

QA Automation · Quality Engineering

조시우

1992년생 · 전체 QA 7년 이상

한국공학대학교 졸업 · 4년제 학사

사용자 경험을 끝까지 확인하는 QA

화면의 동작, 그 뒤의 데이터, 배포 이후의 운영까지. 문제를 정의하고 검증 방법을 설계해 이슈를 닫을 근거까지 만들었습니다.



Mirtec / 미르기술

2023.05 ~ 재직중

3년 5개월

브랜드

2021.06 ~ 2023.05

2년

나인스쿨

2019.04 ~ 2021.05

2년 2개월

이 포트폴리오에서 보여 드릴 세 가지

01 · 커머스 Data QA

브랜드에서 주문·결제, 자체 페이지, 통합회원, 서울스토어 내재화 TFT를 경험했습니다. SDK 이벤트를 직접 확인하며 데이터 정합성도 검증했습니다.

02 · 믿을 수 있는 회귀 자동화

UI·출력물·DB를 교차검증하고 자동화의 완료 판정 자체가 틀린 경우도 다시 설계했습니다.

03 · QA Operations

TC·증거·Jira·변경 이력을 연결하고 실제 실행 범위와 배포 후 독자 화면까지 확인했습니다.

설계부터 이슈 종료까지 문제 정의 → 접근과 시행착오 → 해결·재검증

테스트를 수행하는 데서 출발해 팀이 품질을 판단할 수 있는 근거를 만드는 일까지 맡아 왔습니다.

가입을 마친 사용자가 첫 주문까지 편하게 가도록

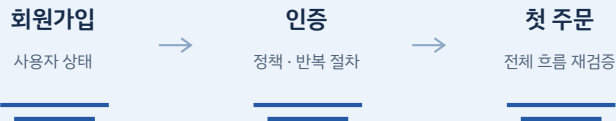
회원가입·인증·주문을 하나의 사용자 흐름으로 연결해 검토했습니다. 통합회원 개선에서는 기능의 정상 동작과 첫 주문까지의 불편을 함께 다뤘습니다.

회원·인증·주문 연결

정책·반복 절차 검토

변경 영향·예외 협의

개선 흐름 재검증



기능의 통과에서 첫 주문까지

회원·인증 상태가 주문으로 이어지는 경계를 검토했습니다. 당시 TC 원본이 아닌 검증 관점의 재구성입니다.
설명용 도식. 도형 개수는 측정값을 나타내지 않습니다.

38.6% 감소

첫 주문까지 소요시간

회원 → 첫 주문

여정 전체를 검토

설계·재검증

내가 담당할 QA 범위

문제 정의 → 접근

기능의 성공과 여정의 불편을 함께 봤습니다

가입·인증·주문이 각각 정상이어도 사용자가 전체 과정을 지나며 겪는 불편은 남을 수 있습니다. 첫 주문까지의 소요시간을 품질 관점에서 검토했습니다.

기능 사이의 상태 전달을 확인했습니다

가입 여부와 인증 상태가 주문으로 전달되는 경계를 중심으로 변경 영향과 예외 조건을 정리했습니다.
기획·개발과 개선 범위를 맞췄습니다.

절차를 줄이는 판단에도 조건이 필요했습니다

반복 인증과 불필요한 절차를 검토하되 필요한 인증·회원 정책을 지키는 조건을 함께 놓고 판단했습니다.
단계 수만으로 개선을 판단하지 않았습니다.

판단 → 재검증 → 배움

정상 흐름과 예외 흐름을 다시 봤습니다

개선 이후에도 가입·인증 상태에 맞게 주문으로 이어지는지 확인했습니다. 사용자 상태별 기대 결과와 실패 후의 다음 행동을 함께 검토했습니다.

프로젝트 성과와 QA 기여를 구분했습니다

통합회원 개선 프로젝트의 첫 주문 소요시간은 38.6% 줄었습니다. 제 기여는 요구사항·예외 조건 검토, 변경 영향 정리와 개선 흐름 재검증입니다.

