sweech 성능 테스트 리포트 — 1차 · v1.0

프로젝트: **FANUZU** iOS/Android 1차 정식 출시 버전 개발 근거: 별첨 1 ① 판정 기준「기준 단말에서 행성/맵/애니메이션 체감 지연 없음」· 별첨 1 ① 검수 방법「성능은 구간별 프레임 측정(성능 테스트 리포트 병행)」· 견적서 v2.1 7p「산출물 목록 · QA」작성: 스위치 주식회사(Sweech) · 2026-09-01

✦ 0. 요약

항목	측정값	판정
앱 시작 — 첫 프레임	**586ms** (중앙값 · 변동 ±4ms)	🟡 에뮬레이터 값 · 기준 단말 재측정 필요
갤럭시 맵 — UI 스레드 프레임	**평균 0.86ms** · p90 0.93ms	🟢 예산 16.7ms 의 6%
갤럭시 맵 — 래스터화	평균 14.65ms · p90 26.83ms	🟡 측정 환경 제약 (§1.2)
실제 내려받는 앱 크기 (arm64)	**23.2MB**	🟢
맵 그리기 최적화	래스터 예산 초과 **95회 → 59회**	🟢 개선 확인

이번 측정에서 최적화를 한 건 했습니다. 갤럭시 맵의 별밭 220개가 궤도가 도는 매 프레임 다시 그려지고 있었습니다. 배경을 별도 층으로 갈라 한 번만 굽게 바꾸었고, 프레임 예산 초과가 142프레임 중 95회에서 59회로 줄었습니다(§3.3).

✦ 1. 측정 환경

1.1 구성

항목	값
빌드	Flutter 3.47.2 · Dart 3.13.2 · **profile 모드**
측정 기기	Android 에뮬레이터 · API 35 · arm64-v8a · Google APIs
호스트	macOS 27.0 · Apple Silicon
GPU	**swiftshader (소프트웨어 렌더러)**
측정 도구	`flutter drive` + `integration_test` 프레임 타이밍 · `--trace-startup`
측정 회차	프레임 4회 · 시작 시간 3회 · 그리기 비용 60표본

profile 모드로 잰 이유 — debug 빌드는 JIT 로 돌고 관측 코드가 붙어 있어 실제보다 몇 배 느립니다. release 는 측정 도구를 붙일 수 없습니다. profile 은 release 와 같은 AOT 컴파일에 관측만 열어 둔 모드라 이 용도의 표준입니다.

1.2 이 수치의 한계 — 먼저 밝힙니다

측정 기기에 GPU 가 없습니다. 에뮬레이터가 swiftshader 로 화면을 CPU 에서 그립니다. 그래서 두 수치를 나눠 읽어야 합니다.

구간	무엇을 재는가	이 환경에서 믿을 수 있는가
UI 스레드 (build)	위젯 트리 구성 · 레이아웃 · 표시 목록 생성 — **우리 코드**	🟢 실기기와 자릿수가 같습니다
래스터 스레드 (raster)	표시 목록을 픽셀로 굽는 일 — **GPU 담당**	❌ 실기기보다 훨씬 나쁩니다

따라서 래스터화 수치를 합격·불합격 판정에 쓰지 않습니다. 같은 환경에서 잰 개선 전후의 **비교**로만 씁니다. 실기기 판정은 기준 단말이 정해진 뒤 다시 잽니다(§5.1).

UI 스레드 수치는 그대로 유효합니다. 우리가 쓴 코드가 한 프레임에 얼마를 쓰는지를 재며, 이 값이 예산을 넘으면 어떤 기기에서도 60fps 가 나오지 않습니다.

✦ 2. 앱 시작 시간

`flutter run --profile --trace-startup` 3회.

구간	중앙값	범위	무엇인가
엔진 진입 → 프레임워크 초기화	**134ms**	128~136	Dart VM 과 Flutter 프레임워크가 서는 시간
엔진 진입 → 첫 프레임	**586ms**	582~589	사용자가 첫 화면을 보기까지
엔진 진입 → 첫 프레임 래스터화 완료	**609ms**	602~614	화면에 실제로 픽셀이 찍히기까지

변동 폭이 ±4ms 로 매우 좁습니다. 측정이 안정적이라는 뜻이며, 앞으로 이 값이 눈에 띄게 움직이면 그것은 잡음이 아니라 회귀입니다. **회차마다 같은 방법으로 재어 추이를 보겠습니다.**

첫 프레임까지의 586ms 중 452ms 가 프레임워크 초기화 이후 구간입니다. 이 구간에는 Firebase 초기화 시도와 라우터 구성이 들어갑니다. 현재는 설정 파일이 없어 초기화가 실패로 떨어지는데, 설정 파일이 투입되면 실제 네트워크 왕복이 붙어 이 값이 늘어날 수 있습니다. **설정 파일 투입 직후 재측정 대상입니다.**

✦ 3. 갤럭시 맵 프레임 타이밍

별첨 1 ①의 「맵 드래그 회전 체감 지연」이 이 측정의 대상입니다.

