[서식1]캡스톤디자인 결과보고서(졸업논문)

AI 기반 실시간 좌석 시각화 안드로이드 애플리케이션 구현

1)

학번 202110387 / 이름 이아림

Implementation of an Android Application for AI-Based Real-Time Seat Visualization

Arim Lee

요 약

본 보고서는 CCTV 기반 이미지 분석과 간이 디지털 도면(Floor Plan)을 결합하여 카페 내 좌석 점유 상태를 실시간으로 시각화하는 안드로이드 애플리케이션 개발 결과를 정리한 개인 결과보고서이다. 시스템은 사용자의 좌석 조회 요청을 기점으로 CCTV 이미지를 수집하고, AWS Rekognition을 통해 사람 객체의 위치 정보를 추출한 뒤, 멀티모달 모델(Gemini Flash 2.0)을 이용해 영상 좌표계와 도면 좌표계 간 매핑을 수행함으로써 점유 좌석을 판별한다. 판별 결과는 모바일 앱의 도면 화면에 반영되어 사용자가 매장 방문 전에 좌석 혼잡도를 직관적으로 확인할 수 있도록 한다.
본인은 클라이언트 핵심 모듈을 중심으로 구현에 참여하였다. 구체적으로 (1) Fused Location 및 Kakao Vector Map SDK를 활용한 위치 기반 카페 탐색 기능, (2) BottomSheet·ViewPager 기반의 카페 상세정보/리뷰 UI 및 좌석 확인 화면으로의 내비게이션, (3) 관리자용 도면 편집 기능(Drag & Drop 객체 배치, 다층 도면 관리, 좌표 정규화 및 DTO 추출)을 설계·구현하여 서비스 운영에 필요한 도면 데이터를 생성하고 등록 API 흐름과 연동하였다. 결과적으로 본 프로젝트는 센서 설치가 필요한 기존 좌석 인식 방식 대비 도입 비용을 낮추면서도, 실시간 좌석 정보를 사용자에게 제공할 수 있는 실용적 스마트 공간 관리 애플리케이션의 구현 가능성을 확인하였다.

<키워드: 안드로이드, 위치기반서비스, MVVM(Model-View-ViewModel)>

I. 서 론

카페 이용자는 방문 전 혼잡도 및 빈 좌석 여부를 알기 어렵고, 운영자는 좌석 점유 데이터를 체계적으로 확보하기 어렵다. 기존 좌석 인식은 압력/열/초음파 등 IoT 센서 기반이 많아 설치 및 유지 비용이 높고 소규모 매장에 확산이 제한적이라는 문제가 있다. 본 캡스톤 프로젝트는 이미 설치된 CCTV를 활용해 추가 하드웨어 없이 좌석 점유를 추정하고, 이를 사용자에게 도면 기반으로 직관적으로 제공하는 것을 목표로 한다.

개인 구현 범위는 사용자가 서비스를 접하는 핵심 접점인 사용자 위치 기반 지도 탐색, 카페 상세 확인, 실시간 카페 좌석 확인의 프론트엔드 흐름과, 운영을 가능하게 하는 관리자 도면 편집이다. 즉 좌석 인식 AI 결과를 보여주기 위한 UX·데이터 구조의 기반을 안드로이드 단에서 완성하는 역할을 담당하였다.

II. 관련 연구

2.1 안드로이드 아키텍처(MVVM) 및 UI 컴포넌트

---

1) 이 보고서는 IT대학 컴퓨터공학과 2025년 2학기 졸업보고서로 심사 통과되었음[지도교수: kyeongpil (인)]

2.1.1 MVVM을 채택하는 이유와 기본 원리

안드로이드에서 아키텍처는 단순 코드 분리가 아니라, 화면 상태(UI state)의 생명주기 관리, 데이터 변경에 대한 일관된 반영을 위한 기본 토대다. Android Developers는 고품질 앱을 위해 UI, 도메인, 데이터 계층을 분리하고, 각 계층의 책임을 명확히 하는 아키텍처 구성을 권장한다.[2)]

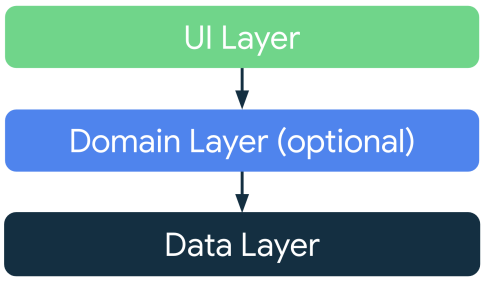

그림 2. 앱 아키텍처 다이어그램

MVVM(Model-View-ViewModel)은 이 권장 아키텍처를 UI 계층에서 실현하기 위한 대표적인 패턴으로, 주요구성은 다음과 같다.

- View: 화면 렌더링 및 사용자 이벤트 전달
- ViewModel: 화면 단위 상태 홀더로서 UI에 노출할 상태를 제공하고, UI 관련 로직을 캡슐화
- Model: Repository, DataSource, UseCase 등으로 구성되며, 네트워크, DB, 캐시 등을 포함한 데이터 처리 담당

특히 ViewModel은 구성 변경(configuration change: 회전 등) 상황에서도 상태를 유지하여, UI가 불필요하게 데이터를 다시 가져오거나 화면이 흔들리는 문제를 완화하는 데 목적이 있다.[3)]

2.1.2 ViewModel과 LiveData의 역할 분담: 상태 홀딩과 생명주기 안전성

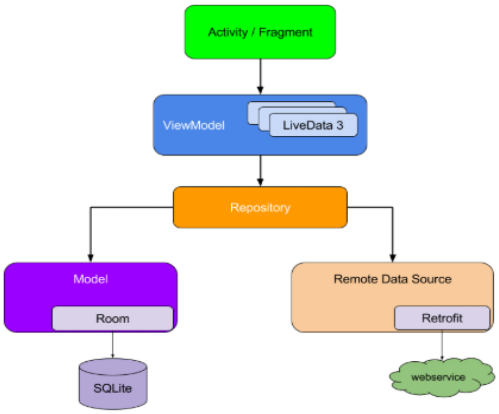

그림 3. MVVM 패턴의 계층 구조와 데이터 흐름

ViewModel은 UI에 필요한 상태를 제공하고 관련 로직을 포함하는 컴포넌트이며, 주요 장점은 구성 변경에도 상태가 유지된다는 점이다. 따라서 “지도 화면의 현재 선택 카페”, “카페 상세 화면 탭 상태”, “카페 도면 화면에서 로딩/성공/실패 상태”처럼 화면 단위로 유지되어야 하는 상태를 ViewModel이 보관하는 것이 합리적이다.

LiveData는 Activity/Fragment 같은 생명주기 소유자와 함께 동작하며, 잘못된 시점에 UI 업데이트가 발생하는 문제를 줄이는 데 유용하다. Android Developers는 LiveData를 ViewModel에서 생

2) Android Developers, 「앱 아키텍처 가이드(Guide to app architecture)」

성하여 UI 계층에 상태를 노출하는 형태를 설명한다.

정리하면, 본 프로젝트 맥락에서 “카페 목록”, “카페 마커 선택 결과”, “카페 도면 데이터 로드 결과” 같은 비동기 결과는 ViewModel의 상태(LiveData 등)로 표현하고, UI는 이를 관찰(observe)하여 렌더링만 담당하게 구성하는 것이 MVVM의 대표적 적용 방식이다.

2.1.3 Repository 기반 데이터 계층

Android Developers는 데이터 계층을 Repository 중심으로 구성하고, Repository가 하나 이상의 데이터 소스(네트워크/로컬 DB/캐시 등)를 조합해 UI에 데이터를 제공하는 구조를 제시한다. 즉, UI(ViewModel)는 어디서 데이터를 가져오는지를 직접 알 필요가 없고, Repository 인터페이스를 통해 결과만 소비하도록 설계한다.

