

afreecaTV

모바일 QA 자동화 프로젝트

아프리카TV 모바일 QA 자동화 프로젝트 구축 리포트

모바일랩 QA팀



INDEX

아프리카TV 모바일 QA 자동화

🔥 지금 우리의 문제

🎯 QA 자동화 필요성

💻 자동화 프로젝트 소개/과정

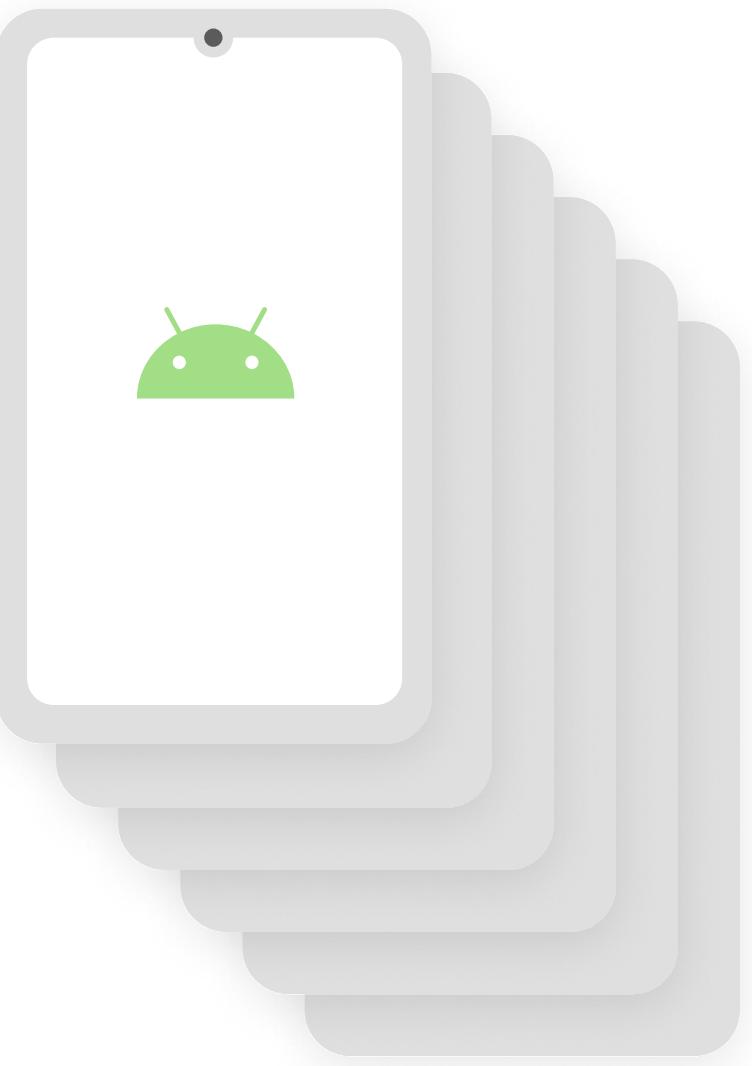
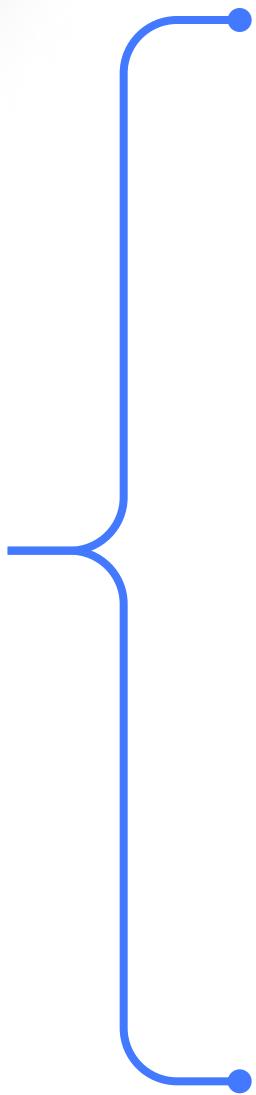
🔮 앞으로 QA 자동화는



반복적인 테스트 영역

OS 호환성 검증을 위해 12개 버전에서
반복된 기능 테스트를 진행합니다.

호환성 검증을 위해 12개 버전에서 반복작업을?



Android 6.x ~ 13.x

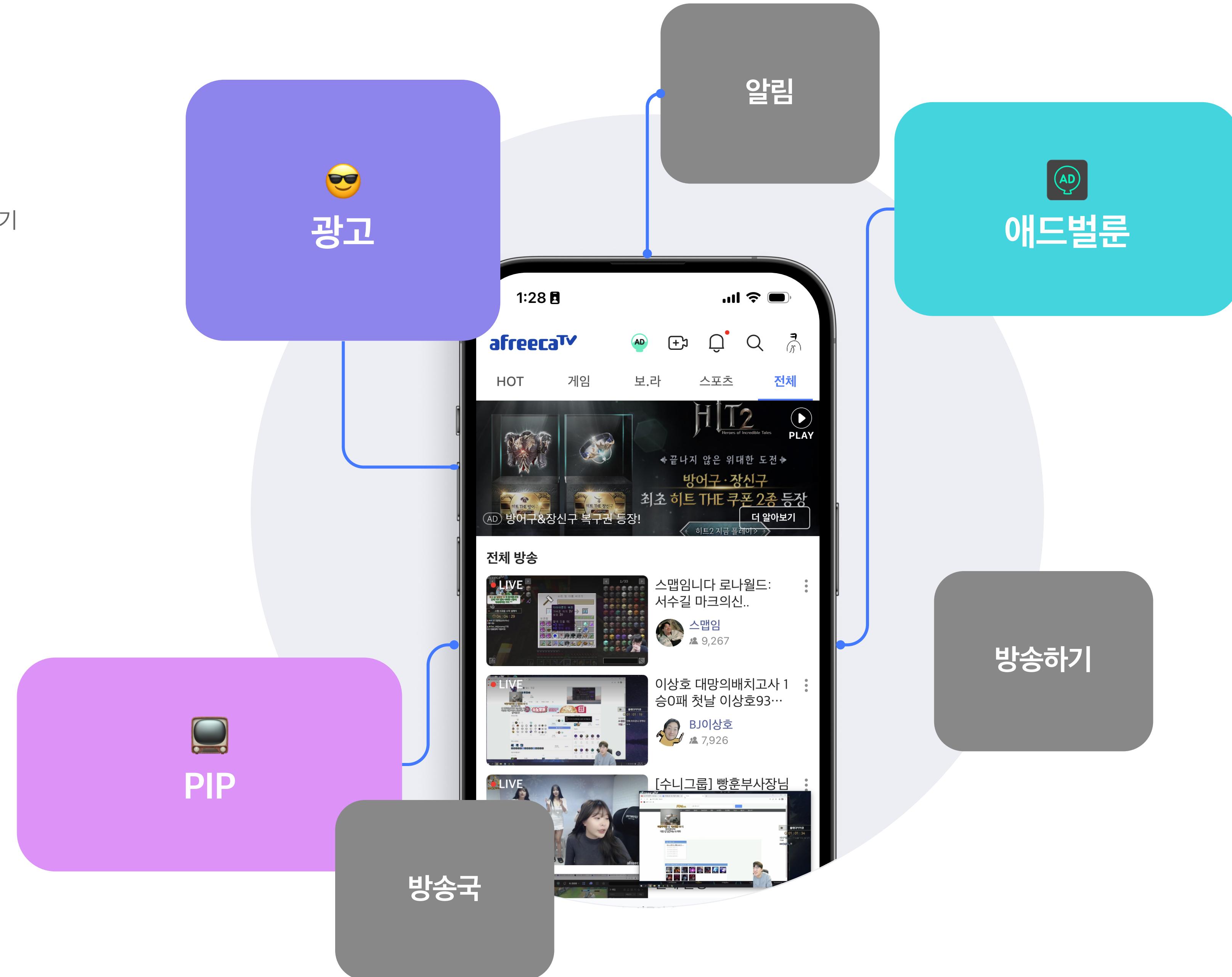


iOS 13.x ~ 16.x

⭐ 지금 우리의 문제

많은 기능

제한된 인원으로 현실적으로 매일 체크하기
어려운 기능 다수 존재합니다.