기능 단위의 통과가 경험 전체의 통과는 아닙니다

검증 대상을 개별 화면에서 사용자가 하려던 일까지 넓혔습니다. 회원·주문 정책과 사용자 불편을 함께 보는 기준을 만들었습니다.

내 기여: 회원가입부터 첫 주문까지의 QA 범위와 예외 조건을 정리하고 기획·개발과 개선 범위를 맞춘 뒤 변경된 흐름을 재검증했습니다.

중복을 막는 로직이 이벤트를 전부 지웠습니다

앱이 GA(구글 애널리틱스)로 직접 보내던 이벤트를 내부 릴레이 서버를 거쳐 보내도록 변경했습니다. 적은 횟수의 중복 검사는 통과했지만 동시에 몰린 이벤트가 전부 사라지는 문제를 찾았습니다.

휴대폰에서 10회 반복

mitmproxy로 포착

동시에 재전송

발행 수·순서 바뀌 재검증



1~3회

적은 중복 횟수는 정상

50ms 이내

집중 발행 시 전량 누락

누락 미관찰

수정 후 조건별 재검증

문제 → 재현 → 원인 구간 특정

변경된 구간부터 의심했습니다

앱 → GA 직접 전송에서는 없던 순서 뒤바뀜과 누락이 릴레이 서버 도입 뒤 발생했습니다. 중복 제거 후에도 유효 이벤트는 남아야 했습니다.

1~3회 검사에서 끝내지 않았습니다

휴대폰에서 같은 행동을 10회 반복하고 프록시로 이벤트를 잡은 뒤 동시에 재전송했습니다. 실제 앱 이벤트로 동시 발행 조건을 만들었습니다.

짧은 시간의 집중 발행에서 재현했습니다

1~11번 쿼가 50ms 이내에 동시에 발행될 때 이벤트를 전부 버리는 구현을 발견했습니다. 중복 횟수와 동시성이 결함을 드러내는 조건이었습니다.

개발팀 조치 → 재검증 → 종료

재현 조건을 개발팀에 전달했습니다

정상적인 적은 횟수의 중복과 전량 누락 조건을 구분해 공유했습니다. 릴레이 구간의 구현 조치 후 같은 재현 절차로 다시 검사했습니다.

원래 실패 조건의 주변도 넓혔습니다

기존 실패 조건과 함께 2·3·5·20·30회 동시 발행을 확인했습니다. 정상 발송과 이벤트 순서 교환도 포함했습니다.

조건별 재검증에서 누락이 재발하지 않았습니다

검사한 조건에서 누락이 다시 발생하지 않는 것을 확인했습니다. 재시도 성공을 확인한 뒤 발행 수와 순서가 달라도 유효한지 재검증했습니다.

내 기여: 전송 경로 변경의 리스크를 정의하고 실기기·프록시 재현을 설계했습니다. 개발팀에 원인을 공유한 뒤 경계 조건을 확대해 수정 내용을 재검증했습니다. 작은 정상 표본이 놓친 동시성 결함을 찾아 종료 근거를 만들었습니다.

행동이 남아도, 누구의 행동인지 빠졌습니다

서드파티 행동 로그에서 사용자 고유 식별값이 약 10% 누락되는 문제를 추적했습니다. 화면 동작과 실제 전송 데이터, 불필요한 Sync 비용을 함께 검토했습니다.

실제 사용자 행동

SDK·요청 데이터 관찰

식별 정책과 값 대조

누락 보정·Sync 개선



행동과 사용자가 연결되는지 확인

검증 관점의 재구성입니다. 식별값 누락과 릴레이 동시 발행 결함의 수치를 합산하지 않았습니다.
설명용 도식. 도형 개수는 측정값을 나타내지 않습니다.

약 10%

당시 식별값 누락 수준

41% 절감

관련 데이터 Sync 비용

관찰 → 조건 검증

mitmproxy 커스터마이징

문제 → 관찰 지점 확보

화면과 수집 데이터의 경계를 봤습니다

버튼이 정상 동작해도 수집된 행동과 사용자가 연결되지 않으면 분석의 신뢰도가 낮아집니다. 식별값의 누락·변경 조건을 확인할 필요가 있었습니다.

