Project report

WIKIPEDIA SEARCH

작성자 **김용기**

Project report

CONTENTS

01 기획 : 커스텀 HTTP 모듈 기반 검색 앱 개발

02 요구 분석: Http, App 주요 요구사항 분석

03 설계 : 아키텍처, Http Module , Data flow

04 구현: 기술 스택 및 구조 구성

05 검증 : 테스트 전략 및 커버리지 확인

06 산출물: 결과물 정리 및 실행 결과

07 회고: 과제 수행 후기 및 개선점

WIKIPEDIA SEARCH – 커스텀 HTTP 모듈 기반 검색 앱 개발

개발 목적

- 커스텀 HTTP 라이브러리를 이용한 외부 API 통신 구조 설계
- Wikipedia Open API 연동을 통한 검색 기능 구현
- MVVM + 클린 아키텍처 기반의 구조적 데이터 처리 흐름 구성
- XML 기반 UI 구현 및 사용자 경험(UX) 반영
- 단위 테스트를 통한 모듈 안정성 및 예외 처리 검증

제출 항목

- APK 실행 파일 (Release, JAR File)
- 전체 소스 코드 (모듈별 구성 포함)
- 기술 문서(PPT, Testcase)

Custom HTTP **MVVM 구조** **모듈화 설계**

요구 분석 : 1단계 통신 모듈 요구사항 분석

기능 명	설명
HTTP/ HTTP 지원	HttpURLConnection, HttpsURLConnection 기반, Http, Https 지원 요구
Method / Header / Body 유연성	GET, POST 등 여러 Method 지원 및 Request Header Body 커스터마이징 요구
응답 포맷 유연성	Json, x-www-form-urlencoded, multipart/form-data등 응답 처리 가능 요구
TimeOut 및 기능성	API 접속 시 연결 / 응답 Timeout 설정 지원 필수
모듈성 및 재사용성	JAR 파일 구성을 통한 Library 제공 및 재사용 가능 구조 요구

고려사항

- 요청/응답 구조 명확화, Content-Type별 유연한 처리
- 예외, 타임아웃 대응 및 재사용 가능한 구조 설계
- HTTPS 인증 흐름 대응 포함
- 테스트 커버리지 확보 및 전략적 계획 필요

요구 분석 : APP 주요 기능 분석

기능 명	설명
검색 요청 처리	사용자가 입력한 키워드를 기반으로 Wikipedia API 호출
검색 결과 출력	수신된 JSON 응답을 DTO로 매핑 후 리스트에 출력
상세 이동 처리	각 항목 클릭 시 해당 문서 URL을 브라우저에 전달
오류 응답 처리	통신 실패 시 사용자에게 메시지(Toast 등) 제공
리스트 새로고침	SwipeRefreshLayout 기반으로 화면 갱신 처리

추가 설명

- 모든 기능은 ViewModel → Repository → DataSource 계층을 통해 흐름이 분리됨
- 커스텀 HTTP 모듈은 GET 요청 기준으로 동작, 비동기 처리 구조 적용
- 검색 결과는 List View 를 이용해 UI에 출력하며, 빈 상태/에러 상태 모두 처리됨

