "사용자 UID(" + uid + ")에 대한 작업을 시작합니다.");
67
68          createNotificationChannel();
69
70          try {
71              // [수정] Repository를 호출할 때 UID를 전달합니다.
72              QuerySnapshot querySnapshot = Tasks.await(repository.getExpiringIngredientsQuery(uid).get());
73
74              if (querySnapshot.isEmpty()) {
75                  Log.d(TAG, msg: "알림을 보낼 재료가 없습니다.");
76                  return Result.success();
77              }
78
79              SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd", Locale.getDefault() );
80
81              for (DocumentSnapshot document : querySnapshot.getDocuments()) {
82                  String ingredientName = document.getString( field: "ingredientName" );
83                  Timestamp expirationTimestamp = document.getTimestamp( field: "expirationDate" );
84                  String status = document.getString( field: "notificationStatus" );
85

```

```

86     if (ingredientName == null || ingredientName.isEmpty() || expirationTimestamp == null) {
87         Log.w(TAG, msg: "확인 실패: 문서 " + document.getId() + "의 재료 이름 또는 유통기한 정보가 없습니다.");
88         continue;
89     }
90
91     Date expirationDate = expirationTimestamp.toDate();
92     long daysRemaining = getDaysRemaining(expirationDate);
93
94     Log.d(TAG, msg: "확인 중: " + ingredientName + " (유통기한: " + sdf.format(expirationDate) + ", D-" + daysRemaining + ")");
95
96     if ("PENDING".equals(status)) {
97         if (daysRemaining >= 0 && daysRemaining <= 3) {
98             String notificationText;
99             if (daysRemaining == 0) {
100                 notificationText = ingredientName + "의 유통기한이 오늘 만료됩니다!";
101             } else {
102                 notificationText = ingredientName + "의 유통기한이 " + daysRemaining + "일 남았습니다.";
103             }
104             Log.i(TAG, msg: "알림 발송: " + ingredientName + " (" + notificationText + ")");
105             sendNotification(notificationText);
106             Tasks.await(repository.updateNotificationStatus(document.getId()));
107             Log.d(TAG, msg: "상태 변경: " + ingredientName + " -> SENT");
108         } else {
109             Log.d(TAG, msg: "알림 제외: " + ingredientName + " (사유: 유통기한이 " + daysRemaining + "일 남아 알람 대상이 아님)");
110         }
111     } else {
112         Log.d(TAG, msg: "알림 제외: " + ingredientName + " (사유: 이미 알람을 보냄)");
113     }
114 }
115
116     return Result.success();
117
118 } catch (ExecutionException | InterruptedException e) {
119     Log.e(TAG, msg: "WorkManager 작업 실패", e);
120     return Result.retry();
121 }
122 }
123
1 usage
124 private long getDaysRemaining(Date expirationDate) {
125     Calendar today = Calendar.getInstance();
126     today.set(Calendar.HOUR_OF_DAY, 0);
127     today.set(Calendar.MINUTE, 0);
128     today.set(Calendar.SECOND, 0);
129     today.set(Calendar.MILLISECOND, 0);
130
131     Calendar expirationCal = Calendar.getInstance();
132     expirationCal.setTime(expirationDate);
133     expirationCal.set(Calendar.HOUR_OF_DAY, 0);
134     expirationCal.set(Calendar.MINUTE, 0);
135     expirationCal.set(Calendar.SECOND, 0);
136     expirationCal.set(Calendar.MILLISECOND, 0);
137
138     long diff = expirationCal.getTimeInMillis() - today.getTimeInMillis();
139     return TimeUnit.MILLISECONDS.toDays(diff);
140 }
```