네트워크 계층을 직접 관찰했습니다

mitmproxy를 커스터마이징해 앱과 마케팅 도구 사이에서 실제 요청을 확인하고 조건을 바꾸는 지점을 마련했습니다. 행동 순서와 전송 값을 함께 봤습니다.

제품의 식별 정책과 대조했습니다

사용자 정보가 없거나 바뀌었을 때 어떤 값을 보내야 하는지 기준을 세우고 요청 값과 수집 결과를 대조하며 원인 범위를 좁혔습니다.

해결과 미완료 확장의 구분

식별값 누락을 추적·보정했습니다

약 10% 수준으로 관찰된 식별값 누락 문제를 추적하고 보정했습니다. 이 수치를 수정 후 누락을 0%나 전체 이벤트의 무결함으로 확장하지 않았습니다.

데이터의 필요성과 비용도 함께 봤습니다

불필요한 사용자 데이터 전송을 개선해 관련 마케팅 도구의 Sync 비용을 41% 줄이는 데 참여했습니다. 전체 마케팅 예산이나 서버 비용의 절감률은 아닙니다.

관찰 도구와 운영 자동화는 별개였습니다

이벤트를 이용한 앱 자동화 PoC도 시도했지만 운영 완료로 이어지지는 않았습니다. 반복 실행에는 입력 조건·기대 결과·실패 근거가 함께 필요하다는 점을 배웠습니다.

내 기여: 네트워크 관찰·조건 변경 환경을 만들고 식별값 정합성과 전송 정책을 검토했습니다. 도구 구축과 실제 운영까지 완료한 범위를 구분해 설명합니다.

파일은 만들어졌는데 안의 값이 틀렸습니다

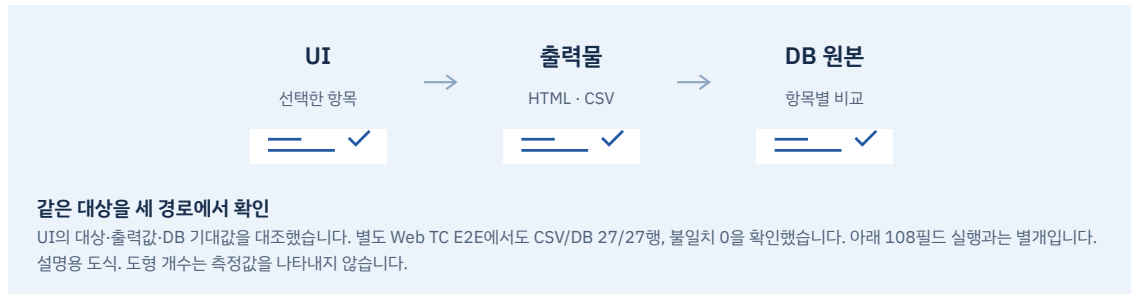
화면에서 선택한 데이터가 출력물에 정확히 담기는지 확인하는 회귀 체계를 설계했습니다. UI 동작, HTML·CSV 출력, 읽기 전용 DB 조회를 같은 검증 흐름으로 연결했습니다.

UI에서 대상·옵션 선택

출력물 생성·읽기

DB 원본과 대조

수정 버전 재검증



108개

선택 필드 비교

13행 + 14행

두 데이터 표본 재검증

불일치 0

수정 반영 후 행·셀 비교

수정 리비전 기록 발췌 `mixedResult.BoardId` 변경 `mixedResult.Pitch.ToString()` 식별값 대신 측정값을 출력

문제 → 시도 → 발견

생성과 정확성을 나눴습니다

파일 생성과 행 수 일치만으로 PASS를 주면 값이 잘못 들어간 오류를 놓칩니다. 데이터 식별자와 선택한 각 필드를 DB와 비교했습니다.

다른 필드가 들어간 원인을 찾았습니다

측정값을 내보내야 하는 분기에 식별값이 들어갔습니다. 화면 옵션에서 출력 코드, DB 원본 필드까지 따라가 원인을 좁혔습니다.