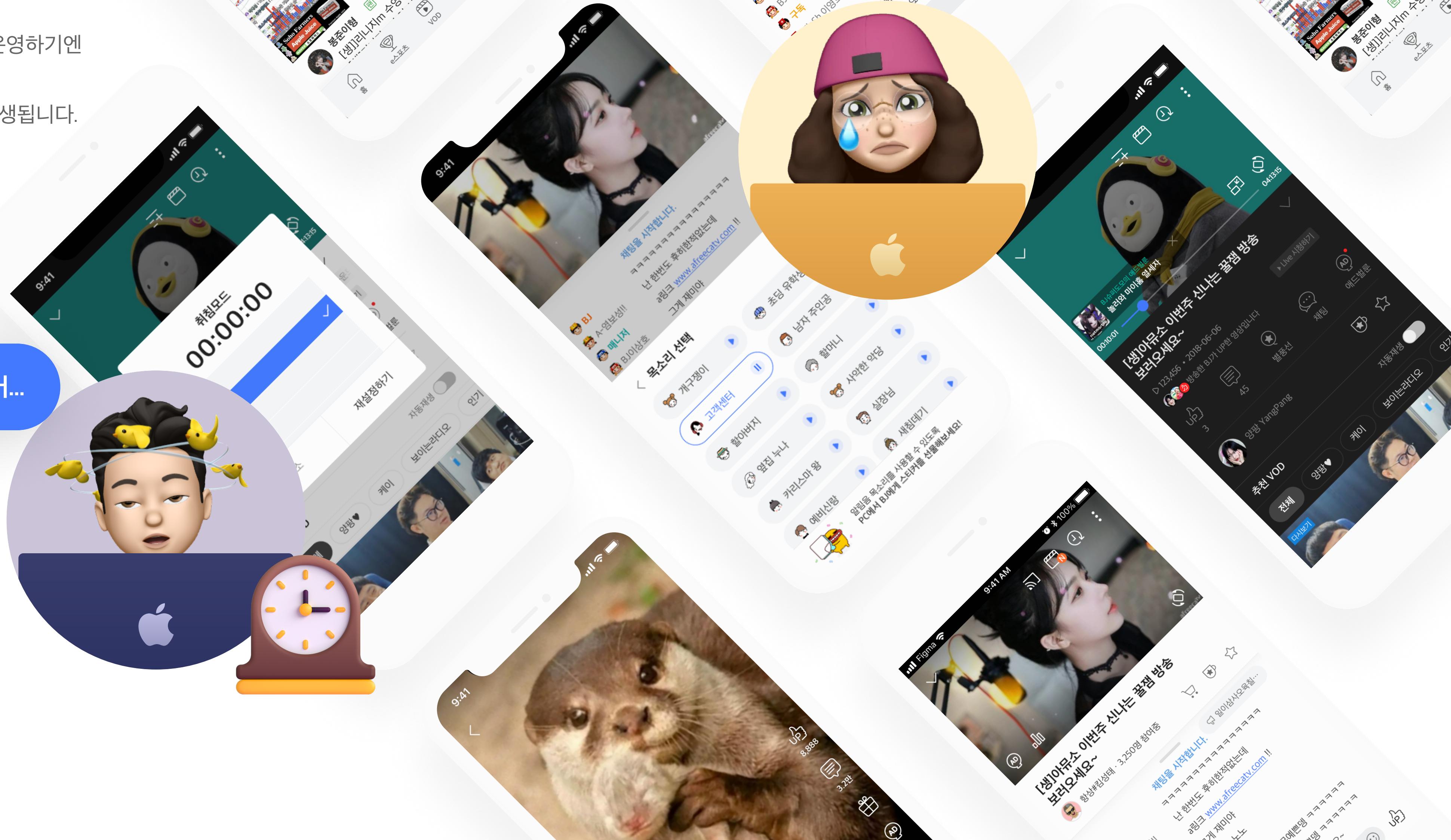


⭐ 지금 우리의 문제

상시 모니터링의 어려움

품질 모니터링을 주 / 야간 지속적으로 운영하기엔
리소스와 비용 한계가 있습니다.
특히, 야간에는 기능 모니터링 공백이 발생됩니다.

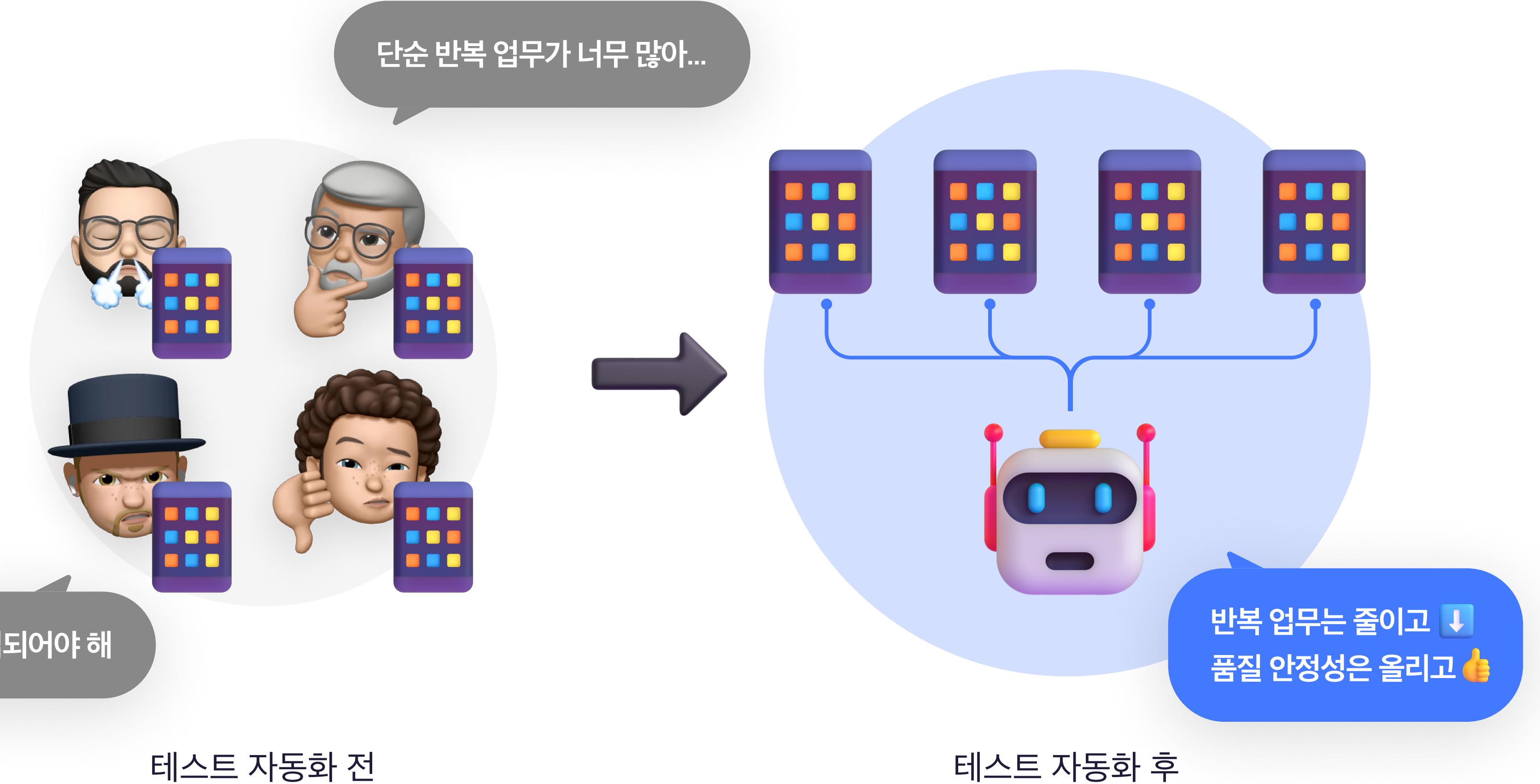
상시 모니터링은 힘들어...