```
1 usage
143 private void sendNotification(String notificationText) {
144     Intent intent = new Intent(context, MainActivity.class);
145     intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
146     PendingIntent pendingIntent = PendingIntent.getActivity(context, requestCode, intent, PendingIntent.FLAG_IMMUTABLE);
147
148     NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID)
149         .setSmallIcon(R.drawable.icon)
150         .setContentTitle("【통기한】 일박 알림")
151         .setContentText(notificationText)
152         .setPriority(NotificationCompat.PRIORITY_DEFAULT)
153         .setAutoCancel(true)
154         .setContentIntent(pendingIntent);
155
156     NotificationManagerCompat notificationManager = NotificationManagerCompat.from(context);
157
158     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
159         if (!ContextCompat.checkSelfPermission(context, Manifest.permission.POST_NOTIFICATIONS) != PackageManager.PERMISSION_GRANTED) {
160             Log.w(TAG, msg: "알림 권한이 없어 알림을 보낼 수 없습니다.");
161             return;
162         }
163     }
164     notificationManager.notify(notificationId.getAndIncrement(), builder.build());
165 }
166
1 usage
167 private void createNotificationChannel() {
168     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
169         NotificationChannel channel = new NotificationChannel(CHANNEL_ID, CHANNEL_NAME, NotificationManager.IMPORTANCE_DEFAULT);
170         channel.setDescription("【통기한】 일박한 재료에 대한 알림을 받습니다.");
171         NotificationManager notificationManager = context.getSystemService(NotificationManager.class);
172         notificationManager.createNotificationChannel(channel);
173     }
174 }
175 }
176 }
```

2. back-end 관련 Data Engineering

- 실시간 데이터 동기화 트리거 (functions/src/index.ts)

설명: Firebase Cloud Functions를 활용하여 Firestore 데이터 변경(추가/수정/삭제) 시 Algolia 검색 인덱스를 실시간으로 동기화하는 서비스 함수.

경로: functions/src/index.ts

```
1  /**
2   * Firebase 및 외부 라이브러리에서 필요한 기능들을 가져옵니다.
3   */
4  import {onDocumentUpdated} from "firebase-functions/v2/firestore";
5  import * as logger from "firebase-functions/logger";
6  import algoliasearch from "algoliasearch";
7  // [추가] Firestore 데이터베이스 접근을 위한 Firebase Admin SDK를 가져옵니다.
8  import * as admin from "firebase-admin";
9
10
11 // --- Algolia 클라이언트 초기화 (최신 .env 방식) ---
12 // .env.foodrecipe-dfc0e 파일에 정의된 환경 변수를 직접 사용합니다.
13 const ALGOLIA_APP_ID = process.env.ALGOLIA_APP_ID!;
14 const ALGOLIA_ADMIN_KEY = process.env.ALGOLIA_ADMIN_KEY!;
15
16 // Algolia 클라이언트를 생성합니다.
17 // .env 파일의 키를 바탕으로 클라이언트를 초기화합니다.
18 const algoliaClient = algoliasearch(ALGOLIA_APP_ID, ALGOLIA_ADMIN_KEY);
19
20 // 'recipes' 인덱스를 지정합니다.
21 const recipesIndex = algoliaClient.initIndex("recipes");
22
23 // [추가] Firebase Admin SDK를 초기화합니다. Cloud Functions 환경에서 자동으로 구성됩니다.
24 admin.initializeApp();
25
26
27 // --- Firebase Function 정의 ---
28 /**
29 * 'recipes' 컬렉션의 문서가 '업데이트'될 때마다 자동으로 실행되는 함수입니다.
30 */
31 export const syncRecommendCountToAlgolia = onDocumentUpdated(
32   "recipes/{recipeId}",
33   async (event) => {
34     const {recipeId} = event.params;
35     const beforeData = event.data?.before.data();
36     const afterData = event.data?.after.data();
37
38     // 데이터 유효성 검사
39     if (!beforeData || !afterData || !(`recommend_count` in afterData)) {
40       logger.log(`[${recipeId}] 데이터 부족으로 동기화 건너뜁니다.`);
41       return;
42     }
43
44     const beforeCount = beforeData.recommend_count;
45     const afterCount = afterData.recommend_count;
46
47     // recommend_count 변경 여부 확인
48     if (beforeCount === afterCount) {
49       logger.log(
50         `[$recipeId] recommend_count 변경 없음 (값: ${afterCount}).`;
51     );
52     }
53   }
);
```