설계: Custom HTTP Module 설계

모듈명

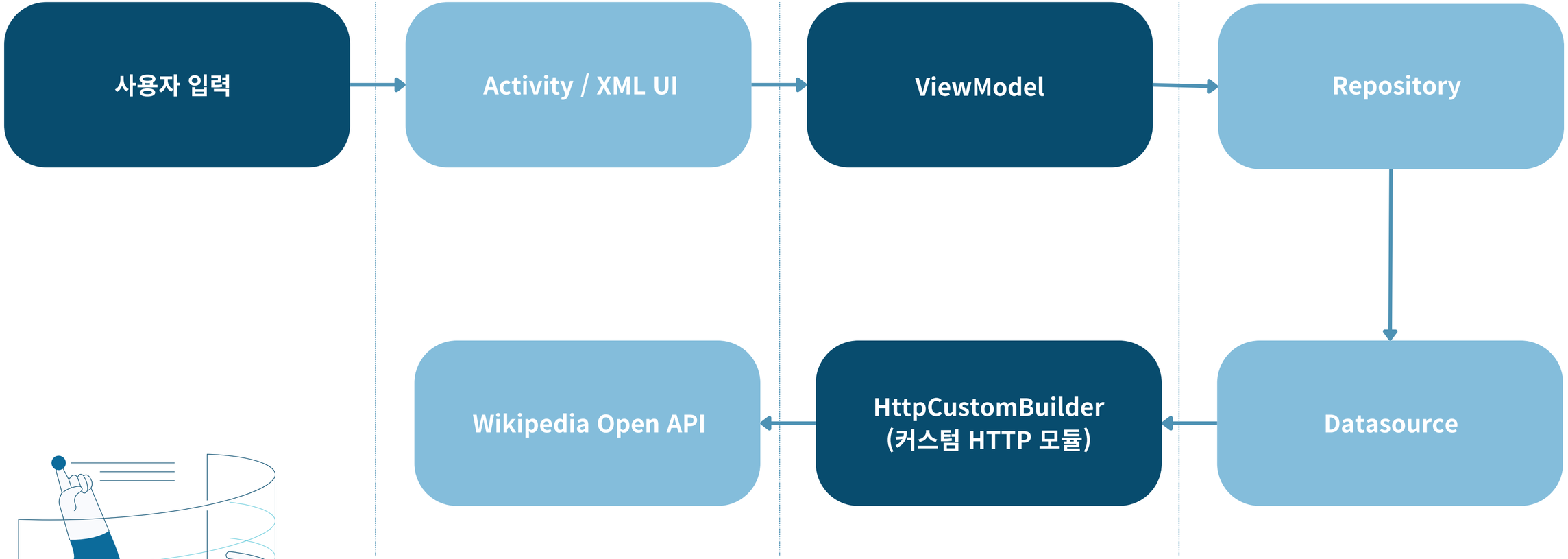
- HttpCustomBuilder

설계 의도

- 기성 HTTP 라이브러리 없이 HttpURLConnection을 기반으로 직접 통신 흐름과 설정을 제어할 수 있도록 설계
- 다양한 요청 방식(GET/POST/PUT/DELETE)을 하나의 모듈로 추상화
- Content-Type, Timeout, 요청 바디 포맷 등을 동적으로 분기 처리할 수 있도록 구성

항목	설명
지원 메서드	GET, POST, PUT, DELETE (동일 클래스 내 처리)
요청 설정	Content-Type, Charset, Timeout 등 유연한 커스터마이징
바디 처리	JSON / x-www-form-urlencoded / multipart-form-data 분기 처리
예외 처리	HTTP 오류 코드 → 사용자 메시지 변환 처리
재사용성	앱 외부 모듈로 분리 가능한 구조 (라이브러리화 가능)

설계 : System Architecture (아키텍처 구성 흐름도)



설계 : Layer Structure Define

01 **Application Layer**

Util

Injector, Utills, Application,등
전역 설정 및 보조 기능 제공

02 **Data Layer**

Data Source

DataSource, Service,
Repository, Local
API 통신, 캐싱, 이미지 로딩 등
최종 데이터 처리 담당

03 **Domain Layer**

Use Case

Search, Summary, Refresh등
비즈니스 로직 수행 및 Repository 연결

04 **Presentation Layer**

UI 중심

Aapter, ViewModel,
UIEffect, UiState, Activity
사용자와 직접 상호작용, UI 상태 관리

설계 : Data Flow

Web

Json
- title, snippet, pageId 등 Json 방식 전달

Data

DTO
- MediaListDto, SummaryDto 등으로 Json 전환

Domain

Entity
- Media, Summary 등 toDomain() 방식 파싱

Ui

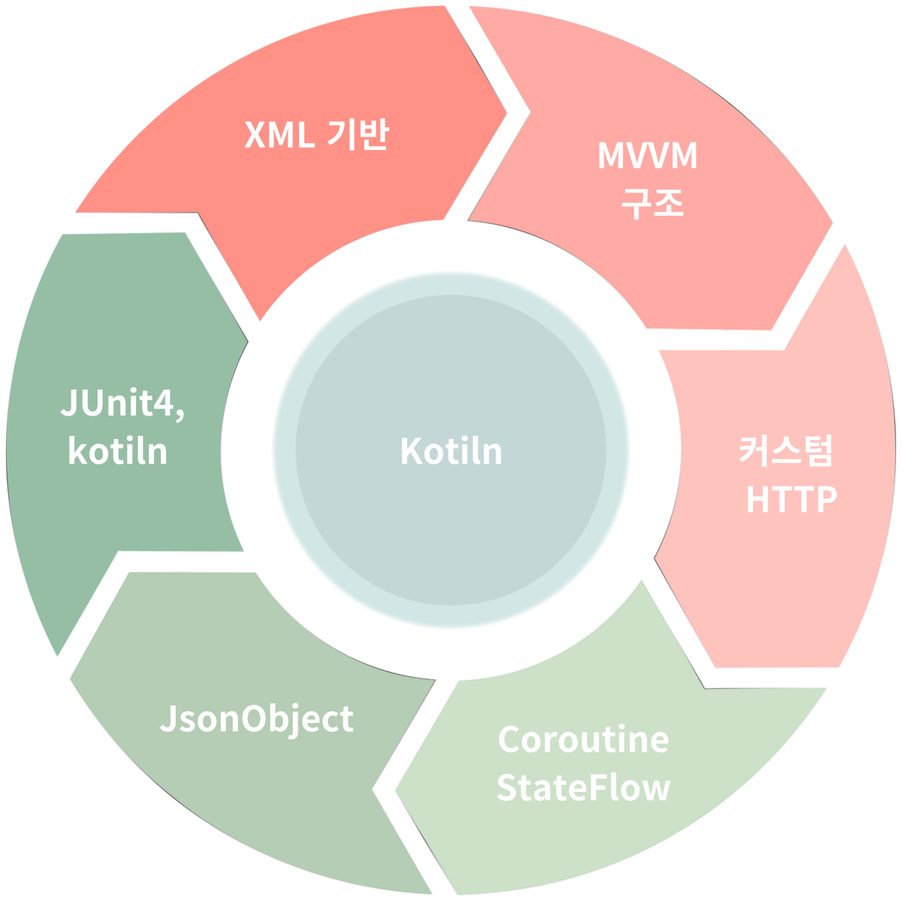
UiModel
MediaUiModel, SearchUiModel 등 Ui 설정 반응 용 모델