🎯 QA 자동화 필요성

반복 업무는 줄이고
품질 안정성은 올리고
대체 불가한 업무에 집중

QA 업무는 반복적인 업무가 많고
사람이 하는 테스트는 많은 리소스가 투입됩니다.
테스트 자동화를 통해 반복 업무를 줄이고
품질 안정성 확보를 해야합니다.





선택과 집중을 통해 기회비용과 생산성을 얻자!

성공적인 도입을 위해서는 신중한 검토가 필요합니다.



초기 투자

단기적으로 사람이 하는 테스트보다
많은 시간과 비용이 투입됨



효율성

반복적인 업무에 우선 도입하여
효율성을 높여야 함



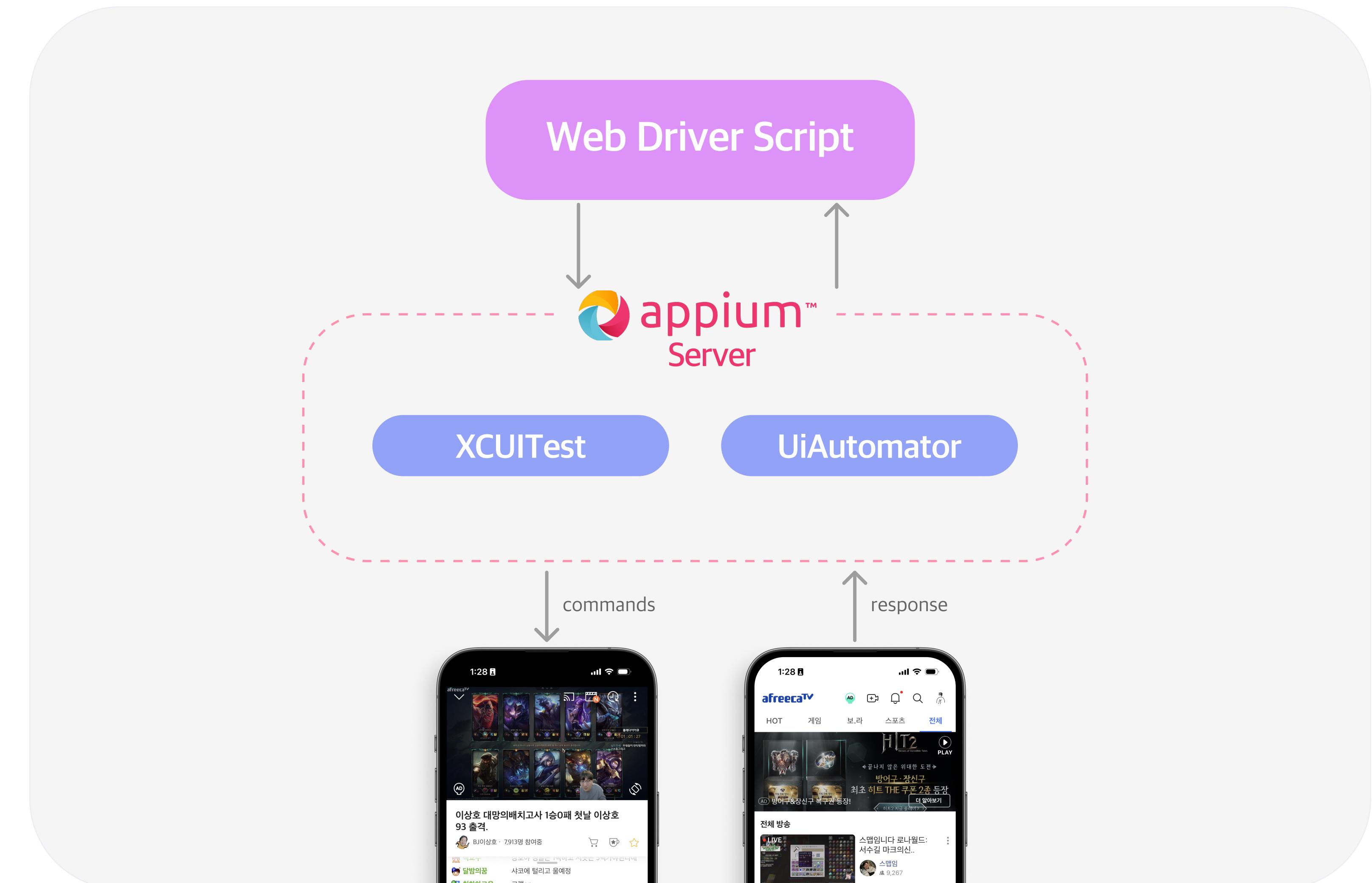
기술력

Test Framework의 활용을 위한
기술력과 테스트 인프라 필요
(개발역량, CI/CD)