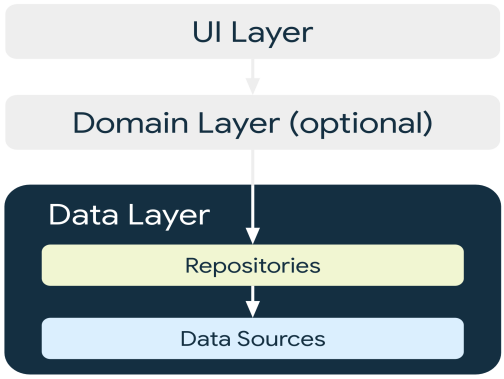

그림 4. 앱 아키텍처에서 데이터 레이어의 역할

본 프로젝트에서는 “카페 목록 조회”, “선택 카페 상세 조회”, “도면 데이터 업로드/다운로드” 등을 Repository 단위로 묶어두면, 화면 확장 시에도 네트워크/캐시 전략 변경이 UI 코드에 전파되는 것을 줄일 수 있다.

2.1.4 Hilt 기반 의존성 주입(DI)

의존성 주입(Dependency Injection; DI)은 객체가 필요한 의존성을 직접 생성하지 않고 외부에서 주입받도록 함으로써, 객체 간 결합도를 낮추고 구조적 유연성을 확보하는 설계 기법이다. Android Developers는 Hilt를 Jetpack에서 권장하는 DI 라이브러리로 설명하며, Android 클래스별 컨테이너와 생명주기 관리를 표준화를 통해 반복적인 보일러 플레이트[4]를 줄인다고 명시한다.[5]

Hilt는 Dagger 기반이며 컴파일 타임 검증을 제공해 런타임 오류 가능성을 낮추는 방향으로 설계되어 있다. 또한 ViewModel에 의존성을 주입하려면 프레임워크가 ViewModel 인스턴스를 생성한다는 특성상 Factory 메커니즘이 필요하며, Android Developers는 ViewModel이 생성자 인자로 의존성을 받을 수 있고 이를 위해 ViewModelProvider.Factory가 사용된다고 안내한다.[6]

Hilt는 이러한 의존성 주입 과정을 Android 컴포넌트의 생명주기와 결합된 컴포넌트 계층 구조로 관리한다. 그림 4는 Hilt가 생성하는 주요 컴포넌트와 Scope 간의 관계를 나타낸 것으로, @SingletonComponent를 최상위로 하여 Activity, Fragment, ViewModel, Service, View 등 Android 구성 요소의 생명주기에 대응하는 하위 컴포넌트들이 계층적으로 연결되어 있다. 이 구조를 통해 애플리케이션 전역에서 공유되어야 하는 객체는 @Singleton으로 관리되고, 화면 단위로 유지되어야 하는 객체는 @ActivityScoped, @FragmentScoped, @ViewModelScoped 등으로 명확히 분리된다.

---

4) Boilerplate: 프로그래밍에서 반복적으로 사용되며, 최소한의 수정만으로 재사용 가능한 표준 코드나 템플릿을 의미함.
5) Android Developers, 「Android의 종속 항목 삽입(Dependency injection in Android)」
6) Android Developers, 「ViewModel 팩토리(ViewModel factories)」

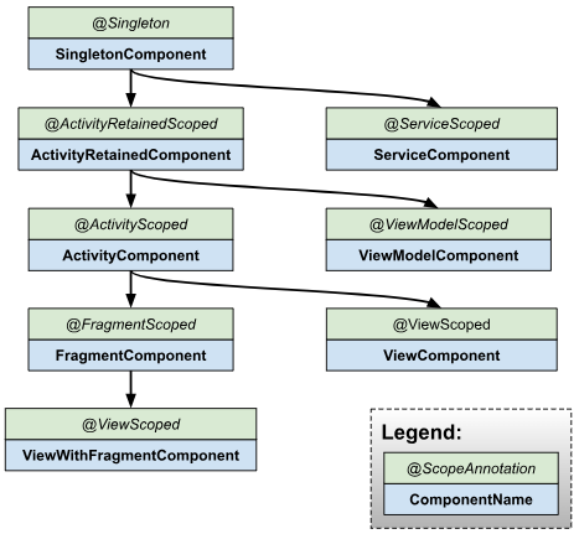

그림 5. Hilt가 생성하는 구성요소의 계층 구조

본 프로젝트와 같이 Network Layer(API Client), Repository, ViewModel이 여러 화면에서 재사용되는 구조에서는 Hilt를 활용한 명확한 Scope 분리와 생명주기 기반 컴포넌트 계층이 의존성 관리의 일관성을 보장한다. 그 결과 기능 추가나 리팩토링 시 사이드 이펙트를 최소화할 수 있으며, 각 계층을 독립적으로 Mock 객체로 대체할 수 있어 단위 테스트 및 통합 테스트 구성이 용이해진다. 이는 전체 시스템의 테스트 가능성과 유지보수성을 향상시키는 핵심 요소로 작용한다.

2.1.5 복수 콘텐츠 UI(ViewPager2/TabLayout)

모바일 UI에서 동등한 위계의 콘텐츠를 한 화면 맥락에서 제공해야 하는 경우, 사용자가 상하 이동만으로는 구조를 파악하기 어렵고, 화면 전환이 잦아질수록 인지 부하가 증가한다. 이에 따라 Android UI 가이드는 동일 레벨의 화면을 가로 스와이프로 전환하는 Swipe views 패턴을 대표적인 네비게이션 방식으로 제시하며, 탭(Tab)을 결합해 현재 위치와 이동 가능 범위를 명확히 드러내는 구성을 설명한다.[7]

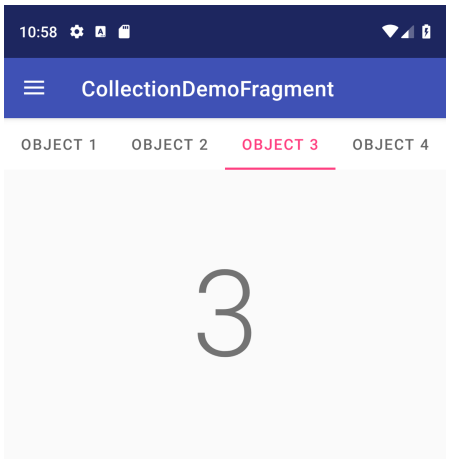

그림 6. 네 개의 탭이 있는 TabLayout

[8]Android Developers는 ViewPager2 기반의 스와이프 뷰에서 TabLayout을 함께 사용해 탭 기반 탐색을 구성할 수 있음을 안내하고, TabLayout과 ViewPager2의 연결을 위해 TabLayoutMediator를 사용하도록 제시한다. 이 Mediator는 탭 선택 시 ViewPager2의 페이지 위치를 동기화하고, 사

7) Android Developers, 「스와이프 뷰 내비게이션 가이드(Navigation with swipe views)」
8) Android Developers, 「TabLayoutMediator 클래스 참조(TabLayoutMediator)」

용자가 ViewPager2를 드래그할 때 TabLayout의 스크롤/선택 상태를 함께 갱신하도록 설계된 구성요소로 정의된다.

2.2 위치 기반 카페 탐색 및 지도 상호작용 설계

위치 기반 서비스(Location Based Service; LBS)는 모바일 환경에서 사용자 주변 맥락을 반영한 탐색 경험을 제공하는 기술로, 정확도, 전력 소모, 응답 지연, 프라이버시 요구사항을 동시에 고려해야 한다. 따라서 플랫폼은 다양한 위치 소스(GPS, Wi-Fi, 셀룰러 등)를 종합해 목적에 맞는 위치 결과를 얻도록 지원하며, 지도 UI에서는 사용자의 카메라 조작(이동/확대/축소)과 데이터 재조회(POI 갱신)를 안정적으로 연결하는 이벤트 설계가 중요하다.

2.2.1 Fused Location Provider 기반 단발성 조회

안드로이드 생태계에서 위치 획득은 일반적으로 Google Play services의 Fused Location Provider를 중심으로 구현되며, 이는 앱이 필요한 수준의 위치를 제공하기 위한 주요 진입점으로 FusedLocationProviderClient를 제공한다.[9)]

FLP 기반 위치 획득은 목적에 따라 크게 (1) 최근 위치기반의 빠른 초기화, (2) 현재 위치 기반의 단발성 조회, (3) 주기적 업데이트로 구분되어 논의된다. Android 공식 가이드는 단일 요청 형태로 최근 위치를 활용하는 접근을 별도로 설명하며, 사용 시점/정확도 요구에 따라 적합한 방식을 선택하도록 안내한다.