```
55 // 변경 로그 기록
56 logger.info(
57   `[${recipeId}] recommend_count 변경: ${beforeCount} -> ${afterCount}.`,
58   "Algolia 동기화 시작.",
59 );
60
61 const objectToUpdate = {
62   objectId: recipeId,
63   recommend_count: afterCount,
64 };
65
66 try {
67   // Algolia에 부분 업데이트 요청
68   const result = await recipesIndex.partialUpdateObject(objectToUpdate, {
69     createIfNotExists: true,
70   });
71   logger.info(
72     `[${recipeId}] Algolia 업데이트 성공. 작업 ID: ${result.taskID}`,
73   );
74 } catch (error) {
75   logger.error(`[${recipeId}] Algolia 업데이트 실패.`, error);
76 }
77 },
78 );
```

```
81
82 /**
83 * [추가] 'users' 컬렉션의 문서가 업데이트될 때마다 자동으로 실행되는 함수입니다.
84 * 'myIngredients' 배열의 변경사항을 감지하여 'expiringIngredients' 컬렉션과 동기화합니다.
85 */
86 export const syncExpiringIngredients = onDocumentUpdated(
87   "users/{uid}",
88   async (event) => {
89     // 1. 컬렉션 및 데이터 추출
90     const {uid} = event.params;
91     const beforeData = event.data?.before.data();
92     const afterData = event.data?.after.data();
93
94     // 2. 변경 전/후의 myIngredients 배열을 안전하게 가져옵니다.
95     const beforeIngredients = (beforeData?.myIngredients && Array.isArray(beforeData.myIngredients)) ? beforeData.myIngredients : [];
96     const afterIngredients = (afterData?.myIngredients && Array.isArray(afterData.myIngredients)) ? afterData.myIngredients : [];
97
98     // 3. 성능 최적화: 변경이 없으면 함수를 일찍 종료합니다.
99     if (JSON.stringify(beforeIngredients) === JSON.stringify(afterIngredients)) {
100       logger.log(`[${uid}] myIngredients 배열에 변경 사항이 없어 동기화를 건너뜁니다.`);
101       return;
102     }
103
104     // 4. 비교를 위해 재료 배열을 Map으로 변환합니다 (Key: id, Value: PantryItem).
105     const beforeMap = new Map(beforeIngredients.map((item: any) => [item.id, item]));
106     const afterMap = new Map(afterIngredients.map((item: any) => [item.id, item]));
107
108     // 5. 모든 비동기 Firestore 작업을 저장할 프로미스 배열을 준비합니다.
109     const promises: Promise<any>[] = [];
110
111     // 6. [삭제된 재료] 처리: beforeMap에만 있는 재료를 찾습니다.
112     for (const [id, item] of beforeMap.entries()) {
113       if (!afterMap.has(id)) {
114         logger.info(`[${uid}] 재료 삭제 감지: ${item.name}. 'expiringIngredients'에서 삭제합니다.`);
115         const deletePromise = admin.firestore().collection("expiringIngredients").doc(id).delete();
116         promises.push(deletePromise);
117       }
118     }
119   }
120 }
```