■ 자동화 프로젝트 소개 | Appium 개요

자동화의 시작

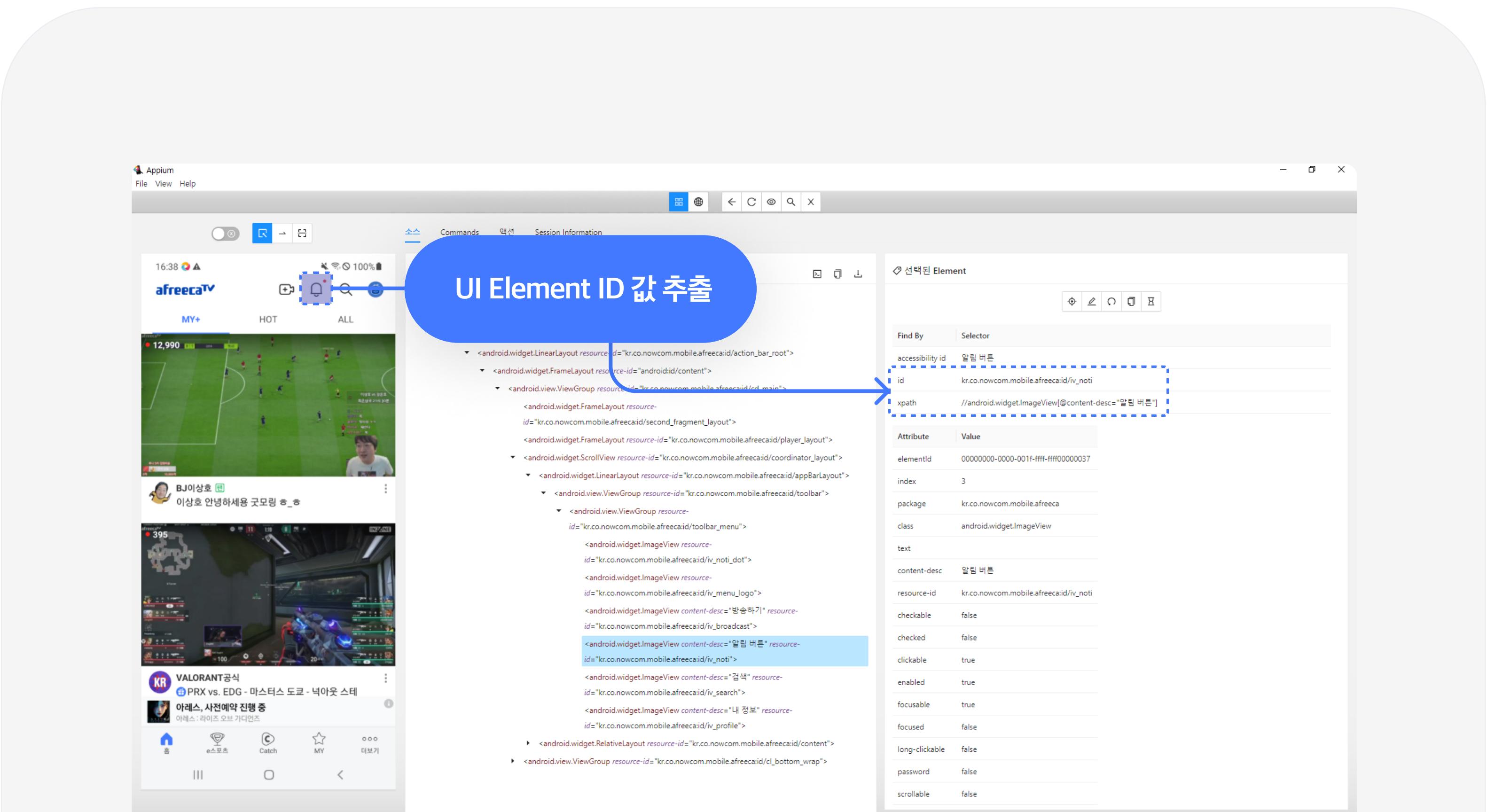
Node.js 기반의 테스트 프레임워크(Appium)를 이용하여 하나의 코드 베이스로 여러 모바일 기기를 제어할 수 있습니다.



■ 자동화 프로젝트 소개 | Appium 개요

동작 원리

테스트에 필요한 UI Element ID 값을 추출하고,
ID 기반으로 디바이스 제어(클릭, 스크롤 등) 합니다.



■ 자동화 프로젝트 소개 | Appium 개요

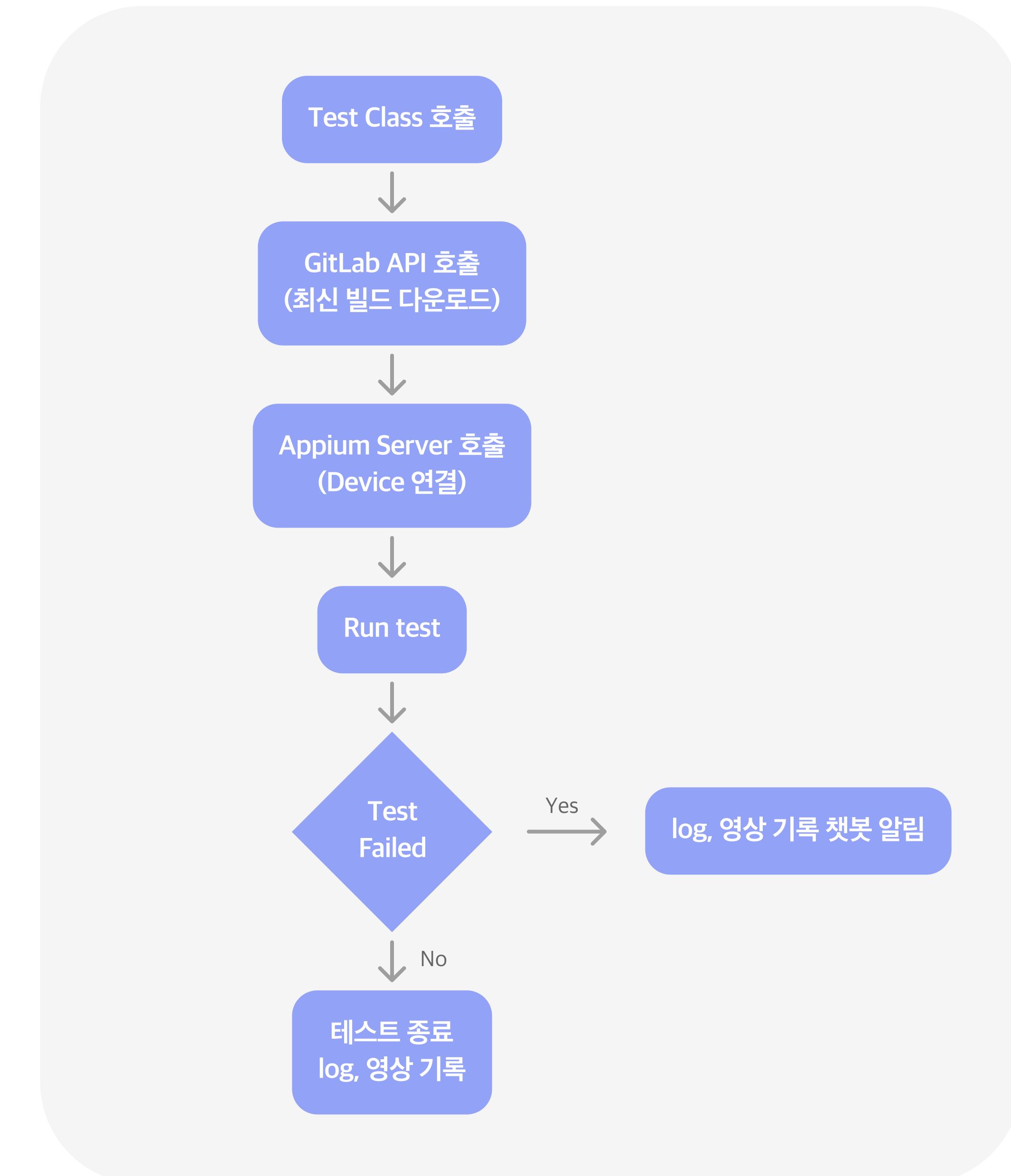
Appium 선택 이유

테스트 자동화 솔루션의 비싼 가격으로 인해
오픈소스 프레임워크 활용하여 직접 구현하였습니다.



프로젝트 구조 개요

테스트 시작 ~ 종료까지의 전체 흐름도입니다.



프로젝트 구조 개요

모든 테스트는 Test<->View 의 1:N 대응 구조로

역할을 분리하여 코드가 복잡해지는 것을 방지합니다.