본 프로젝트는 Google Play 서비스의 Location API 중 getCurrentLocation[10)] 기반의 단발성 현재 위치 조회를 사용하여 초기 지도 중심을 설정한다.

2.2.2 위치 권한 모델과 런타임 권한 요청

위치 데이터는 개인정보 성격을 가지므로, Android는 앱의 위치 요구 수준에 따라 적절한 권한을 요청하도록 강제한다. Android 공식 문서는 위치 요구사항을 분류하고, 그에 따라 COARSE(대략 위치) 또는 FINE(정밀 위치) 권한을 런타임에 요청하는 절차를 제시한다.[11)]

Target platform version	Coarse or fine permission granted?	Background permission defined in manifest?	Updated default permission state
Android 10	Yes	Yes	Foreground and background access
Android 10	Yes	No	Foreground access only
Android 10	No	(Ignored by system)	No access
Android 9 or lower	Yes	Automatically added by the system at device upgrade time	Foreground and background access
Android 9 or lower	No	(Ignored by system)	No access

그림 7. Android Platform Version에 따른 위치 정보 접근 권한 정책

또한 백그라운드 위치는 플랫폼 및 배포 정책 측면에서 요구사항이 강화되어, 핵심 기능상 반드시 필요한지를 엄격히 검토하고 체크리스트에 따라 설계하도록 안내된다. 즉, 일반적인 주변 탐색 앱에서 무분별하게 채택하기 어렵다. 따라서 본 프로젝트에서는 지도 화면에서의 주변 탐색을 전제로, Foreground 위치 권한만 요청한다.

2.2.3 지도 상호작용 이벤트 설계: 카메라 이동 종료 기반 데이터 재조회

9) Android Developers, 「FusedLocationProviderClient (위치 서비스 클라이언트)」
10) Android Developers, 「현재 위치 정보 가져오기 가이드(Retrieve current location)」
11) Android Developers, 「위치 권한 설명(Location permissions)」

지도 기반 탐색 UI에서 사용자는 드래그/줌을 반복하므로, 카메라가 움직이는 동안 매번 네트워크 요청을 수행하면 트래픽·지연·전력 측면의 부담이 커진다. 이에 따라 실무 및 SDK 가이드에서는 카메라 이동이 끝난 시점을 기준으로 주변 POI/매장 데이터를 재조회하는 패턴이 널리 사용된다.

12)KakaoMaps SDK v2는 카메라 움직임 이벤트를 제공하며, 특히 카메라 이동 종료 시점에 대한 이벤트 수신이 가능하고, 이때 전달되는 파라미터를 통해 이동이 사용자 제스처에 의한 것인지 여부도 확인할 수 있다고 안내한다. 또한 카카오 문서는 카메라 값이 변경 직후 즉시 반영되는 것이 보장되지 않을 수 있으므로, 정확한 카메라 변경 값을 얻기 위해서는 리스너/콜백에서 값을 취득하는 방식을 권장한다.13)

```
kakaoMap.setOnCameraMoveStartListener(new KakaoMap.OnCameraMoveStartListener() {
    @Override
    public void onCameraMoveStart(KakaoMap kakaoMap, GestureType gestureType) {
        // 카메라 움직임 시작 시 이벤트 호출
        // 사용자 제스처가 아닌 코드에 의해 카메라가 움직이면 GestureType 은 Unknown
    }
});

kakaoMap.setOnCameraMoveEndListener(new KakaoMap.OnCameraMoveEndListener() {
    @Override
    public void onCameraMoveEnd(KakaoMap kakaoMap, CameraPosition cameraPosition, GestureType gestureType) {
        // 카메라 움직임 종료 시 이벤트 호출
        // 사용자 제스처가 아닌 코드에 의해 카메라가 움직이면 GestureType 은 Unknown
    }
});
```

그림 8. Kakao Map API를 활용한 지도 카메라 이동 이벤트 처리 로직

2.2.4 마커(라벨) 클릭 이벤트와 선택 상태 유지

지도에서 POI/매장을 선택할 때, 사용자는 무엇이 선택되었는지를 지속적으로 인지할 수 있어야 하므로 선택 상태를 시각적으로 유지하는 UI 설계가 중요하다. KakaoMaps SDK v2는 라벨 클릭 이벤트 리스너를 제공하며, 최근 버전에서는 라벨 클릭 리스너의 반환값을 통해 이벤트 전파를 제어할 수 있도록 인터페이스가 변경되었다.14) 릴리즈 노트에 따르면 반환값이 true인 경우 라벨 클릭 이벤트가 해당 리스너에서 종료되고, false인 경우 다른 클릭 리스너로 전달될 수 있다.

이러한 이벤트 소비 모델은 지도 위에서 라벨/POI/지도 배경 클릭 이벤트가 중첩될 수 있는 상황에서, 의도한 상호작용만 확정적으로 처리하도록 돕는다.

2.3 카페 도면(Floor Plan) 모델링 및 좌표 정규화의 필요성

2.3.1 좌표계 문제: 터치 좌표와 저장 좌표는 동일하지 않다

도면 편집은 터치/드래그 입력을 기반으로 하며, 터치 이벤트는 MotionEvent로 전달된다. Android Developers는 터치 이동을 추적하기 위해 onTouchEvent(MotionEvent)에서 ACTION_MOVE 등을 처리하는 방식을 안내한다.15)

또한 디바이스 환경에 따라 좌표 기준이 달라질 수 있어, 좌표 혼선을 피하려면 MotionEvent.getX()/getY() 같은 View 기준 좌표 사용을 권장하고 getRawX()/getRawY() 사용을 지양하라고 안내한다.16)

즉, 사용자가 도면 위에 객체를 놓을 때 얻는 값은 화면/뷰 좌표(대개 픽셀 단위)이며, 이를 그대로

12) Kakao Developers, 「Android 지도 제스처 가이드(Gesture guide for Android map)」
13) Kakao Developers, 「Android 지도 API 이용 시 주의사항(Precautions for Android map API)」
14) Kakao Developers, 「Android 지도 API 릴리즈 노트(Release notes)」
15) Android Developers, 「제스처 기반 이동(Movement gestures)」
16) Android Developers, 「디스플레이 컷아웃 가이드(Display cutout)」

저장하면 다음 문제가 발생한다.

- 화면 크기/해상도/밀도(dpi)가 다른 기기에서 동일 도면을 로드할 때 위치가 달라짐
- 도면 편집 UI의 패딩/오프셋/스케일 변경 시 기존 데이터와 불일치
- 서버가 필요로 하는 좌표 체계와 직접 호환되지 않음

따라서 입력 좌표(픽셀)를 도면 기준 좌표로 변환하여 저장하는 과정이 필수적이다. 본 프로젝트에서는 이러한 좌표계 불일치를 해결하기 위해 도면 기준 좌표로의 정규화 방식을 적용하였으며, 구체적인 좌표 변환 및 정규화 방법은 이후 3.4.2절에서 상세히 설명한다.

2.3.2 다층(Floor) 도면 모델링: 층 단위 분리와 편집 상태 유지에 관한 선행 접근

다층(2층 이상) 실내 공간은 층마다 배치 요소(좌석/문/창/카운터 등)의 구성이 달라, 단일 평면에 모든 객체를 합쳐 관리할 경우 표현의 복잡도와 편집 오류 가능성이 증가한다. 이에 따라 선행 표준 논의에서는 층/레벨을 독립적인 구획 단위로 두고, 층별로 객체 집합을 분리하는 접근이 반복적으로 등장한다.