```

119
120     // 7. [추가/수정된 재료] 처리: afterMap의 모든 재료를 순회합니다.
121     for (const [id, item] of afterMap.entries()) {
122         const beforeItem = beforeMap.get(id);
123
124         // 7-1. 변경 여부 판단
125         const isNew = !beforeItem;
126         let isModified = false;
127         if (beforeItem) {
128             const nameChanged = beforeItem.name !== item.name;
129             const beforeDate = beforeItem.expirationDate; // Timestamp object or null
130             const afterDate = item.expirationDate; // Timestamp object or null
131             let dateChanged = false;
132             if (beforeDate && afterDate) {
133                 // 둘 다 유효한 Timestamp면 isEqual로 비교
134                 dateChanged = !beforeDate.isEqual(afterDate);
135             } else if (beforeDate || afterDate) {
136                 // 한쪽만 있거나 없으면(추가/삭제) 변경된 것으로 간주
137                 dateChanged = true;
138             }
139             isModified = nameChanged || dateChanged;
140         }
141
142         // 7-2. 변경된 경우에만 동기화 작업 수행
143         if (isNew || isModified) {
144             // 7-3. 유통기한이 있는 재료는 'expiringIngredients'에 추가/수정
145             if (item.expirationDate) {
146                 logger.info(`[${uid}] 재료 추가/수정 감지: ${item.name}. 'expiringIngredients'에 반영합니다.`);
147                 const docToSet = {
148                     uid,
149                     ingredientName: item.name,
150                     expirationDate,
151                     notificationStatus: "PENDING", // 항상 'PENDING'으로 초기화
152                 };
153                 const upsertPromise = admin.firestore().collection("expiringIngredients").doc(id).set(docToSet);
154                 promises.push(upsertPromise);
155             } else {
156                 // 7-4. 유통기한이 없는 재료는 'expiringIngredients'에서 삭제
157                 logger.info(`[${uid}] 재료 '${item.name}'에 유통기한이 없어 'expiringIngredients'에서 삭제합니다.`);
158                 const deletePromise = admin.firestore().collection("expiringIngredients").doc(id).delete();
159                 promises.push(deletePromise);
160             }
161         }
162     }
163
164     // 8. 모든 동기화 작업을 한 번에 실행합니다.
165     if (promises.length > 0) {
166         await Promise.all(promises);
167         logger.log(`[${uid}] 'expiringIngredients' 동기화 완료. 처리된 작업 수: ${promises.length}`);
168     } else {
169         logger.log(`[${uid}] 'expiringIngredients' 동기화할 내용이 없습니다.`);
170     }
171 },
172 );
173

```

• 데이터 구축 파이프라인 (Python Scripts)

설명: KADX 무료 레시피 데이터 (만개의 레시피) CSV와 웹 크롤링을 결합하여 10만 건의 고품질 레시피 데이터를 구축하는 4단계 자동화 스크립트.

핵심 파일: data_pipeline/step2_hybrid_crawler.py

```

1 import json
2 import requests
3 from bs4 import BeautifulSoup
4 import time
5 from tqdm import tqdm
6 import re
7 import random
8 import os
9
10 # --- 설정값 ---
11 # (입력 파일) 1단계에서 만든 '쇼핑 리스트' JSON
12 INPUT_JSON_FILE = 'all_recipes_for_firestore.json'
13 # (중간 저장) 1000개마다 중간 결과를 저장합니다.
14 CHECKPOINT_INTERVAL = 1000
15 # (중간 저장 폴더) 중간 결과 파일들이 저장될 폴더 이름
16 CHECKPOINT_DIR = 'crawling_checkpoints'
17 #
18
19 # '진짜 브라우저'로 헤더 정보
20 headers = {
21     'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36',
22     'Referer': 'https://www.google.com/'
23 }
24
25 # 중간 저장 폴더가 없으면 새로 생성합니다.
26 if not os.path.exists(CHECKPOINT_DIR):
27     os.makedirs(CHECKPOINT_DIR)
28     print(f'{CHECKPOINT_DIR} 폴더를 생성했습니다.')
29
30 # --- '이어하기' 기능: 이미 처리된 레시피 목록을 확인합니다. ---
31 processed_ids = set()
32 checkpoint_files = os.listdir(CHECKPOINT_DIR)
33 if checkpoint_files:
34     print("기존에 저장된 체크포인트 파일을 확인합니다...")
35     for file_name in tqdm(checkpoint_files, desc="진행 상황 로딩 중"):
36         file_path = os.path.join(CHECKPOINT_DIR, file_name)
37         with open(file_path, 'r', encoding='utf-8') as f:
38             partial_data = json.load(f)
39             for item in partial_data:
40                 processed_ids.add(item['RCP_SNO'])
41     print(f'총 {len(processed_ids)} 개의 레시피가 이미 처리되었습니다. 이제서 크롤링을 시작합니다.')
42
43 # --- '쇼핑 리스트'를 불러옵니다. ---
44 try:
45     with open(INPUT_JSON_FILE, 'r', encoding='utf-8') as f:
46         all_recipes = json.load(f)
47 except FileNotFoundError:
48     print(f'X 오류: "{INPUT_JSON_FILE}" 을 찾을 수 없습니다.')
49     raise SystemExit("스크립트 종단")
50
51 # 아직 처리되지 않은 레시피 목록만 골라냅니다.
52 recipes_to_crawl = [r for r in all_recipes if r['RCP_SNO'] not in processed_ids]
53
54 if not recipes_to_crawl:
55     print("\n 모든 레시피의 크롤링이 이미 완료되었습니다! 'merge_files.py'를 실행하여 최종 파일을 만드세요.")
56     raise SystemExit("작업 종료")
57

```