View

해당 화면에서 수행할 수 있는 동작
ex. 클릭, 스크롤 등



Test

여러 View를 조합하여 테스트 시나리오 정의
ex. 퀴뷰 유저 방송 입장 Test

■ 자동화 프로젝트 소개 | 프로젝트 구조

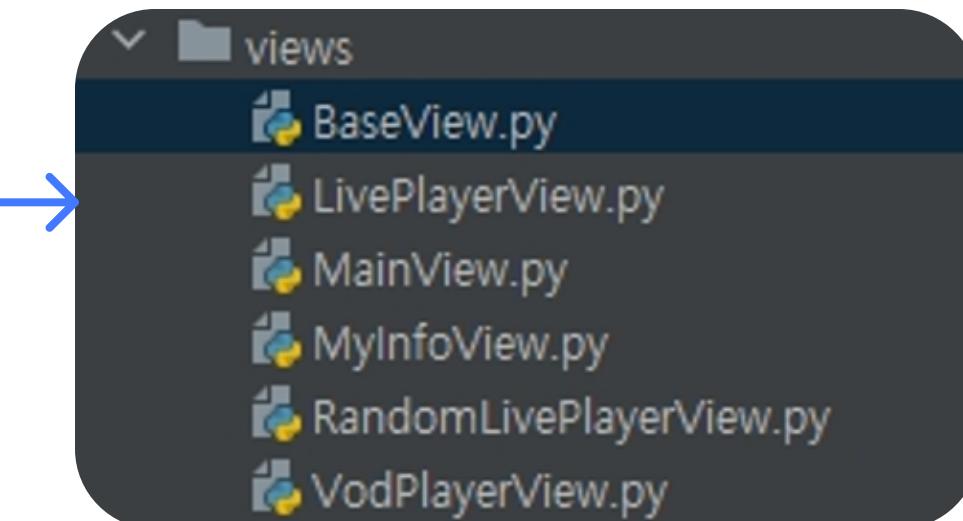
View

테스트를 진행하는 각 화면 단위를 의미하며,

BaseView에서 공통 동작을 정의(클릭, 스크롤 등) 합니다.

BaseView를 상속 받은 각 View에서

필요한 기능은 추가로 구현하여 확장하는 구조입니다.



View



MainView



BaseView



MyInfoView



VodPlayerView



LivePlayerView

```
class MainView(BaseView):
    def __init__(self, driver):
        super().__init__(driver)

    def login_by_afreeca(self, login_id, login_password):
        self.driver.get_screenshot_as_base64()
        assert self.click('login_btn_ID', self.locator_find_timeout)
        self.find_input('afreeca_id_text_field_ID', self.locator_find_timeout)
        assert self.click('afreeca_password_text_field_ID', self.locator_find_timeout)
        self.find_input('afreeca_password_text_field_ID', self.locator_find_timeout)
        assert self.click('confirm_afreeca_login_btn_ID', self.locator_find_timeout)
```

MainView.py

- 메인화면에서 가능한 동작 정의

```
class LivePlayerView(BaseView):
    def __init__(self, driver):
        super().__init__(driver)

    def up_broad(self):
        assert self.click('up_btn_ID', self.locator_find_timeout)
        assert self.true_checker('up_completion_btn_XPATH', self.locator_find_timeout)

    def send_starballon(self):
        assert self.click('starballon_btn_ID', self.locator_find_timeout)
        assert self.click('gift_edit_text_ID', self.locator_find_timeout)
        self.find_input('gift_edit_text_ID', self.locator_find_timeout, '1')
        assert self.click('starballon_send_btn_ID', self.locator_find_timeout)
```

LivePlayerView.py

- Live Player 안에서 가능한 동작 정의

■ 자동화 프로젝트 소개 | 프로젝트 구조

Test

테스트 시나리오를 정의하고, 결과를 기록하는 영역이며
시나리오에 필요한 View를 여러개 호출합니다.

View와 마찬가지로 공통적으로 필요한 부분은
BaseTest에 정의하고, 확장하는 구조입니다.

```
class TestLiveBasic(BaseTest):

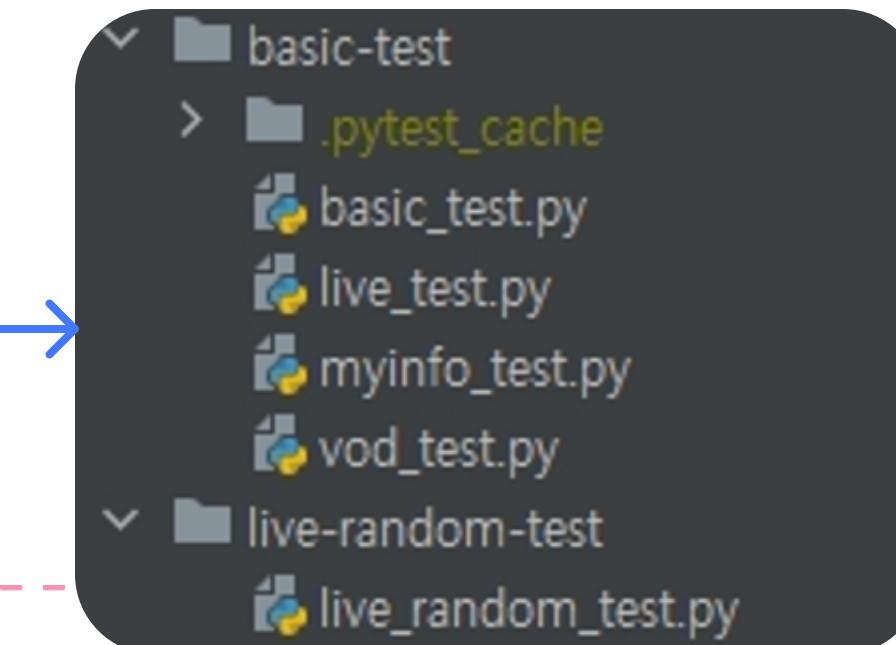
    def test_login(self, appium_driver):
        main_view = MainView(appium_driver)
        test_account = self.get_test_account(test_env='pipeline')
        main_view.login_by_afreeca(login_id=test_account.id, login_password=test_account.password)

    def test_search_broad(self, appium_driver):
        main_view = MainView(appium_driver)
        main_view.search_password_broad(broad_user_id='afueo021')

    def test_up(self, appium_driver):
        live_player_view = LivePlayerView(appium_driver)
        live_player_view.up_broad()
```