(1) 실내 공간 데이터 모델에서의 층(레벨) 단위의 보편성

건물 공간의 구조를 기술하는 대표 표준 중 IFC에는 IfcBuildingStorey를 건물 공간 구조를 구성하는 주요 단위로 설명하며, 상위 구조 안에서 요소들을 담는 컨테이너 역할 수행을 전제로 한다.[17)]

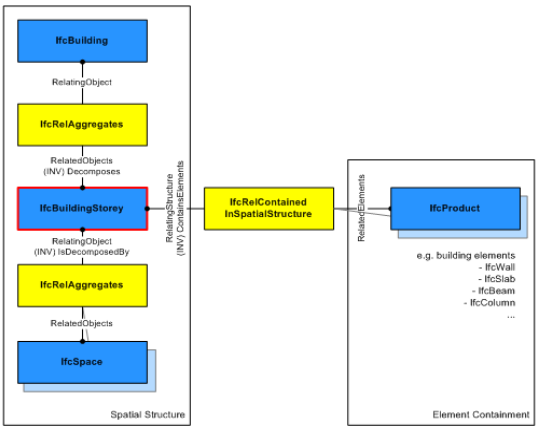

그림 9. IFC 공간 구조 계층

또한 OpenStreetMap의 Indoor Mapping 논의에서도 실내 지도/내비게이션/시각화 등 활용 맥락을 전제로 레벨 기반 실내 표현을 정리하고 있으며, 실내 공간을 층 단위로 다루는 관행이 널리 확산되어 있음을 보여준다.

위와 같은 흐름은 층별로 객체(좌석/문/창 등)를 묶어 관리한다는 구조가 특정 구현의 편의가 아니라, 다층 공간을 일관되게 표현·편집·교환하기 위한 일반적인 분해 방식임을 시사한다.

(2) 다층 편집 UI에서의 상태 유지: 층별 편집 단위의 필요

다층 도면 편집은 사용자가 층을 전환하며 작업하는 반복적 컨텍스트 전환을 포함하므로, 층 전환 시 편집 상태가 즉시 복원되지 않으면 작업 흐름이 끊기거나 객체 누락/중복 같은 편집 오류가 증가할 수 있다. 이러한 문제를 줄이기 위한 일반적인 접근은 편집 상태를 컨텍스트 단위로 분리하여 독립적으로 유지하는 것이다.

안드로이드 앱 아키텍처 가이드는 UI 상태를 state holder가 보유하도록 하여 구성 변경이나 화면 전환에도 상태가 유지되게 하는 구조를 설명하며, 특히 ViewModel은 구성 변경에서도 작업과 상태

17) buildingSMART International, 「IfcBuildingStorey 엔티티 정의(IfcBuildingStorey entity definition)」

그림 10. OpenStreetMap Indoor Mapping 논의에서 제시되는 층(level) 기반 실내 지도 시각화 사례

가 유지되고, Navigation과 결합될 경우 백스택에 있는 화면의 ViewModel을 캐시하여 이전 화면으로 돌아왔을 때 즉시 이전 상태를 활용할 수 있다고 명시한다.[18] 비록 층 전환이 내비게이션 목적지 전환과 동일한 개념은 아니지만, 편집 관점에서는 각 층이 독립된 편집 컨텍스트로 기능하므로, 층별 상태(객체 집합)를 분리해 관리하는 방식은 “컨텍스트 단위 상태 보존”이라는 플랫폼 권장 원칙과 일관된 설계 근거를 갖는다.

(3) 팔레트 기반 객체 배치: Drag & Drop 상호작용의 표준화

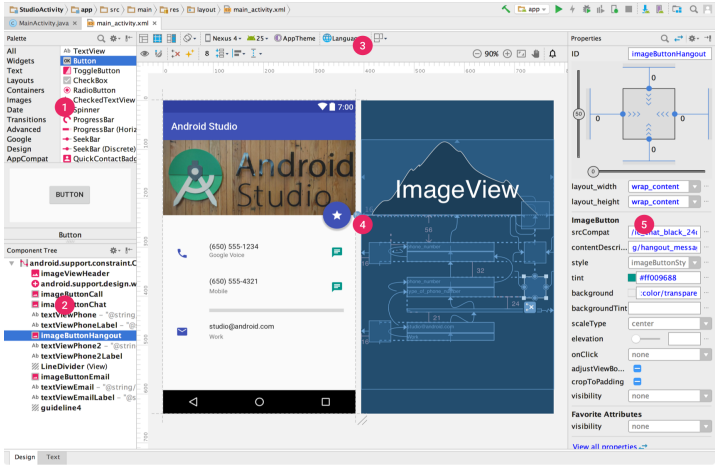

그림 11. Palette에서 UI 요소를 드래그하여 Design editor(작업 영역)에 배치하는 예시

레이아웃 편집기, 실내 도면 편집기 등에서 팔레트(palette)에서 객체를 선택해 작업 영역(canvas)에 배치하는 상호작용은 대표적인 직접 조작(Direct Manipulation) 방식으로 분류된다. 직접 조작 인터페이스는 사용자가 화면상의 객체를 직접 이동·배치하면서 즉각적 피드백을 받는 상호작용을 강조하며, 드래그 앤 드롭은 이를 구현하는 전형적인 기법 중 하나로 널리 논의된다.[19]

안드로이드 플랫폼에서도 이러한 상호작용을 지원하기 위해 View 기반 Drag & Drop 프레임워크를 제공한다.[20] Android Developers 문서는 드래그 시작 트리거에 반응하고, 드롭 대상(View)의 리스너가 DragEvent를 수신하여 드롭을 처리하는 흐름을 제시한다. 이 과정에서 데이터 전달은 ClipData를 통해 수행되며, 드롭 대상은 이벤트의 좌표와 함께 전달된 데이터(예: 객체 유형 식별자)를 기반으로 배치 로직을 수행하는 구조가 일반적이다.

[21]드래그 앤 드롭은 조작이 직관적이라는 장점이 있지만, 사용자에게 동작 가능성을 명확히 보여주

18) Android Developers, 「상태 소유자(State holders)」
19) Nielsen Norman Group, 「Direct Manipulation」
20) Android Developers, 「드래그 앤 드롭 UI 요소 가이드(Drag and drop view)」

는 signifier(드래그 가능 표시)와 단계별 피드백(드래그 중/드롭 가능/드롭 완료)이 없으면 발견 가능성이 떨어질 수 있다는 UX 관점의 가이드가 존재한다. 따라서 팔레트 기반 배치 UI는 기술 구현뿐 아니라, 제스처 기반 상호작용(Drag)의 의미가 사용자에게 전달되도록 시각적 단서와 피드백을 설계하는 것이 중요하다는 점이 함께 언급된다.

그림 12. Drag & Drop 상호작용 과정에서 드롭 가능 영역을 단계적으로 강조하는 시각적 피드백 예시

III. 주요 내용

3.1 전체 시스템에서 개인 구현 모듈

본 프로젝트는 (i) 카페/좌석 관련 데이터를 제공하는 서버 측 기능과 (ii) 사용자가 이를 탐색·확인하고, 관리자가 도면을 구성·등록할 수 있도록 하는 안드로이드 클라이언트로 구성된다. 개인 구현 범위는 클라이언트 영역 중에서도 다음 사용자 접점 흐름과 운영 입력 기능에 집중되어 있다.

1) 사용자 기능

- 현재 위치 기반 주변 카페 지도 표시
- 카페 마커 선택 시, 상세 BottomSheet 제공
- 상세 화면에서 실시간 좌석 확인 진입
- 사용자 리뷰 탭 구조 및 정보 노출 레이아웃 구성

2) 관리자 기능

- 도면 편집 화면에서 객체 Drag & Drop 배치
- 도면 층 전환/층 추가(다층 관리)
- 도면 객체를 저장 DTO로 추출하여 카페 등록 API 흐름에 연결

3.2 위치 기반 카페 지도 표시 기능 구현

3.2.0 기능 개요

본 지도화면(그림 12)은 사용자가 좌석 현황을 확인할 카페를 탐색·선택하는 서비스의 진입화면이다. 지도 렌더링 및 상호작용(카메라 이동, 라벨/마커 표시, 클릭 이벤트 처리)은 Kakao Vector Map SDK를 사용하여 구현하였다. 사용자는 지도에서 주변 카페를 마커로 확인하고, 원하는 카페 마커를 선택하여 카페 상세 정보(BottomSheet)로 진입한 뒤 좌석 확인 화면으로 이동한다(그림 13). 또한 상단 검색 입력 영역을 통해 카페 검색 화면으로 전환하여, 지도 상에서 직접 탐색이 어려운 경우에도 빠르게 카페를 찾을 수 있도록 구성하였다(그림 14).