```

57
58     print(f"\n총 {len(all_recipes)}개 중, {len(recipes_to_crawl)}개의 레시피에 대한 크롤링을 시작합니다.")
59     print(f"주의: 작업은 언제든지 중단하고 다시 실행할 수 있습니다.")
60
61     # --- 메인 크롤링 루프 ---
62     partial_results = []
63     start_index = len(processed_ids)
64
65     for recipe in tqdm(recipes_to_crawl, desc="레시피 크롤링/복원 중"):
66         rcp_sno = recipe.get('RCP_SNO')
67         if not rcp_sno: continue
68
69         url = f"https://www.10000recipe.com/recipe/{rcp_sno}"
70
71         try:
72             response = requests.get(url, headers=headers)
73             if response.status_code == 200:
74                 soup = BeautifulSoup(response.text, 'html.parser')
75
76                 # =====
77                 # [최종 하이브리드] '보여주기용'과 '검색용'을 각자 최고의 방식으로 처리합니다.
78                 # =====
79                 ingre_div = soup.find('div', class_='ready_inge3')
80                 if ingre_div:
81                     scraped_ingredients_with_amount = [] # '재료 이름 + 양' 목록 (보여주기용)
82                     scraped_ingredient_names = []       # '재료 이름'만 있는 목록 (검색용)
83
84                     for li in ingre_div.find_all('li'):
85                         # 1. <li> 태그 안의 텍스트를 가져와 '구매' 단어를 제거하고 정리합니다.
86                         raw_text = li.get_text()
87                         cleaned_text = ' '.join(raw_text.replace('구매', '').split())
88                         if not cleaned_text: continue
89
90                         # 2. [보여주기용] 정리된 전체 텍스트를 그대로 저장합니다.
91                         scraped_ingredients_with_amount.append(cleaned_text)
92
93                         # 3. [검색용] 전체 텍스트에서 '양/단위' 부분을 제거하여 '이름'만 추출합니다.
94                         amount_tag = li.find('span', class_='inge_unit')
95                         amount = amount_tag.get_text(strip=True) if amount_tag else ""
96                         name = cleaned_text.replace(amount, '').strip()
97
98                         # <a> 태그가 있는 경우, 더 깨끗한 이름을 얻기 위해 한 번 더 정제합니다.
99                         name_link = li.find('a')
100                        if name_link:
101                            name = name_link.get_text(strip=True)
102
103                        if name:
104                            scraped_ingredient_names.append(name)
105
106                        if scraped_ingredients_with_amount:
107                            recipe['ingredients_raw'] = f"[재료] {f' | '.join(scraped_ingredients_with_amount)}"
108                            recipe['ingredients'] = sorted(list(set(scraped_ingredient_names)))
109
110                         # =====

```