LiveTest

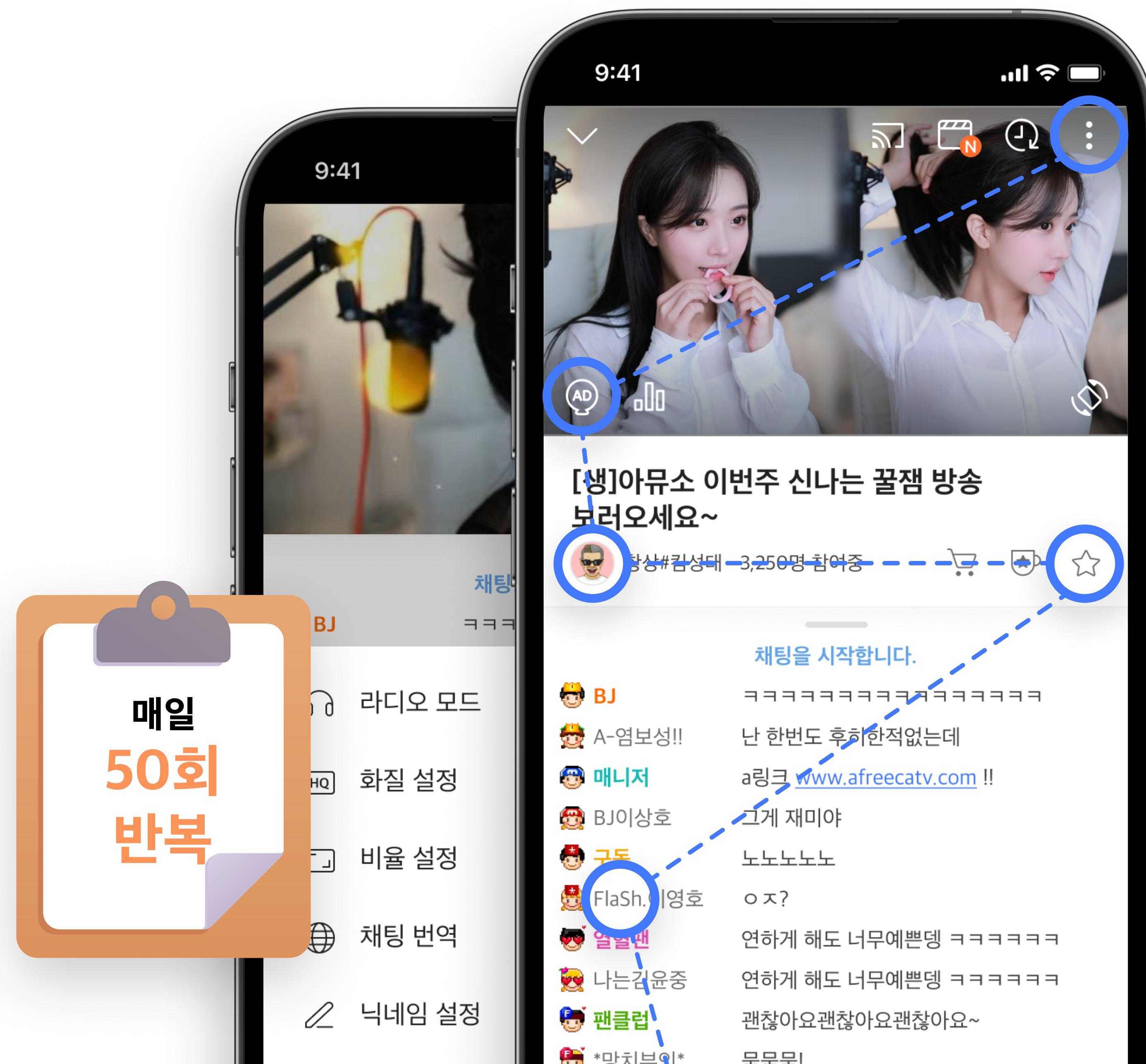
- 테스트 필요한 View 호출
- 함수 단위로 테스트 시나리오 정의



✓ 자동화 주요 과정 | Android 참여 테스트 자동화

방송 참여 기본 기능 테스트

유저 사용빈도가 가장 높은
방송 참여 영역 먼저 테스트 자동화 시행 중 입니다.
배치스크립트로 매일 50회 이상 반복 검증 진행되고 있습니다.



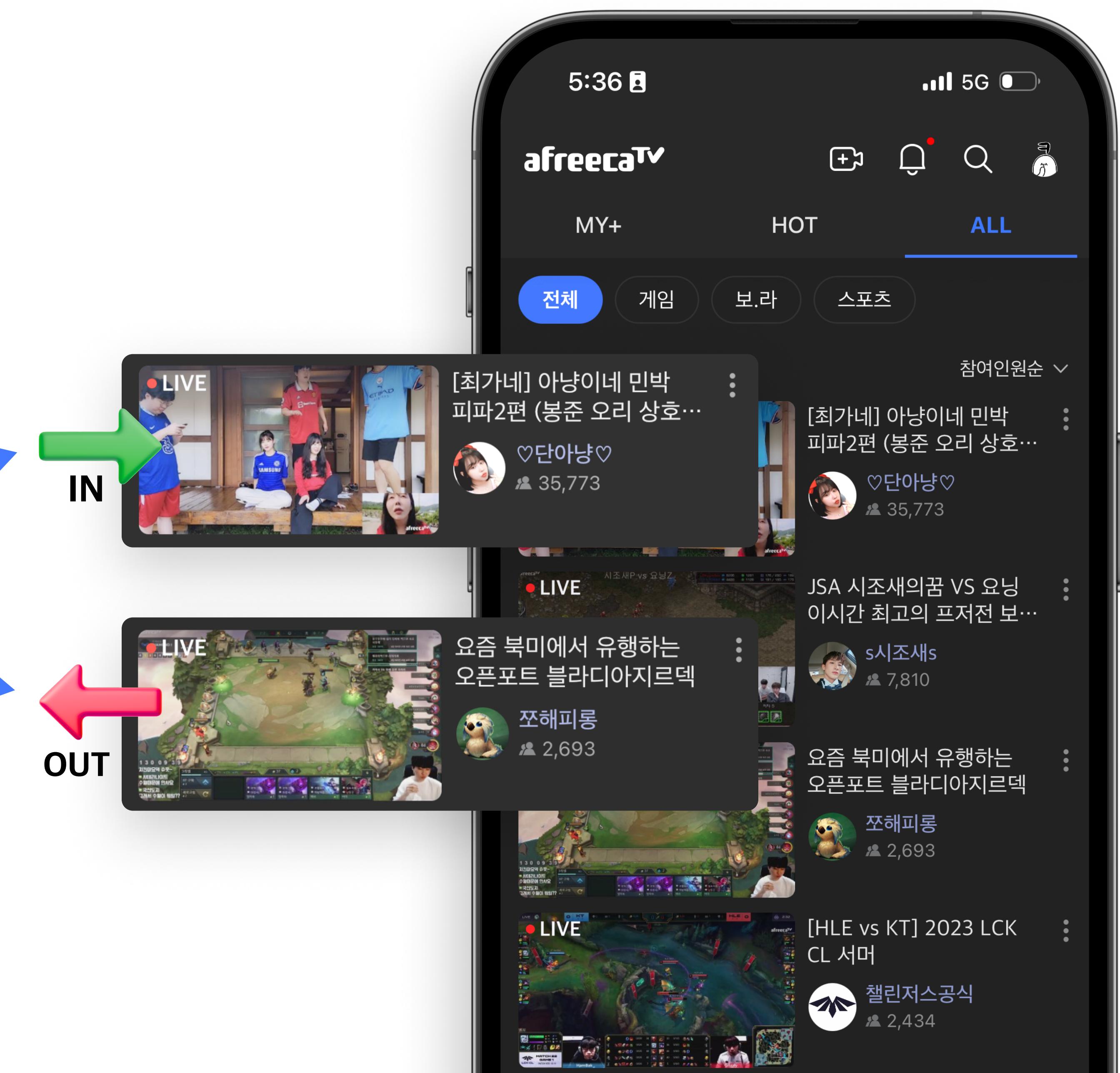
✓ 자동화 주요 과정 | Android 참여 테스트 자동화

방송 입/퇴장 반복 테스트

방송리스트 중 랜덤하게 방송에 입장하여

미디어, 채팅서버 연결 반복 검증하고

방송 참여 모니터링 용도로 활용하고 있습니다.



 자동화 주요 과정 | 도입 성과

일정한 테스트 품질 확보

사람이 테스트할 경우 역량 차이가 발생합니다.