데이터는 계층별 책임 분리에 따라

DTO → Entity → UiModel로 변환되며,

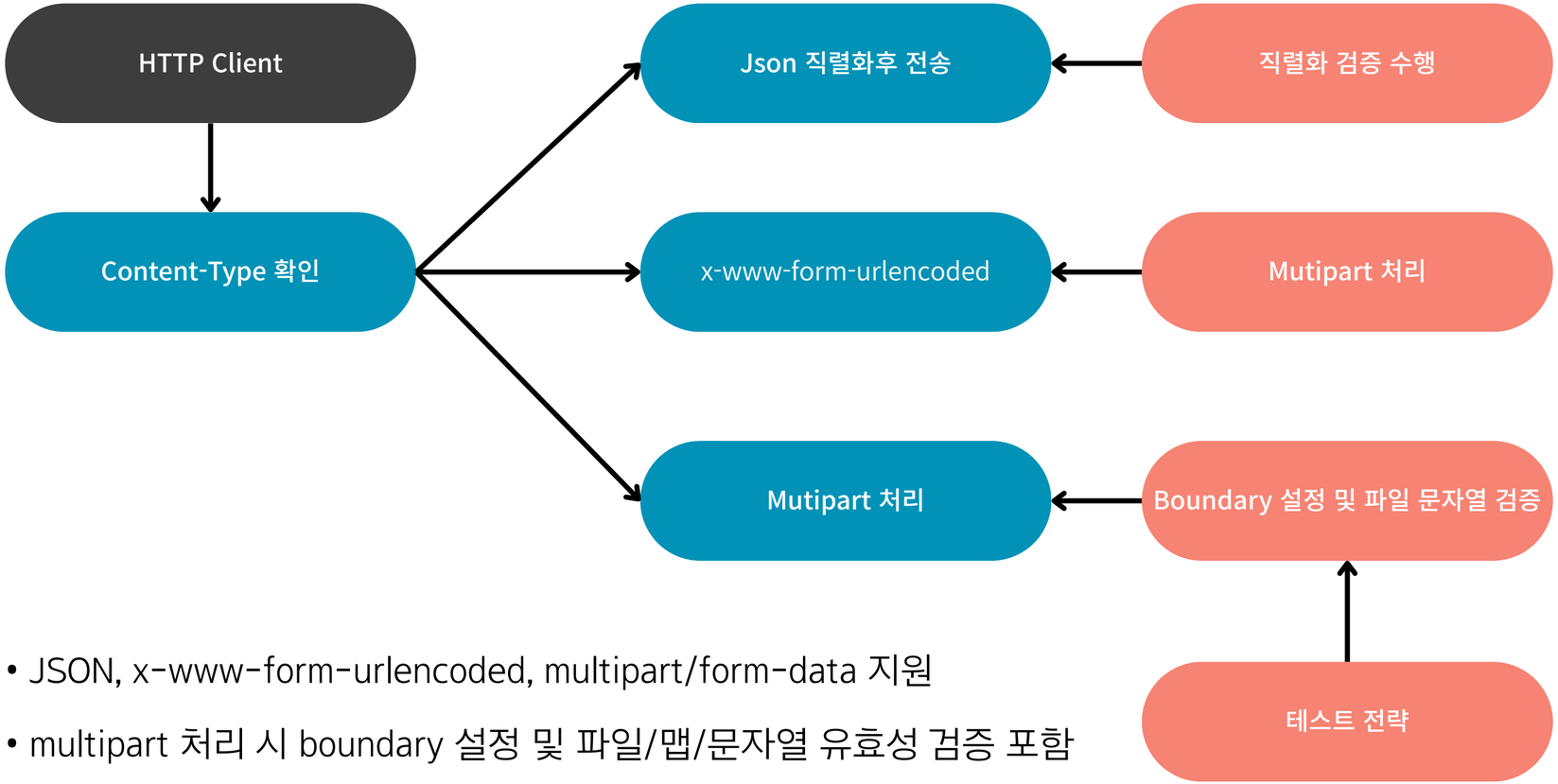
실제 출력은 XML 기반 커스텀 UI를 통해 처리됩니다.