```

110
111     # --- 기준 입력: imageUrl 및 cooking_steps 해우기 ---
112     if not recipe.get('imageUrl'):
113         main_img = soup.find('img', id='main_thumbs')
114         if main_img and 'src' in main_img.attrs: recipe['imageUrl'] = main_img['src']
115         else:
116             main_thumb_div = soup.find('div', class_="view_pic")
117             if main_thumb_div and main_thumb_div.find('img'): recipe['imageUrl'] = main_thumb_div.find('img')['src']
118
119     recipe['cooking_steps'] = [] # cooking_steps 초기화
120     step_divs_new = soup.findall('div', class_="view_step_cont")
121     if step_divs_new:
122         for index, step_div in enumerate(step_divs_new):
123             desc = step_div.find('div', class_="media-body")
124             img = step_div.find('img')
125             if desc: recipe['cooking_steps'].append({'step': index + 1, "description": desc.get_text(strip=True), "imageUrl": img['src'] if img and 'src' in img.attrs else ""})
126         else:
127             recipe_step_div_old = soup.find('div', id='recipe_step')
128             if recipe_step_div_old:
129                 elements = recipe_step_div_old.findall(['div', 'p'])
130                 text_buffer = ""
131                 for el in elements:
132                     text = el.get_text(strip=True)
133                     if text:
134                         if re.match(r'^\d+.', text):
135                             if text_buffer: recipe['cooking_steps'].append({'step': len(recipe['cooking_steps']) + 1, "description": text_buffer, "imageUrl": ""})
136                             text_buffer = text
137                         else: text_buffer += " " + text
138                 if text_buffer: recipe['cooking_steps'].append({'step': len(recipe['cooking_steps']) + 1, "description": text_buffer, "imageUrl": ""})
139
140     except Exception as e:
141         pass # 오류가 발생해도 멈추지 않고 다음으로 넘어갑니다.
142
143     partial_results.append(recipe)
144
145     # --- '자주 중간 저장' 가능 ---
146     if len(partial_results) >= CHECKPOINT_INTERVAL:
147         file_num = (start_index // CHECKPOINT_INTERVAL) + 1
148         file_name = f"partial_results_{file_num}.json"
149         with open(os.path.join(CHECKPOINT_DIR, file_name), 'w', encoding='utf-8') as f:
150             json.dump(partial_results, f, ensure_ascii=False, indent=4)
151             print(f"\n✓ {file_name} 저장! {file_name}")
152         partial_results = []
153         start_index += CHECKPOINT_INTERVAL
154
155         time.sleep(random.uniform(0.3, 0.8))
156
157     # --- 마지막에 남은 데이터 저장 ---
158     if partial_results:
159         file_num = (start_index // CHECKPOINT_INTERVAL) + 1
160         file_name = f"partial_results_{file_num}.json"
161         with open(os.path.join(CHECKPOINT_DIR, file_name), 'w', encoding='utf-8') as f:
162             json.dump(partial_results, f, ensure_ascii=False, indent=4)
163             print(f"\n✓ {file_name} 저장! {file_name}")
164
165     print("\n모든 레시피 크롤링/파arse 완료! 'merge_files.py'로 실행하세요.")

```

3. DB 관련 Firestore Data Modeling

- **User 컬렉션 구조 (NoSQL Document)**

설명: 사용자 기본 정보와 냉장고 재료(myIngredients), 즐겨찾기 목록을 단일 문서 내에서 관리하여 읽기 성능을 최적화한 구조.

```

+ 필드 추가
▶ bookmarked_recipes: ["6982014"]
createdAt: 2025년 11월 16일 오후 10시 13분 8초 UTC+9
email: "ppmm9155@naver.com"
emailVerified: true
▼ myIngredients
  ▶ 0 {category: "육류 🍖", expira...}
  ▶ 1 {category: "채소 🥑", expira...}
  ▶ 2 {category: "채소 🥑", expira...}
provider: "password"
uid: "LtdABbM7kWNn7AehqSywhffe98U2"
username: "기준"
usernameLower: "기준"

```

- **Recipes 컬렉션 구조**

설명: 검색 및 상세 정보 표시에 최적화된 레시피 데이터 모델. 검색용 키워드(ingredients)와 표시용 텍스트(ingredients_raw)를 분리하여 저장함.

```
+ 필드 추가
RCP_SNO: "1000240"
category_kind: "양식"
▶ cooking_steps: [{imageUrl: "https://recip..."]
cooking_time: "60분이내"
difficulty: "초급"
imageUrl: "https://recipe1.ezmember.co.kr/cache/recipe/2015/09/27/f41058cea2daaca36cfee2f97af55db.jpg"
▶ ingredients: ["기꼬망마리네이트소스", "노른자", "닛신 ..."]
ingredients_raw: "[재료] 소고기 다짐육 300g | 빵가루 50g | 노른자 1개 | 양파 작은것 1/2개 | 기꼬망마리네이트소스 | 샴로소 포도씨유 1큰술 | 모짜렐라치즈 150g | 토스터드세사미 마리네이트소스 1.5큰술 | 후추가루 1작은술 | 파슬리가루 1작은술 | 낫신 니스 토마토소스 1팩"
recommend_count: 14
scrap_count: 56
servings: "2인분"
title: "휴일 점심메뉴로 제격 토마토소스 미트볼 그라탕"
view_count: 7605
```