3.1 측정 방법

궤도 애니메이션이 도는 동안 드래그를 섞어 6회 반복하고, 그 사이의 모든 프레임을 잽니다. **애니메이션과 제스처가 겹치는 구간이 가장 무겁기 때문에** 둘을 나눠 재지 않았습니다. 나눠 재면 실제로 끊기는 순간을 놓칩니다.

한 프레임의 예산은 60fps 기준 **16.7ms** 입니다.

3.2 결과 — 개선 후

프레임 4회 측정, 회차당 약 141프레임.

지표	중앙값	범위	예산 대비
UI 스레드 평균	**0.86ms**	0.68~1.12	**5%**
UI 스레드 p90	**0.93ms**	0.76~1.04	6%
UI 스레드 최악	24.42ms	11.54~43.60	회차당 0~1프레임
래스터 평균	14.65ms	11.32~18.09	88%
래스터 p90	26.83ms	22.84~30.41	161%

UI 스레드는 예산의 5%만 씁니다. 우리 코드가 프레임을 막고 있지 않다는 뜻입니다. 남은 여유는 5주차 이후 실시간 데이터가 붙을 때 쓸 몫입니다.

UI 스레드 최악값이 24ms 로 튀는 지점은 첫 프레임과 드래그 시작 프레임입니다. 회차당 0~1회이고 연속되지 않아 체감 끊김으로 이어지지 않습니다.

3.3 개선 전후 — 배경 층 분리

측정 중에 문제를 하나 찾았습니다. **별밭 220개와 궤도가 애니메이션과 같은 층에 있어 궤도가 도는 매 프레임 전부 다시 그려지고 있었습니다.** 별 좌표와 궤도 반지름은 순위가 바뀌지 않는 한 그대로인데도 그랬습니다.

배경을 `RepaintBoundary` 로 감싼 별도 층으로 갈라 한 번만 굽게 했습니다.

지표	개선 전 (3회)	개선 후 (4회)	변화
UI 스레드 평균	1.08ms	**0.86ms**	**−20%**
UI 스레드 p90	1.00ms	**0.93ms**	−7%
UI 스레드 최악	43.60ms	**24.42ms**	**−44%**
UI 스레드 예산 초과	1회	**0.5회**	감소
래스터 평균	18.34ms	**14.65ms**	**−20%**
래스터 p90	31.19ms	**26.83ms**	−14%
래스터 예산 초과	**95회**	**59회**	**−38%**

래스터 최악값만 나빠 보입니다 — 40.00ms → 50.57ms. 개선 후 4회 중 한 회차에 138ms 이상치가 하나 있었고 그것이 중앙값을 끌어올렸습니다. 나머지 3회는 38.67~47.43ms 로 개선 전 범위(34.11~47.43ms)와 겹칩니다. **이 한 지표는 개선을 주장할 근거가 되지 못하므로 그대로 적어 둡니다.**

그 외 여섯 지표는 전부 같은 방향으로 움직였습니다.

3.4 표시 목록 생성 비용 — 마이크로 측정

기기 없이도 회귀를 잡기 위해 `CustomPainter.paint` 만 따로 60회 재고 중앙값을 봅니다. 이 측정은 `flutter test` 안에서 돌아 **매 커밋 자동 실행**됩니다.

대상	중앙값	예산(8.35ms) 대비
갤럭시 맵 · 행성 12개	**70μs**	0.8%
갤럭시 맵 · 행성 24개	**106μs**	1.3%
FPI 육각형 그래프	**113μs**	1.4%

배경 층을 가르기 전 갤럭시 맵은 **132μs** 였습니다. 별밭이 빠져 70μs 로 내려갔습니다.

행성을 12개에서 24개로 배로 늘려도 106μs 입니다. 행성 수가 늘어도 그리기 비용이 문제가 되지 않는다는 뜻이며, 팬덤이 늘었을 때 한 화면에 몇 개까지 띄울지는 성능이 아니라 **가독성**으로 정하면 됩니다.