구현 : Tech Stack



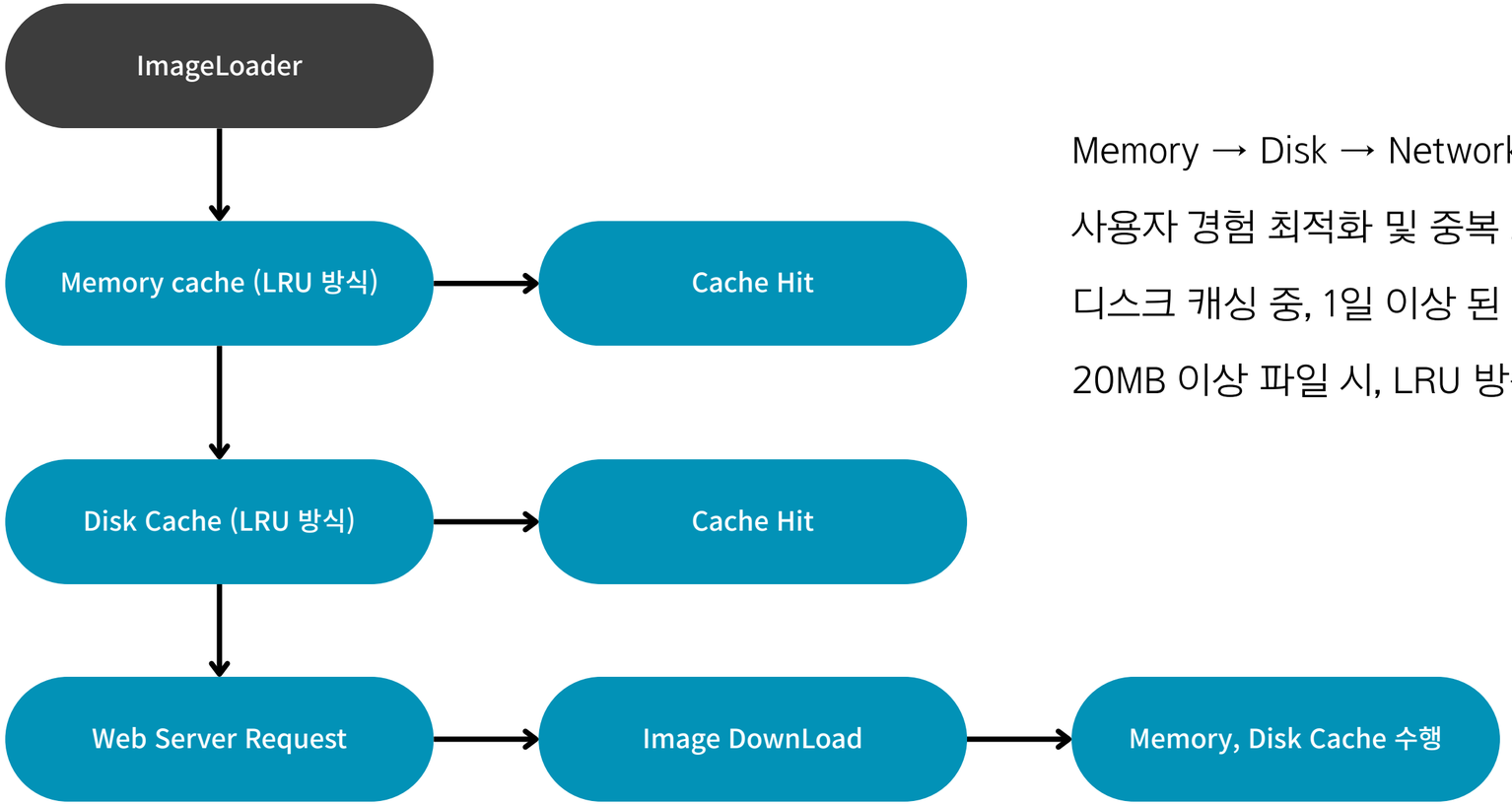
본 프로젝트는 커스텀 HTTP 모듈을 중심으로,
Kotlin 기반 MVVM + 클린 아키텍처와 XML
UI 구성, Coroutine 기반 비동기 처리 구조를
적용한 모듈화된 안드로이드 앱입니다.

