키오스크 앱 변환

1. ⚙️ 현재 코드의 전체 구성

현재 코드는 **Jetpack Compose**의 권장 아키텍처인 **MVVM (Model-View-ViewModel)** 패턴을 따르고 있으며, 햄버거와 카페 두 가지 매장을 유연하게 지원하도록 완성되었습니다.

영역	패키지/파일	역할
Data (모델)	`data/model/*`	앱에서 사용하는 모든 데이터 구조 (메뉴, 카트, 미션, `KioskType` 등) 정의.
Data (저장소)	`data/repository/HistoryRepository.kt`	학습 기록을 기기 내부 저장소에 저장하고 불러오는 로직 담당.
ViewModel	`ui/screens/simulator/KioskViewModel.kt`	UI의 **상태와 로직** 관리. 키오스크 타입에 따라 메뉴 데이터를 분리하여 제공하고, 장바구니 및 옵션 가격 계산을 수행.
UI (화면)	`ui/screens/main/MainMenuScreen.kt`	앱의 진입 화면. **모드 선택 후 매장 선택 화면으로 전환**하는 로직 포함.
UI (시뮬레이터)	`ui/screens/simulator/KioskSimulatorScreen.kt`	**햄버거/카페 테마**를 동적으로 적용하여 메뉴 표시, 옵션 선택 팝업(`OptionDialog`) 관리.
UI (공통)	`ui/components/OptionDialog.kt` 외	메뉴 선택 시 **Hot/Ice 같은 옵션을 선택하는 팝업**을 구현.
메인 진입점	`MainActivity.kt`	앱 실행 후 화면 전환 (`ScreenState`)을 관리하고, `KioskSimulatorScreen` 에 현재 `KioskType` 을 전달.

2. 💻 사용된 언어 및 기술

- **주 언어**: **Kotlin (코틀린)**. 안드로이드 공식 개발 언어입니다.
- **UI 툴킷**: **Jetpack Compose**. 코틀린 코드로 UI를 선언적으로 구성하는 최신 프레임워크입니다. (React와 유사한 방식)

- **아키텍처**: **MVVM (Model-View-ViewModel)**. 코드를 데이터, 로직, 화면으로 명확하게 분리하여 유지보수를 쉽게 합니다.

---

3. 🚀 새로운 키오스크 종류 추가하는 법

현재 구조는 **새로운 매장(예: 분식점)**을 추가하는 데 매우 효율적입니다. 세 가지만 수정하면 됩니다.

단계 1: 데이터 모델에 새로운 타입 추가

`data/model/KioskType.kt` 파일을 열고, **'분식점'** 설정을 추가합니다.

```
// KioskType.kt 수정
enum class KioskType(
    val title: String,
    val themeColor: Color,
    val icon: String,
    val categories: List<String>
) {
    // ... BURGER, CAFE는 유지 ...
    BUNSIK(
        title = "분식점",
        primaryColor = Color(0xFFC04257), // 분홍색 계열
        icon = "🍜",
        categories = listOf("메인", "튀김", "음료")
    )
}
```

단계 2: ViewModel에 분식점 메뉴 추가

`ui/screens/simulator/KioskViewModel.kt` 파일에 분식 메뉴(`bunsikItems`)와 미션(`bunsikMissions`) 리스트를 정의하고, `getCurrentMenuItems()` 함수에 `when` 절을 추가합니다.

```
// KioskViewModel.kt (일부)

// 1. [추가] 분식 메뉴 데이터
private val bunsikItems = listOf(
    MenuItem("s1", "떡볶이", 4000, "메인"),
    MenuItem("s2", "순대", 3500, "메인"),
```

```
    MenuItem("s3", "튀김(모듬)", 4500, "튀김"),
    MenuItem("s4", "쿨피스", 2000, "음료"),
)
private val bunsikMissions = listOf(/* ... 분식점 미션 정의 ... */)


// 2. getCurrentMenuItems() 함수 수정
fun getCurrentMenuItems(): List<MenuItem> = when (currentType) {
    KioskType.BURGER -> burgerItems
    KioskType.CAFE -> cafeItems
    KioskType.BUNSIK -> bunsikItems // 👈 분식 메뉴 반환 추가
}

// 3. init 함수 (미션 설정)에도 분식점 분기 추가
fun init(isPractice: Boolean, type: KioskType) {
    // ...
    if (!isPractice) {
        val missions = when (type) {
            KioskType.BURGER -> burgerMissions
            KioskType.CAFE -> cafeMissions
            KioskType.BUNSIK -> bunsikMissions // 👈 미션 로드 추가
        }
        _currentMission.value = missions.random()
    }
    // ...
}
```

단계 3: UI 연결

앱을 실행하면, **매장 선택 화면에 '분식점' 버튼이 자동으로 추가됩니다.** (저희가 `KioskType.values().forEach { type -> ... }`를 사용했기 때문입니다.)

새로 추가된 '분식점' 버튼을 누르면, `KioskSimulatorScreen`이 자동으로 분식점의 테마 색상, 아이콘, 그리고 분식 메뉴를 보여주게 됩니다.