✦ 4. 앱 크기

4.1 산출물

산출물	크기
`app-release.apk` (ABI 3종 통합)	59.0MB
`app-release.aab` (스토어 제출본)	57.2MB
`Runner.app` (iOS · 서명 전)	20.7MB

4.2 실제 내려받는 크기

59MB 는 사용자가 받는 크기가 아닙니다. 통합 APK 에는 CPU 아키텍처 3종의 네이티브 라이브러리가 전부 들어 있고, 스토어는 기기에 맞는 하나만 내려보냅니다.

아키텍처	크기	대상
arm64-v8a	**23.2MB**	**현행 기기 대부분**
armeabi-v7a	20.7MB	구형 32비트 기기
x86_64	24.8MB	에뮬레이터

실사용 기기가 받는 크기는 23.2MB 입니다.

4.3 구성

구성 요소	크기	비고
Flutter 엔진 (`libflutter.so` · arm64)	11.7MB	프레임워크 런타임
앱 코드 (`libapp.so` · arm64)	6.4MB	우리 Dart 코드 + 플러그인
`classes.dex`	3.8MB	Android 네이티브 층
에셋	0.2MB	아이콘. **폰트 원본 미투입**

아이콘 폰트를 99.8% 줄였습니다 — 1,645KB 에서 4KB 로. 빌드가 실제로 쓰는 아이콘만 남기는 tree-shaking 이 동작하고 있습니다.

폰트 원본이 투입되면 크기가 늘어납니다. Gmarket Sans 3종과 Pretendard Variable, JetBrains Mono 를 합쳐 **약 3~5MB** 를 예상합니다. 투입 후 재측정합니다.

✦ 5. 남은 것

5.1 기준 단말 — 정정합니다

앞서 「기준 단말이 어느 문서에도 정의되어 있지 않다」고 적었습니다. 사실이 아닙니다. 견적서 v2.1에 이미 있습니다.

> 기준 단말(iPhone 11·Galaxy S21 동급 이상) 성능 기준·성능 테스트 리포트

계약서에 있는 것을 못 찾고 발주사께 여쭈려 한 저희 잘못입니다. 제출 전에 발견해 고쳤습니다.

지금 수치는 GPU 가 없는 에뮬레이터 값이라 그대로 판정에 쓸 수 없습니다. **두 기기를 확보해 다시 재고**, 아래 기준으로 합격·불합격을 가릅니다.

지표	판정선	사유
UI 스레드 p90	≤ 8ms	예산의 절반. 나머지는 래스터화 몫
래스터 p90	≤ 12ms	합쳐 16.7ms 안에 들어야 60fps
예산 초과 프레임	≤ 5%	1초에 3프레임까지는 체감되지 않습니다
앱 시작 첫 프레임	≤ 2초	스토어 심사·이탈률 통상 기준

저사양 열화 판정의 하한도 같은 기기입니다. 그보다 낮은 기기는 최소 지원 OS 범위 안이라도 성능 판정 대상이 아닙니다.

5.2 아직 재지 못한 것

항목	왜 못 쟀는가	언제
실기기 프레임·시작 시간	**기기 미확보** — 기준은 견적서가 정했습니다	기기 확보 후 즉시
iOS 프레임 타이밍	이 장비에 Xcode 미설치	재설치 후
저사양 단말 이펙트 축소	기기 미확보	기기 확보 후
실 데이터 위 목록 스크롤	Firestore 연동이 표본 데이터	5주차
동시 접속 부하	서버 배포 전	9주차
배터리·메모리	장시간 실기기 측정 필요	10주차

✦ 6. 재현 방법

측정을 그대로 다시 하실 수 있게 명령을 남깁니다.

측정	명령
그리기 비용	`flutter test test/perf_paint_test.dart`
프레임 타이밍	`flutter drive --driver=test_driver/perf_driver.dart --target=integration_test/galaxy_perf_test.dart --profile --no-dds`
시작 시간	`flutter run --profile --trace-startup`
앱 크기	`flutter build apk --release --split-per-abi`

프레임 타이밍의 원본 JSON 은 `build/perf/` 에 남습니다. 요약값만이 아니라 원본을 남기는 이유는, 이 리포트의 수치가 어디서 나왔는지 되짚을 수 있어야 하기 때문입니다.

✦ 7. 다음 회차

시점	무엇을
기준 단말 확보 직후	실기기 전 항목 재측정 · 판정선 적용
Firebase 설정 투입 직후	시작 시간 재측정 (초기화에 네트워크가 붙습니다)
2026-10-16	4차 QA 회차와 함께 전 항목 재측정
준공 전	최종본 제출