구현 : Content-Type 자동 분기 처리 설계 및 검증



- JSON, x-www-form-urlencoded, multipart/form-data 지원
- multipart 처리 시 boundary 설정 및 파일/맵/문자열 유효성 검증 포함
- 각 분기 테스트 케이스 별도 구성 (Coverage 확보)

구현 : 이미지 캐싱



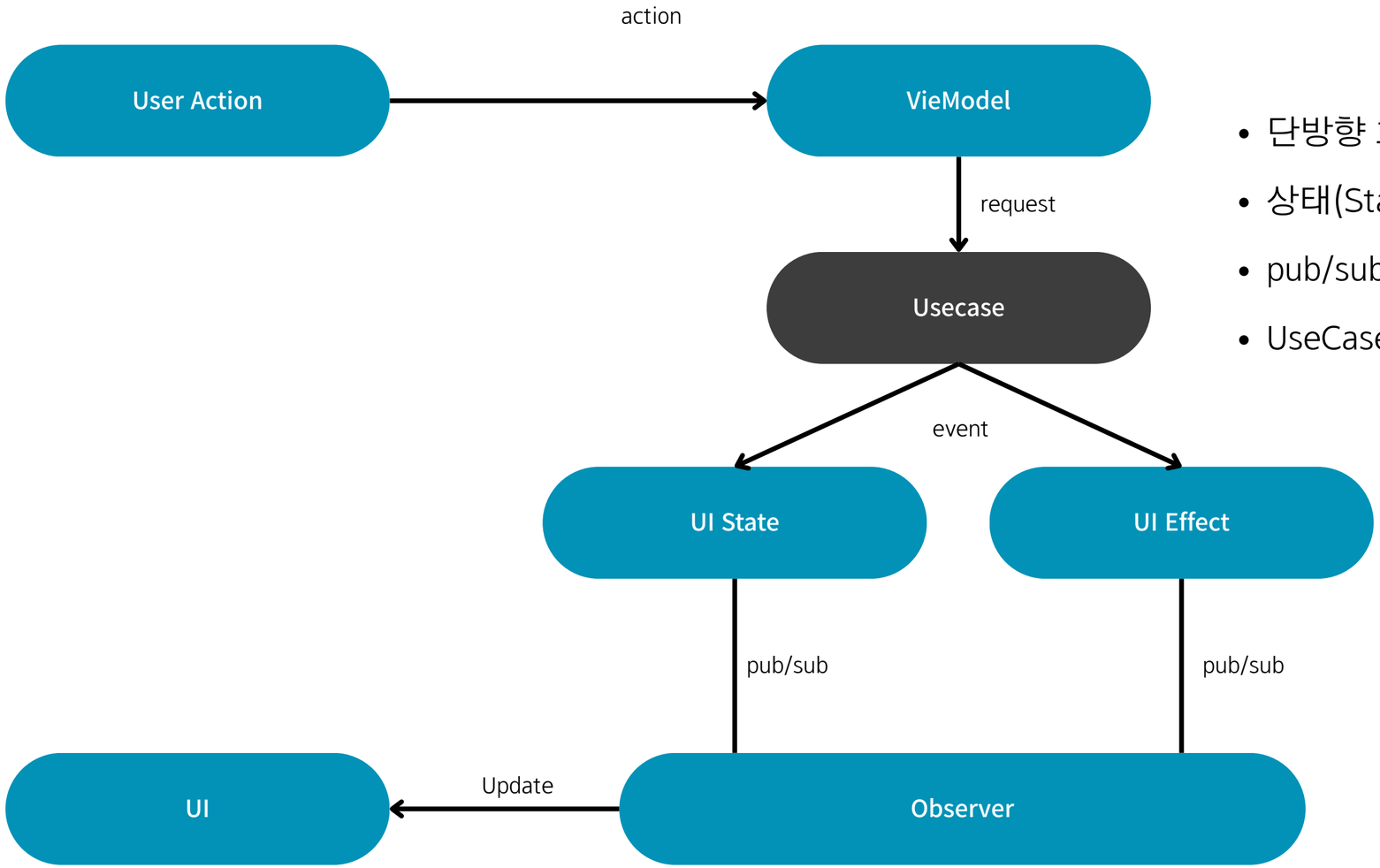
Memory → Disk → Network 순 캐시 처리 흐름 구성으로

사용자 경험 최적화 및 중복 요청 최소화

디스크 캐싱 중, 1일 이상 된 데이터 삭제 로직 구현

20MB 이상 파일 시, LRU 방식의 데이터 삭제

구현 : UI 상태 중심 구현



- 단방향 흐름 (ViewModel → State/Effect → UI)
- 상태(State) / 효과(Effect) 분리로 유지보수성 향상
- pub/sub 패턴으로 UI 반응 처리
- UseCase 중심 구조로 테스트 용이