특히 지도 화면은 현재 위치(녹색 마커)를 기준으로 초기 카메라를 배치하고, 이후 사용자의 지도

21) Nielsen Norman Group,「드래그 앤 드롭(Drag & Drop) 사용성 연구」

그림 13. 위치 기반 카페 지도 표시

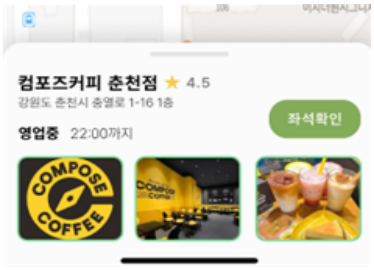

그림 14. 카페 상세화면

그림 15. 카페 검색화면

이동(드래그/줌) 종료 시점에 중심 좌표를 기준으로 서버 카페 목록을 재조회하여 화면에 반영한다.

3.2.1 위치 권한 처리 및 지도 초기화

```
46        /**
47         * 위치 권한 요청
48         */
49        private val locationPermissionRequest =
50            registerForActivityResult(ActivityResultContracts.RequestMultiplePermissions()) { permissions ->
51                val fineGranted = permissions[Manifest.permission.ACCESS_FINE_LOCATION] ?: false
52                val coarseGranted = permissions[Manifest.permission.ACCESS_COARSE_LOCATION] ?: false
53
54                if (fineGranted || coarseGranted) { setupMapView() }
55                else { Toast.makeText(requireContext(), "위치 권한이 필요합니다", Toast.LENGTH_SHORT).show() }
56            }
```

그림 16. MapFrament.kt 코드일부 - 위치 권한 요청 코드 일부

지도 진입 시 런타임 권한 체계에 따라 위치 권한(FINE/COARSE)을 요청하고, 권한이 승인된 경우에만 지도 초기화 루틴(setupMapView())을 수행하도록 분기하였다. 권한이 거부되면 안내 토스트를 노출하여 필수 권한 요구를 명확히 한다.

지도 뷰는 mapView.start()로 시작되며, MapLifeCycleCallback으로 정상 종료 및 오류를 로그로 남기도록 구성하였다. 또한 준비가 완료되는 onMapReady 시점에만 후속 초기화(현재 위치 이동, 마커 스타일, 리스너 설정, 옵저빙)를 연결하고, 시작 완료 플래그(isMapViewStarted)를 기록한다.

3.2.2 현재 위치 획득 및 초기 카메라 이동

사용자에게 즉시 주변 카페 정보를 제공하기 위해, 앱 실행 직후 현재 위치를 획득하고 이를 기준으로 지도를 초기화하는 흐름을 구성하였다. 현재 위치 획득은 FusedLocationProviderClient.getCurrentLocation()을 사용하고, 초기 진입의 위치 정확도를 우선하여 [22]PRIORITY_HIGH_ACCURACY를 지정하였다. 위치 획득 성공 시 해당 좌표로 카메라를 이동시키고, 별도의 현재 위치 라벨(그림 17)을 추가하여 사용자 탐색의 기준점을 시각적으로 제공한다.

22) Google Developers,「LocationRequest 클래스 참조(LocationRequest)」

```kotlin
        /**
         * 카카오맵 뷰 설정
         */
        private fun setupMapView(){
            binding.mapView.start(object : MapLifeCycleCallback() {
                override fun onMapDestroy() {
                    Log.d("kakaoMap", "카카오맵 정상종료")
                }
                override fun onMapError(p0: Exception?) {
                    p0.let { Log.e("kakaoMap", p0.toString()) }
                }
            }, object : KakaoMapReadyCallback() {
                override fun onMapReady(p0: KakaoMap) {
                    Log.d("kakaoMap", "카카오맵 정상실행")
                    moveMapToCurrentLocation(p0)    // 현재 위치로 이동
                    initMarkerStyles(p0)            // 마커 스타일 초기화
                    setupLabelClickListener(p0)     // 마커 클릭 리스너 설정
                    setupCameraMoveEndListener(p0)  // 카메라 이동 리스너 설정
                    observeCafes(p0)                // 카카오 API 카페 리스트 옵저빙
                    observeServerCafes(p0)          // 서버 API 카페 리스트 옵저빙
                    isMapViewStarted = true         // 카카오맵 뷰 시작 완료
                }
            })
        }
```

그림 17. MapFrament.kt 코드일부 - KakaoMap 초기화 코드 일부

이 초기 위치가 확보되는 즉시 서버 기반 카페 목록 로드(loadServerCafes())를 호출하여 첫 화면에서 주변 카페 마커가 바로 표시되도록 흐름을 구성하였다.

그림 18. 녹색 마커 - 사용자 현재 위치 라벨

3.2.3 카메라 이동 종료 기반 재검색 및 중복 요청 방지

지도 기반 탐색에서는 사용자의 드래그/줌 동작이 빈번하기 때문에, 카메라 이동 중간 단계에서 매번 네트워크 요청을 수행할 경우 불필요한 트래픽과 배터리 소모가 발생할 수 있다. 이를 완화하기 위해 본 구현은 카카오맵의 카메라 이동 종료 이벤트(setOnCameraMoveEndListener)를 기준으로 서버 카페 목록을 재조회하도록 구성하였다.

```kotlin
    /**
     * 카메라 이동 리스너 설정
     */
    private fun setupCameraMoveEndListener(map: KakaoMap) {
        map.setOnCameraMoveEndListener { _, cameraPosition, _ ->
            val center = cameraPosition.position

            // 동일 좌표 중복 요청 방지
            if (lastRequestedLatLng != null &&
                lastRequestedLatLng!!.latitude == center.latitude &&
                lastRequestedLatLng!!.longitude == center.longitude) return@setOnCameraMoveEndListener

            lastRequestedLatLng = center

            // viewModel.loadCafes(center.latitude, center.longitude)       // 카카오 API 카페 리스트 요청
            viewModel.loadServerCafes(center.latitude, center.longitude)    // 서버 API 카페 리스트 요청
        }
    }
```

그림 19. MapFrament.kt 코드일부 - 카메라 이동 리스너 코드 일부

즉, 사용자가 지도를 조작하는 동안에는 요청을 발생시키지 않고, 조작이 완료되어 중심 좌표가 확정되는 시점에만 카페 리스트 갱신이 수행된다. 또한 직전 요청 중심 좌표(lastRequestedLatLng)를 저장하고, 새로 계산된 중심 좌표와 동일한 경우에는 재호출을 수행하지 않도록 하여 중복 요청을 최소화하였다.

3.2.4 지도 라이프사이클 제어

Fragment 생명주기에서 onResume()/onPause() 시점에 mapView의 resume()/pause()를 호출하되, 아직 start되지 않은 상태에서의 호출을 피하기 위해 isMapViewStarted를 조건으로 두어 안정성을 확보하였다.

```
392         override fun onResume() {
393             super.onResume()
394             if (isMapViewStarted) {
395                 binding.mapView.resume()
396             } else {
397                 Log.d("MapFragment", "MapView resume 생략 - 아직 start되지 않음")
398             }
399         }
400
401         override fun onPause() {
402             super.onPause()
403             if (isMapViewStarted) {
404                 binding.mapView.pause()
405             } else {
406                 Log.d("MapFragment", "MapView pause 생략 - 아직 start되지 않음")
407             }
408         }
```

그림 20. MapFrament.kt 코드일부 - Fragment 생명주기 관리코드 일부

3.3 카페 상세정보/리뷰 UI 및 좌석 확인

3.3.0 기능 개요

본 기능은 지도에서 사용자가 특정 카페를 선택했을 때, 지도 맥락을 유지한 채 상세 정보를 빠르게 확인하고(매장명/주소/전화/평점/이미지), 상세정보·리뷰를 탭으로 전환하며(그림 20, 21), 최종적으로 좌석 확인 화면(그림 22)으로 진입하도록 하는 중간 허브 UI이다. 구현은 BottomSheet-DialogFragment 기반으로 구성되며, 내부 콘텐츠는 ViewPager2, TabLayoutMediator 조합으로 구성하였다.