신입급 팀원이 테스트하는 것보다 빠른 속도로

검증 가능하며 일정 수준 이상의 품질을 유지할 수 있습니다.

- 자동화 : 약 6분
- 신입급 QA : 약 10분

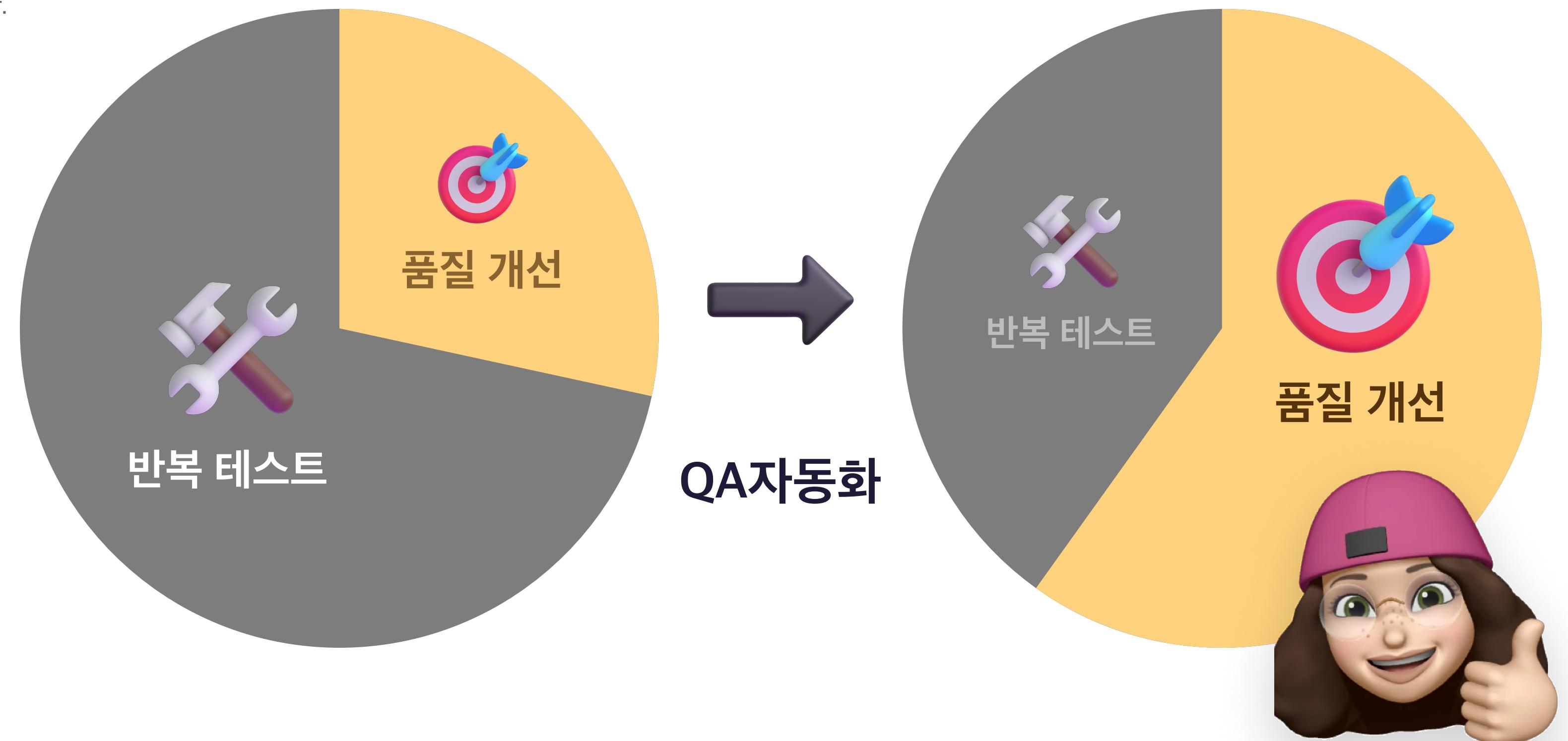


✓ 자동화 주요 과정 | 도입 성과

리소스 확보

자동화 테스트로 확보한 업무 시간을
난이도가 높거나 본질적인 품질 개선 업무에 활용합니다.

- 기획 / 개발된 프로젝트에 정책적 결함은 없는지
- 유저가 실사용했을 때 UI / UX가 적절한지
- 개선해야 할 기본적인 품질 이슈는 없는지

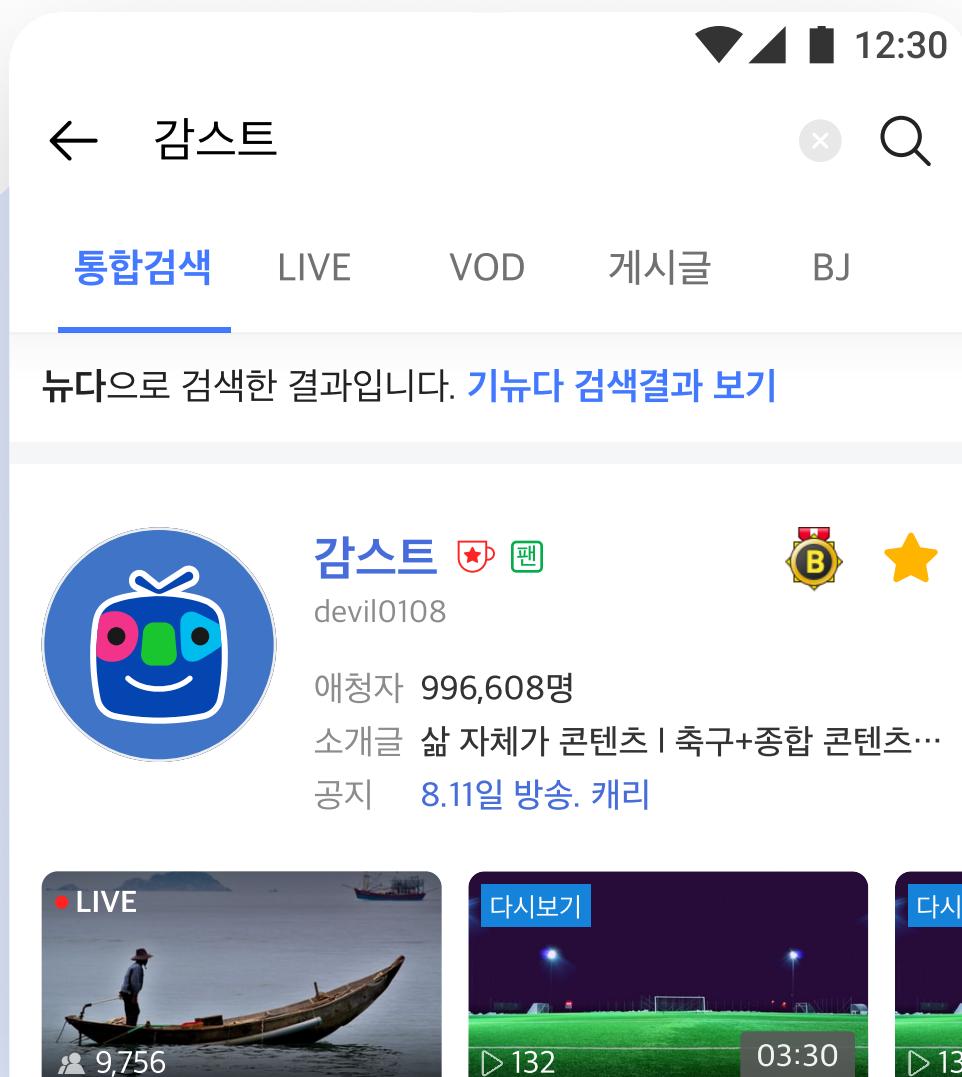


자동화 주요 과정 | 도입 성과

자동화를 통한 주요 사례들

기본 자동화 시나리오로 3개의 버그를 발견하였고

시나리오 추가 시 더 많은 영역에서 안정성 확보할 수 있을 것으로 예상됩니다.