테스트 : 예외 처리 및 테스트 전략

01

대상 선정 및 목표설정

테스트 대상

ViewModel, Repository,
Datasource, Data, 주요Model,
커스텀 HTTP 모듈
Disk cache, Memory cache

02

테스트 계획 및 구성

테스트 방식

JUnit4, Kotiln, Mock, 계측 테스트
Coroutine Test Dispatcher 활용

03

테스트 수행

주요 항목

정상 응답, 에러 응답,
타임아웃, Content-Type별 처리
Multipart, JSON,
x-www-form-urlencoded
400, 500 code 및 실측 테스트

04

검증 및 보고서 작성

커버리지

GET / POST / PUT / DELETE
전 요청 방식 검증 완료
Jacoco Html 작성

테스트 커버리지 분석

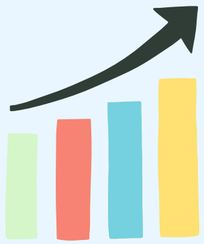
- customHttp 모듈은 단위 테스트를 집중적으로 수행하여
- Class / Method / Line 기준에서 90% 이상의 커버리지를 확보함
- 반면, wikipedia search 쪽은 UI 및 ViewModel 중심의 구조로 인해
- 일부 기능만 테스트되었고, 전반적인 커버리지는 낮은 편임
- 이번 과제를 통해 기능별 테스트 전략의 필요성과 균형에 대해
- 실제로 체감하고 보완 방향을 도출할 수 있었음