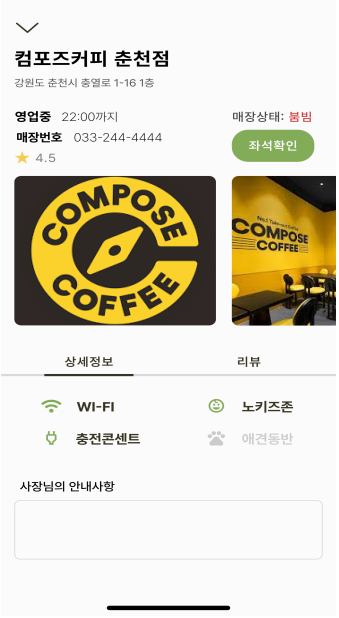

그림 21. 카페 상세정보

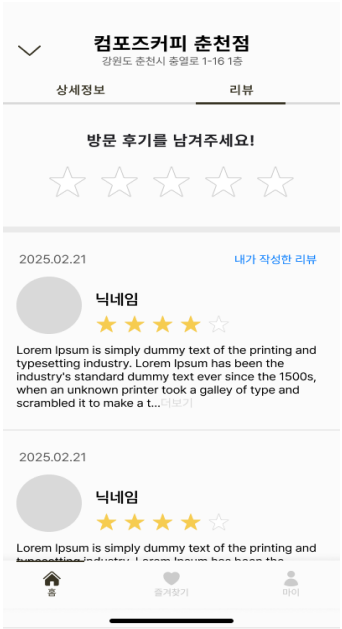

그림 22. 카페 리뷰

그림 23. 좌석확인

3.3.1 BottomSheet 기반 카페 상세정보 화면 구조

카페 상세정보 화면은 지도 탐색 맥락을 유지한 상태에서 자연스럽게 확인할 수 있도록 BottomSheetDialogFragment를 기반으로 구성하였다. 사용자는 마커 선택 후 하단에서 올라오는 BottomSheet를 통해(그림 13) 매장명, 주소, 연락처, 평점, 이미지 등의 정보를 확인할 수 있다.

또한 BottomSheet의 동작 상태에 따라 UI 요소를 단계적으로 전환하여, 현재 화면 상태를 사용자에게 명확히 전달하도록 설계하였다(그림 23). 초기 상태는 STATE_COLLAPSED로 설정하고, peekHeight를 고정값으로 지정하여 지도 위에 과도한 정보가 노출되지 않도록 하였다. 사용자가 BottomSheet를 확장하여 STATE_EXPANDED 상태에 진입하면, 상단 인디케이터를 숨기고 뒤로가기 버튼(btnBack)을 노출함으로써 상세 정보 탐색 모드로 전환되었음을 인지할 수 있도록 하였다(그

림 22). 반대로 BottomSheet가 다시 축소되어 STATE_COLLAPSED 상태로 전환될 경우에는 인디케이터를 표시하고 뒤로가기 버튼을 숨겨, 지도 탐색 중심의 화면으로 복귀했음을 시각적으로 명확히 표현한다.

```
207        private fun attachBottomSheetCallback(bottom: View) {
208            behavior.addBottomSheetCallback(object : BottomSheetBehavior.BottomSheetCallback() {
209                override fun onStateChanged(bottomSheet: View, newState: Int) {
210                    when (newState) {
211                        BottomSheetBehavior.STATE_EXPANDED -> applyExpandedState(bottomSheet)
212                        BottomSheetBehavior.STATE_COLLAPSED -> applyCollapsedState(bottomSheet)
213                        BottomSheetBehavior.STATE_HIDDEN -> dismiss()
214                    }
215                }
216
217                override fun onSlide(bottomSheet: View, slideOffset: Float) {}
218            })
219        }
220
221        private fun applyExpandedState(bottom: View) {
222            binding.indicator.visibility = View.GONE
223            binding.btnBack.visibility = View.VISIBLE
224
225            bottom.layoutParams.height = ViewGroup.LayoutParams.MATCH_PARENT
226            bottom.requestLayout()
227        }
228
229        private fun applyCollapsedState(bottom: View) {
230            binding.indicator.visibility = View.VISIBLE
231            binding.btnBack.visibility = View.GONE
```

그림 24. CafeBottomSheetFragment.kt - BottomSheet 동작 상태에 따른 UI 전환

3.3.3 탭 UI(상세정보/리뷰) 구성: ViewPager2 + TabLayoutMediator

카페 상세정보 화면의 콘텐츠는 ViewPager2로 구성하고, TabLayoutMediator를 통해 탭 선택과 페이지 이동이 동기화되도록 구성하였다.

```
158        private fun setupViewPager(cafe: Cafe) {
159            val adapter = ViewPagerAdapter(this, cafe = cafe, cafeDetail = null)
160            binding.viewPager.adapter = adapter
161            attachTabs()
162        }
163
164        private fun setupViewPager(detail: CafeDetail) {
165            val adapter = ViewPagerAdapter(this, cafe = null, cafeDetail = detail)
166            binding.viewPager.adapter = adapter
167            attachTabs()
168        }
169
170        private fun attachTabs() {
171            binding.viewPager.isNestedScrollingEnabled = false
172            TabLayoutMediator(binding.tabLayout, binding.viewPager) { tab, position ->
173                tab.text = when (position) {
174                    0 -> "상세정보"
175                    1 -> "리뷰"
176                    else -> ""
177                }
178            }.attach()
179        }
```

그림 25. CafeBottomSheetFragment.kt - 탭 UI 구성

3.3.4 리뷰 탭 표시 및 별점 입력 UI

리뷰 탭은 RecyclerView, ListAdapter(DiffUtil) 기반으로 리뷰 목록을 표시한다. 리뷰 데이터는 ViewModel의 reviewList를 관찰하여 리스트 갱신 시 submitList()로 반영한다(그림25).

또한 별점 입력 UI(그림 21)는 1~5점 버튼을 제공하며, 사용자가 별점을 선택하면 즉시 선택 상태(채움/비움)를 반영하고, Handler를 통해 300ms 후 리뷰 작성 다이얼로그(CafeReview-WriteFragment)를 호출하도록 구성하였다(그림 26).

```
68            val adapter = CafeReviewAdapter()
69            binding.rvReviews.layoutManager = LinearLayoutManager(requireContext())
70            binding.rvReviews.adapter = adapter
71
72            cafeDetail?.let {
73                viewModel.loadReviewsFromDetail(it)
74            }
75
76            viewModel.reviewList.observe(viewLifecycleOwner) { list ->
77                adapter.submitList(list)
78            }
```

그림 26. CafeReviewFragment.kt - 리뷰 탭 초기화 및 리뷰 데이터 로드

3.3.5 좌석 확인 화면전환

BottomSheet(그림 20) 내 좌석확인 버튼(btnSeatCheck, 그림 27) 클릭 시 CafeDrawing-

```
83              starViews = listOf(
84                  binding.star1, binding.star2, binding.star3, binding.star4, binding.star5
85              )
86
87              starViews.forEachIndexed { index, starView ->
88                  starView.setOnClickListener {
89                      val selectedRating = index + 1
90                      updateStarRating(selectedRating)
91
92                      // 300ms 후 CafeReviewWriteFragment로 이동
93                      Handler(Looper.getMainLooper()).postDelayed({
94                          cafe?.let {
95                              CafeReviewWriteFragment.newInstance(it, selectedRating)
96                                  .show(childFragmentManager, "ReviewWriteDialog")
97                          } ?: cafeDetail?.let {
98                              CafeReviewWriteFragment.newInstance(it, selectedRating)
99                                  .show(childFragmentManager, "ReviewWriteDialog")
100                         }
101                     }, 300)
102                 }
```

그림 27. CafeReviewFragment.kt – 리뷰 별점 UI 및 리뷰 작성 다이얼로그 호출

Fragment(카페 좌석 확인 화면)로 전환한다. 이때 BottomSheet가 전달받은 인자(cafeId 또는 Cafe)를 그대로 다음 화면으로 전달하여 선택 컨텍스트를 유지한다. 화면 전환 직후에는 BottomSheet를 dismiss()하여 UI 중첩을 방지하였다.