검색 결과 오노출 이슈

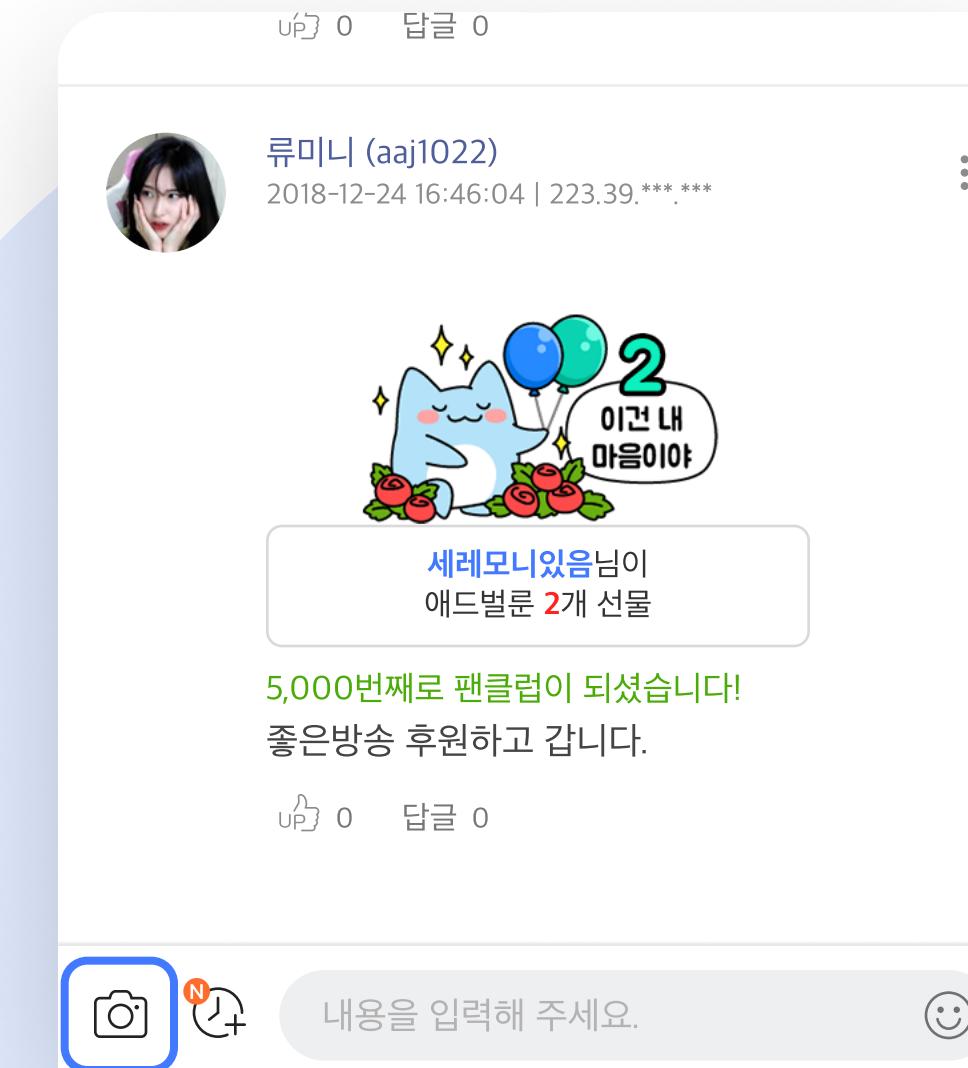
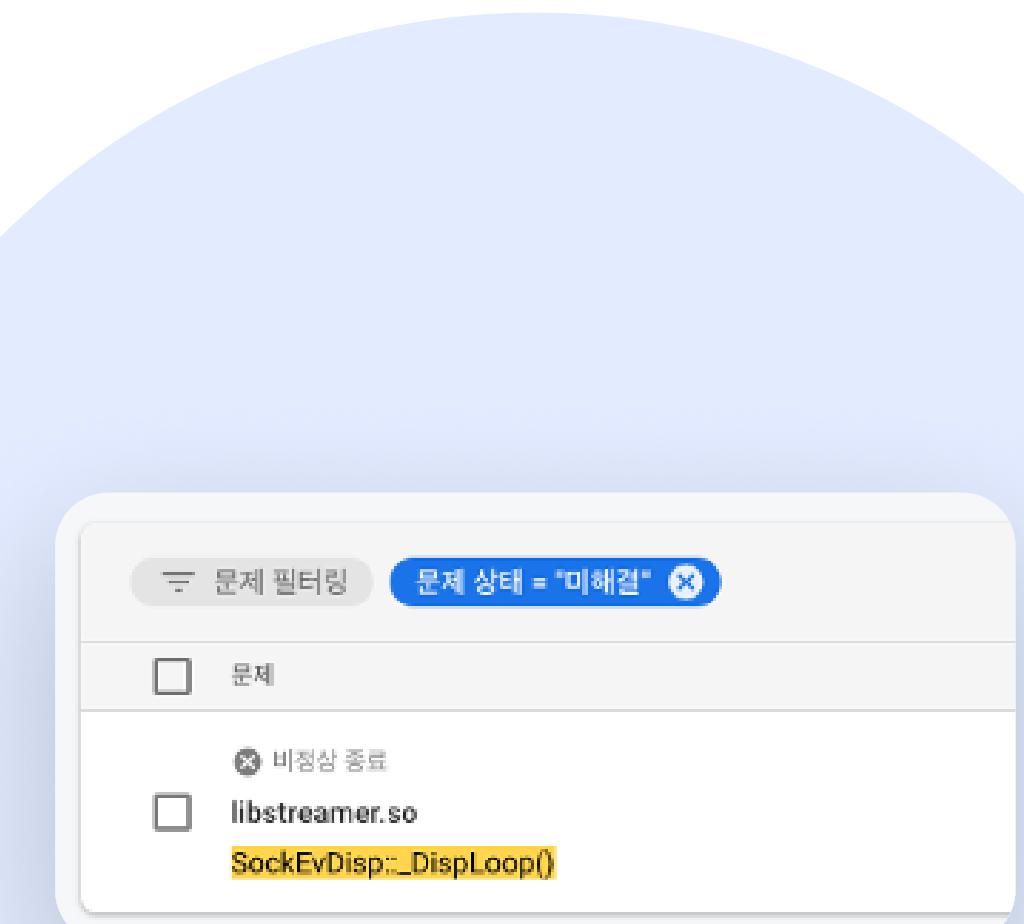


사진 첨부 버튼 미동작 이슈



간헐적 크래시 확인

자동화 주요 과정 | 도입 성과

다양한 활용 가능

기능적인 테스트 뿐만 아니라 시간 투자비용이 큰
간헐적 이슈 재현 용도로 활용이 가능합니다.
또한 앱 안정성을 확인하는 용도로 확대 가능합니다.



감사합니다