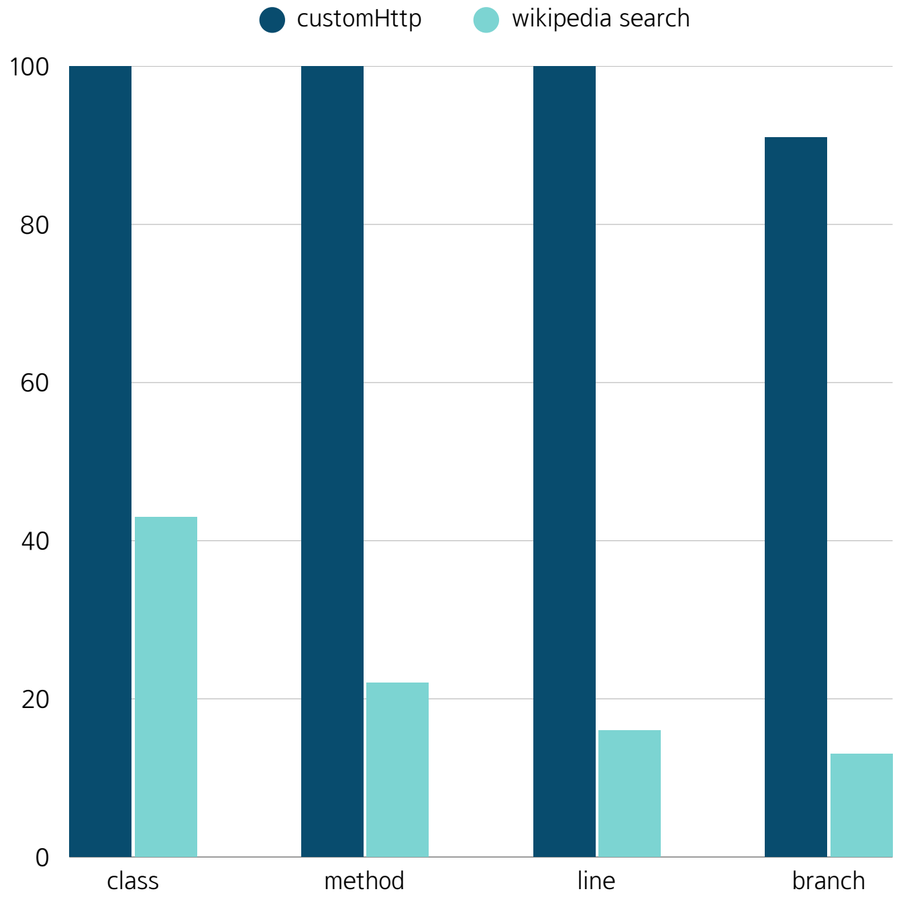
분석 결과

Dagger 못 써서 의존성 관리 아쉬움 있었다

기능 개발 후, 모듈간 종속성 문제로 인해

커버리지 확보가 힘들었음, TDD 위주의 개발을 했어야함





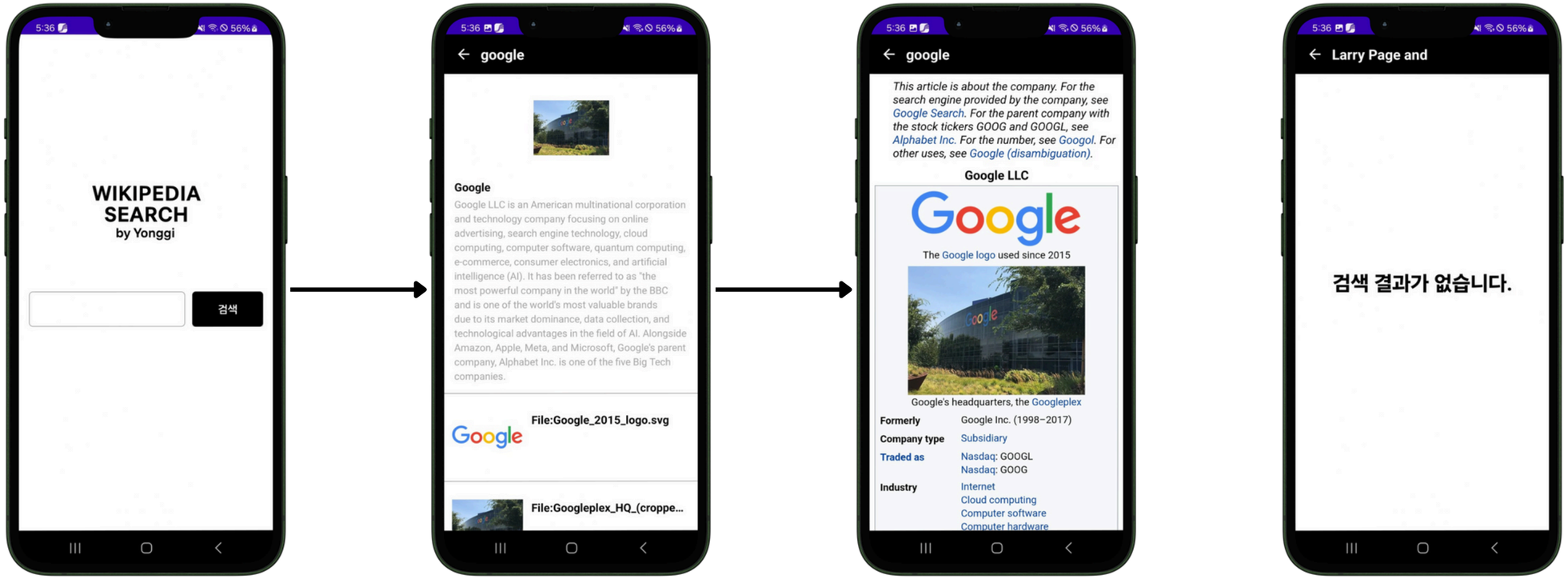
산출물 : 산출물 리스트

리스트	설명
APK 파일	Build File > wikipedia_search.apk, customHttp,jar
소스코드	WikipediaSearch (APP), CustomHttp (통신 모듈)
테스트 산출물	테스트 문서 > 테스트 케이스
커버리지 리포트	테스트 문서 > Wikipedia Search.Html, CustomHttp.Html, Jacoco HTML 리포트 제공
기술 문서	기술 문서 pdf

추가 설명

- MAC 환경에서 압축하여, 윈도우에서 압축이 깨질 수 있습니다
- 파일에 문제가 있다면, 재 요청 부탁드립니다.