그림 28.좌석 확인 버튼 UI

```
binding.btnSeatCheck.setOnClickListener {
    val fragment = CafeDrawingFragment().apply {
        arguments = Bundle().apply {
            if (cafeId != null && cafeId > 0) putInt(ARG_CAFE_ID, cafeId)
            else if (cafe != null) putSerializable(ARG_CAFE, cafe)
        }
    }
    requireActivity().supportFragmentManager.beginTransaction()
        .replace(R.id.fragment_container, fragment)
        .addToBackStack(null)
        .commit()

    dismiss() // BottomSheet 닫기
}
```

그림 29. CafeBottomSheetFragment.kt – 좌석확인버튼 이벤트리스너

3.4 관리자 도면 편집(Drag & Drop) 및 다층 관리 기능 구현

3.4.0 기능 개요

본 기능은 카페 운영 및 관리 관점에서 카페의 도면을 모바일에서 직접 구성하기 위한 화면이다(그림 29). 카페 관리자는 하단 팔레트에서 객체(의자/문/창문/화장실/카운터/테이블)를 선택해 도면 영역에 드롭하여 배치하고, 층을 추가/전환하며 다층 도면을 구성한 뒤, 제출 시 층별 객체 정보를 DTO(Data Transfer Object)로 추출하여 등록 흐름에 연결한다. 도면 편집 영역은 Custom CanvasView를 사용한다.

3.4.1 카페 도면 등록을 위한 CanvasView 및 오브젝트 팔레트

서비스 운영을 위해서는 매장 도면 데이터가 선행되어야 하므로, 관리자가 모바일에서 직접 도면을 구성할 수 있는 편집 UI를 제공하였다. 본 구현에서는 CanvasView로 구성하여 객체 배치 및 편집 입력을 처리하도록 했고, 하단에는 RecyclerView 기반의 오브젝트 팔레트를 배치하여 좌석, 문, 창문, 화장실, 카운터, 테이블의 요소를 제공하였다(그림 30).

관리자는 팔레트에서 원하는 요소를 선택한 뒤 Drag & Drop 방식으로 도면 영역(canvas)에 배치할 수 있다. 이는 별도의 외부 도면 제작 도구를 사용하거나 복잡한 입력 절차를 거치지 않더라도 도면을 생성할 수 있게 하여, 도면 구축 비용과 운영 부담을 낮추는 방향으로 설계한 것이다.

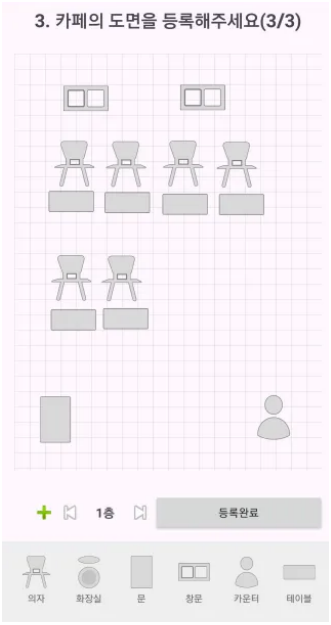

그림 30. 카페 도면등록

```
private fun setupObjectRecyclerView() {
    binding.objectRecyclerView.layoutManager =
        LinearLayoutManager(requireContext(), LinearLayoutManager.HORIZONTAL, false)

    binding.objectRecyclerView.adapter = ObjectListAdapter().apply {
        val objectList = listOf(
            ObjectItem("chair", "의자", com.binbean.ui.R.drawable.obj_seat),
            ObjectItem("toilet", "화장실", com.binbean.ui.R.drawable.obj_toilet),
            ObjectItem("door", "문", com.binbean.ui.R.drawable.obj_door),
            ObjectItem("window", "창문", com.binbean.ui.R.drawable.obj_window),
            ObjectItem("counter", "카운터", com.binbean.ui.R.drawable.obj_casher),
            ObjectItem("table", "테이블", com.binbean.ui.R.drawable.obj_table)
        )
        submitList(objectList)
    }
}
```

그림 31. RegisterDrawingFragment.kt – 오브젝트 팔레트 구현을 위한 Recyclerview

```
private fun setupObjectDragAndDrop() {
    val dragListener = View.OnDragListener { _, event ->
        when (event.action) {
            DragEvent.ACTION_DROP -> {
                val type = event.clipData.getItemAt(0).text.toString()
                addObjectView(type, event.x, event.y)
            }
        }
        true
    }
    binding.drawingContainer.setOnDragListener(dragListener)
}
```

그림 32. RegisterDrawingFragment.kt - Drag & Drop 구현

구현 측면에서는 도면 영역에 OnDragListener를 등록하고, 드롭 이벤트(ACTION_DROP)가 발생하면 전달된 데이터에서 객체 타입 문자열을 읽어 좌표와 함께 addObjectView(type, x, y)로 배치 로직을 위임한다. 이를 통해 팔레트에서 선택한 요소가 사용자의 드롭 위치에 따라 도면에 생성되며, 관리자는 필요한 구성 요소를 반복적으로 배치하여 도면을 완성할 수 있다.

3.4.2 좌표 정규화 및 그리드 스냅

객체 팔레트에서 객체를 드래그하여 도면 영역에 드롭하면, DragEvent가 제공하는 (x,y)는 현재 화면(View) 좌표계 기준의 픽셀 값이다. 그러나 이 값은 (1) 단말 해상도, (2) 도면 컨테이너의 크기/여백, (3) CanvasView 내부 스크롤·패딩·좌우 오프셋 등의 UI 배치 조건에 따라 달라질 수 있어, 그대로 저장할 경우 층 전환·재진입·재렌더링 시 동일 위치 재현이 어렵다. 따라서 드롭 좌표를 화

면 픽셀이 아니라 도면 내부의 기준 좌표계로 변환한 뒤 저장·처리하도록 설계하였다.

본 구현이 채택한 도면 기준 좌표계는 그리드(Grid) 단위 좌표이며, 그리드 간격을 50f로 고정하여 객체 위치를 (gridX, gridY) 형태로 정규화한다. 정규화는 다음의 두 단계로 수행된다.

(1) CanvasView 기준으로 원점 보정

드롭 좌표 x는 컨테이너 기준으로 들어오므로, 실제 도면 좌표로 해석하기 위해 CanvasView 내부 도면 원점이 화면에서 시작하는 위치(좌측 오프셋)를 제거해야 한다. 이를 위해 x에서 canvasView.getOffsetX()를 차감한다.

- 보정 후 좌표: x' = x - offsetX

(2) 그리드 단위로 스케일 변환

보정된 좌표를 그리드 간격으로 나누어 픽셀 기반 위치를 그리드 인덱스 공간으로 변환한다.

- gridX = x' / 50f, gridY = y / 50f

```kotlin
private fun addObjectView(type: String, x: Float, y: Float) {
    val canvasView = binding.drawingContainer.getChildAt(0) as? CanvasView ?: return

    // (1) CanvasView 기준 원점 보정 (x축)
    val gridX = (x - canvasView.getOffsetX()) / 50f

    // (2) 그리드 단위 변환 (y축)
    val gridY = y / 50f

    val objectType = when (type) {
        "chair" -> CanvasView.ObjectType.SEAT
        "door" -> CanvasView.ObjectType.DOOR
        "toilet" -> CanvasView.ObjectType.TOILET
        "counter" -> CanvasView.ObjectType.COUNTER
        "window" -> CanvasView.ObjectType.WINDOW
        "table" -> CanvasView.ObjectType.TABLE
        else -> return
    }

    canvasView.addInteractiveObject(gridX, gridY, objectType)
```

그림 33. RegisterDrawingFragment.kt – 좌표 보정 코드 일부

(드롭 좌표의 재렌더링 동작을 실험적으로 검증한 결과, X축 좌표는 CanvasView의 좌우 여백 및 내부 오프셋 변화에 따라 재표시 시 위치 편차가 발생한 반면, Y축 좌표는 도면 재진입 및 층 전환 이후에도 동일한 위치에 안정적으로 재현됨을 확인하였다. 이에 따라 X축에 대해서만 offsetX 기반 보정을 적용하고, Y축은 별도의 오프셋 보정 없이 그리드 단위 변환만 수행하도록 설계하였다.)