산출물 : 완성된 APP UI 구조



사용자 요청에 따라 검색 → 결과 출력 → 상세 페이지 이동 → 예외 상황까지 전체 흐름을 설계 및 구현함

결과: 회고 및 향후 확장 방향

회고

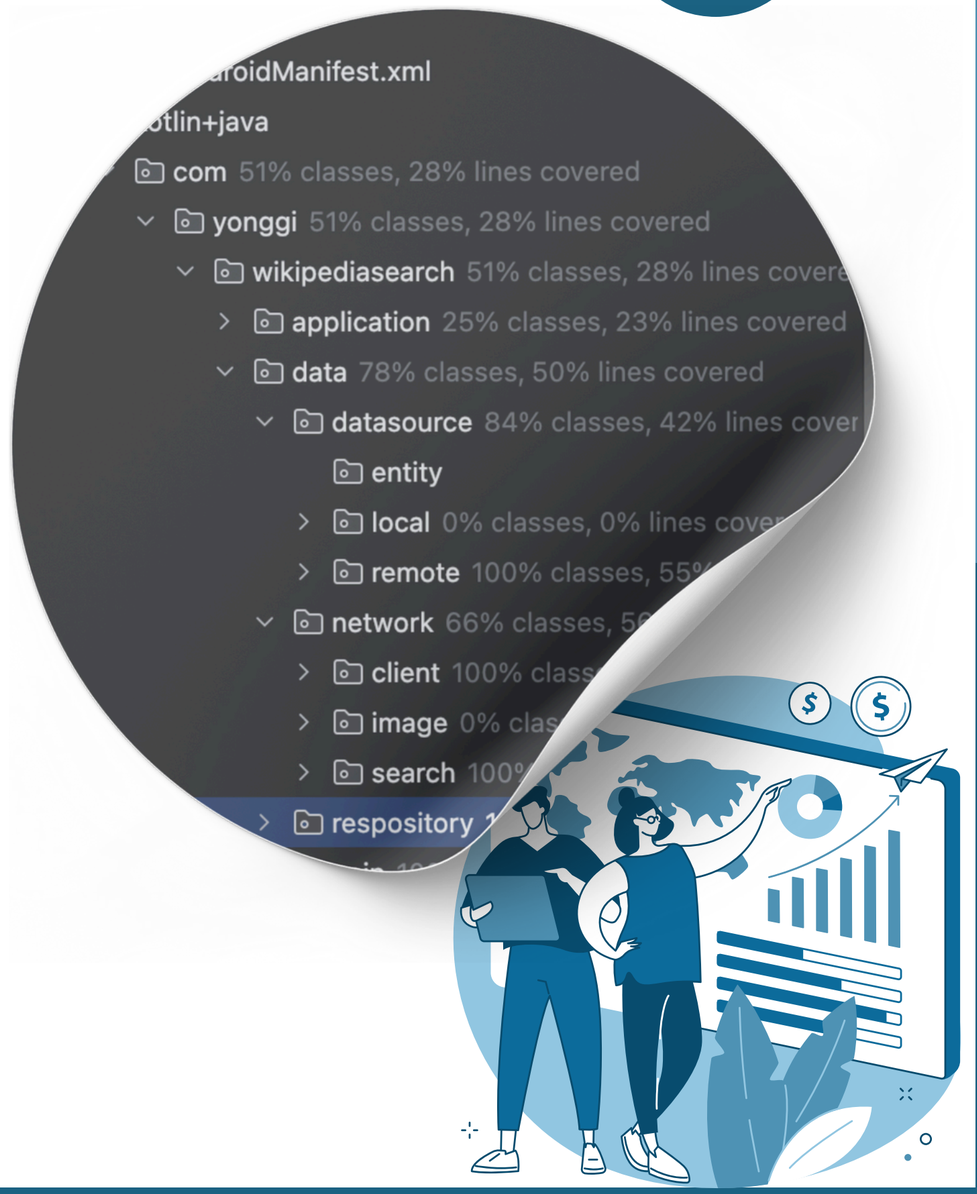
- 다양한 요청 방식과 Content-Type 분기 처리를 직접 구현하며
- HTTP 통신 로직의 구조화와 예외 처리를 실제로 설계해볼 수 있었음
- XML 기반 UI에서 MVVM 구조를 연결하면서
- ViewModel 중심의 상태 관리에 대한 이해도를 높일 수 있었음

아쉬운 점

- 오픈소스 제약으로 Dagger 등 DI 프레임워크를 사용하지 못해
- 의존성 주입 및 모듈 간 결합도 관리에 아쉬움이 있었음
- 기능 중심의 개발로 Disk Cache 처리나 전체 커버리지 확보에는 미흡한 부분이 있었음
- 일부 테스트는 계측 테스트(Instrumentation Test)로 구성되었으며,
- TDD 기반의 개발 방식 전환 필요성을 강하게 느꼈음
- UI 간 ViewModel 설계에서 역방향 종속성 문제가 일부 존재하며, 구조적 개선의 여지가 있음

확장 아이디어

- 키워드 자동완성 기능 도입
- RecyclerView diff 적용 + Paging 연동
- WebView 기반 상세보기 화면 커스터마이징
- 모듈을 Gradle로 외부 JAR 분리하여 공용화



Project
report

감 사 합 니 다

김용기 010-9655-1859 viccwe@naver.com