3.4.3 다층 도면 관리

실제 카페는 2층 이상 구조를 가질 수 있으므로, 단일 도면으로는 운영 요구사항을 충족하기 어렵다. 이를 해결하기 위해 층별 CanvasView 인스턴스를 Map<floorNumber, CanvasView>로 관리하여, 층 전환 시에도 기존에 작성된 도면 상태가 유지되도록 구현하였다. 사용자가 다음 층으로 이동할 때 이미 존재하는 층이면 즉시 전환하고, 아직 생성되지 않은 층이라면 AlertDialog로 추가 여부를 확인받은 뒤 신규 도면을 생성하도록 흐름을 구성하였다.

```kotlin
126         private fun setupFloorControl() {
127             binding.btnNextFloor.setOnClickListener {
128                 val nextFloor = currentFloor + 1
129
130                 if (floorViews.containsKey(nextFloor)) {
131                     // 이미 존재하는 층 → 바로 이동
132                     currentFloor = nextFloor
133                     showFloor(currentFloor)
134                     Toast.makeText(requireContext(), "${currentFloor}층으로 이동했습니다.", Toast.LENGTH_SHORT).show()
135                 } else {
136                     // 새로운 층 → 알림 띄우기
137                     AlertDialog.Builder(requireContext())
138                         .setTitle("도면 추가")
139                         .setMessage("${nextFloor}층 도면을 추가하시겠습니까?")
140                         .setPositiveButton("추가") { _, _ ->
141                             currentFloor = nextFloor
142                             showFloor(currentFloor)
143                             Toast.makeText(requireContext(), "${currentFloor}층 도면이 추가되었습니다.", Toast.LENGTH_SHORT)
144                                 .show()
145                         }
146                         .setNegativeButton("취소", null)
147                         .show()
148                 }
149             }
```

그림 34. RegisterDrawingFragment.kt – 층 추가 로직 일부

3.4.4 층별 DTO 추출 및 등록 흐름 연동

제출 시점에는 floorViews에 존재하는 층들을 순회하며 각 층의 CanvasView에서 extractObjects-ForSave()를 호출해 FloorDetail을 추출하고, 이를 FloorWrapper로 패키징한다.

```
160         private fun extractFloorDataToDto(floor: Int): FloorWrapper {
161             val canvasView = floorViews[floor] ?: return FloorWrapper(
162                 floorList = FloorDetail(emptyList(), emptyList(), emptyList(), emptyList(), emptyList(), emptyList()),
163                 floorNumber = floor,
164                 maxSeats = 0
165             )
166
167             val floorDetail = canvasView.extractObjectsForSave()
168
169             return FloorWrapper(
170                 floorList = floorDetail,
171                 floorNumber = floor,
172                 maxSeats = floorDetail.seatPosition.size
173             )
174         }
```

그림 35. RegisterDrawingFragment.kt - 층별 데이터 추출 로직 일부

IV. 결과 평가

4.1 기능 요구사항 충족 여부

본 구현은 위치 기반 카페 탐색, 상세 정보 제공, 좌석 확인 화면 진입, 관리자 도면 편집이라는 핵심 기능 요구사항을 전반적으로 충족하였다. 먼저 지도 탐색 측면에서는 사용자의 현재 위치를 기반으로 초기 카메라를 이동시키고, 주변 카페를 마커로 표시하였으며, 카메라 이동이 종료되는 시점에 서버 카페 목록을 재조회하도록 구성하여 기본적인 LBS(Location Based Service) 사용자 경험 흐름을 구현하였다. 카페 상세 화면은 BottomSheet 기반으로 구성하여 지도 화면의 맥락을 유지한 채 카페 정보를 확인할 수 있도록 했고, 상세정보/리뷰를 탭으로 분리하여 정보 구조를 명확히 하였다. 좌석 확인 기능은 상세 화면에서 선택한 카페 컨텍스트(cafeId 등)를 유지한 채 좌석 도면 화면으로 전환하도록 내비게이션을 구성하여, 카페 탐색-상세-좌석 확인으로 이어지는 사용자 여정을 끊김없이 연결하였다. 관리자 기능에서는 Drag&Drop 기반 객체 배치를 제공하고, 층전환 및 층 추가를 포함한 다층 도면 관리를 구현하였으며, 최종적으로 도면 객체 정보를 DTO로 추출하여 등록 API 호출 흐름과 연계함으로써 운영 관점에서 필요한 핵심 입력 데이터 생성 과정을 완성하였다.

4.2 한계점 및 개선 방향

구현의 정교성과 확장성 관점에서 개선 여지도 존재한다. 우선 카페 지도화면에서 마커 렌더링은 현재 갱신 시점에 해당 타입 마커를 순회 제거 후 재추가하는 구조이므로, 카페 수가 증가할 경우 성능 저하가 발생할 가능성이 있다. 향후에는 diff 기반 업데이트로 변경된 항목만 반영해 렌더링 비용을 줄일 수 있다. 관리자 도면 편집 기능 역시 현재는 배치 중심 기능에 집중되어 있어 객체 크기 조절, 회전과 같은 편집 기능이 제한적이며, 실제 운영 환경에서는 이러한 편집 기능이 필수적으로 요구될 가능성이 높다. 따라서 제스처 처리 로직을 추가하여 편집 정밀도를 높이는 방향이 바람직하다. 마지막으로 좌석 시각화 결과는 AI 매핑의 불확실성(카메라 왜곡, 가림 등)의 영향을 받을 수 있으므로, 사용자에게 단순 점유 표시뿐 아니라 신뢰도 혹은 최근 업데이트 시각과 같은 메타 정보를 함께 제공하면 서비스 신뢰성과 사용성이 개선될 수 있다.

V. 결 론

본 캡스톤 프로젝트는 CCTV 기반 AI 멀티모달 분석을 통해 카페 좌석 점유를 추정하고 이를 도면 형태로 제공하는 실시간 좌석 시각화 서비스를 구현하였다. 개인 구현 관점에서 본인은 사용자가 서비스를 실제로 이용하는 시작점인 사용자 위치 기반 지도 탐색, 카페 상세/리뷰 UI, 좌석 확인 화면 진입 흐름을 완성하고, 서비스 운영에 필수적인 관리자 도면 편집 및 다층 도면 저장 DTO 생성을 구현하였다.

이를 통해 (1) 사용자는 지도에서 매장을 탐색하고 좌석 확인까지 자연스럽게 연결되는 UX를 확보하였고, (2) 운영자는 별도 하드웨어 설치 없이 도면 데이터를 구축하여 AI 좌석 시각화 기능의 기반을 마련할 수 있었다. 향후에는 도면 편집의 정교화, 지도 데이터 갱신 최적화, AI 결과 신뢰도 표현 및 예측 기능 확장 등을 통해 상용 수준의 완성도로 발전시킬 수 있다.

VI. 참 고 문 헌

1. Google Developers, FusedLocationProviderClient (Google Play services Location), API Reference.
2. Android Developers, Request location permissions (위치 권한 요청 가이드).
3. Android Developers, UI layer / App architecture (UI 레이어 가이드).
4. Android Developers, State holders and UI state (상태 홀더/상태 관리).
5. Android Developers, ViewModel overview (ViewModel 개요).
6. Android Developers, LiveData overview (LiveData 개요).
7. Android Developers, Create dynamic lists with RecyclerView (RecyclerView 가이드).
8. Android Developers, Implement drag and drop with views (View 기반 Drag & Drop).
9. Android Developers, Track touch and pointer movements (터치 이동 추적/ACTION_MOVE).
10. Android Developers, Create swipe views with tabs using ViewPager2 (스와이프 뷰/탭 구성).
11. Android Developers, TabLayoutMediator API Reference.
12. Android Developers, BottomSheetDialogFragment API Reference (Material Components).
13. Kakao Developers, KakaoMaps SDK v2 for Android – Docs.
14. Kakao Developers, Gestures & Events (제스처/이벤트 가이드).
15. Kakao Developers, Camera (카메라/이동 이벤트 관련).
16. Kakao Developers, 개발 주의사항(Precautions) (MapView 라이프사이클).
17. Kakao Developers, Change Logs(Release notes).
18. Amazon Web Services, Amazon Rekognition – DetectLabels API Reference.