수정 버전을 같은 기준으로 검사했습니다

상위 소스의 수정 리비전을 반영하고 재빌드했습니다. 기존 비교 규칙과 데이터 표본을 유지해 두 표본의 불일치가 0인지 확인했습니다.

시행착오와 종료 근거

시도 1 · 자동화 연결 오류도 발생

오래 사용한 창에서 UI 자동화 연결 오류가 반복됐습니다. 이를 제품의 데이터 결함과 분리했습니다.

보정 · 제한된 재시도와 새 프로세스

화면 제어 대상을 다시 찾도록 하고 재시도 후에도 실패하면 소유한 앱을 정상 종료한 뒤 새 프로세스로 검증했습니다.

종료 · 판정기를 느슨하게 만들지 않음

측정값을 원본 필드와 비교하는 규칙을 유지했습니다. 수정 버전에서 108개 선택 필드의 값과 식별자가 일치하는 것을 확인해 결함 종료 근거를 확보했습니다.

내 기여: 비교 기준과 실행 경로를 설계하고 원인을 추적했습니다. 수정 버전을 반영해 검증하고 회귀 기록까지 남겼습니다. 제품 수정과 자동화 보정을 구분해 관리했습니다.

"5.91초 완료"를 결과에서 버린 이유

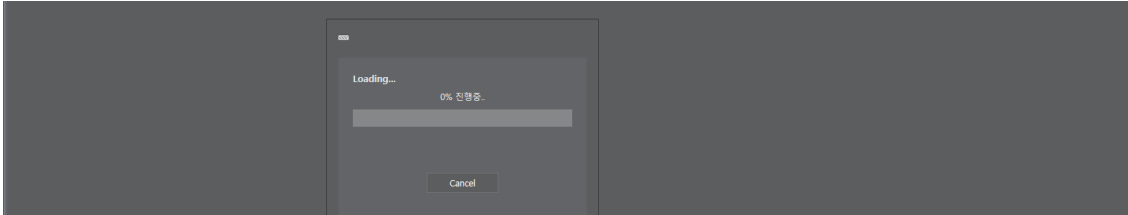
자동화는 완료라고 했지만 캡처에는 여전히 Loading과 0% 진행 상태가 남아 있었습니다. 사용자가 결과를 보고 다시 조작할 수 있는 시점으로 완료 기준을 바꿨습니다.

로딩 표시 사라짐

내용 렌더·내보내기 활성화

프로세스 응답

실제 스크롤·복원



완료 로그도 검증 대상

기록된 성공과 실제 사용자 경험이 일치하는지 확인했습니다.
최종 재실행의 로딩 화면, 5.91초 오판 당시 캡처가 아닙니다.

5.91초

오판으로 폐기한 측정값

10종 · 55조건

최종 실행 범위

55 / 55

완료·응답 조건 검증

트라이앵글 · 측정기를 먼저 검증

실패 1 · 로딩 중인데 완료로 판정

완료 로그를 캡처·화면 구조와 대조하자 로딩 중임이 드러났습니다. 5.91초를 성능 결과에서 제외했습니다.

보정 1 · 관찰과 조작을 함께 검사

렌더된 화면, 활성화된 내보내기, 프로세스 응답에 실제 스크롤과 복원까지 더했습니다.

실패 2 · 자동화 도구가 느려짐

보고서 탭이 쌓이자 옵션 탐색이 모든 열린 창을 반복 검색했습니다. 11조건까지 진행한 진단 실행은 최종 성과에 넣지 않았습니다.

다시 실행한 결과와 남은 판단

보정 2 · 현재 문서와 대화상자로 탐색 제한

선택한 문서의 컨트롤을 한 번 수집하고 별도 옵션 창은 정확한 창으로 한정했습니다. 제품 로딩 시간과 자동화 준비 시간을 분리했습니다.

검증 · 한 프로세스에서 55조건 완주

최종 실행은 10종 보고서의 55조건을 모두 수행했습니다. 데이터가 없는 차트도 화면 틀·라벨·응답 조건으로 판정했습니다.

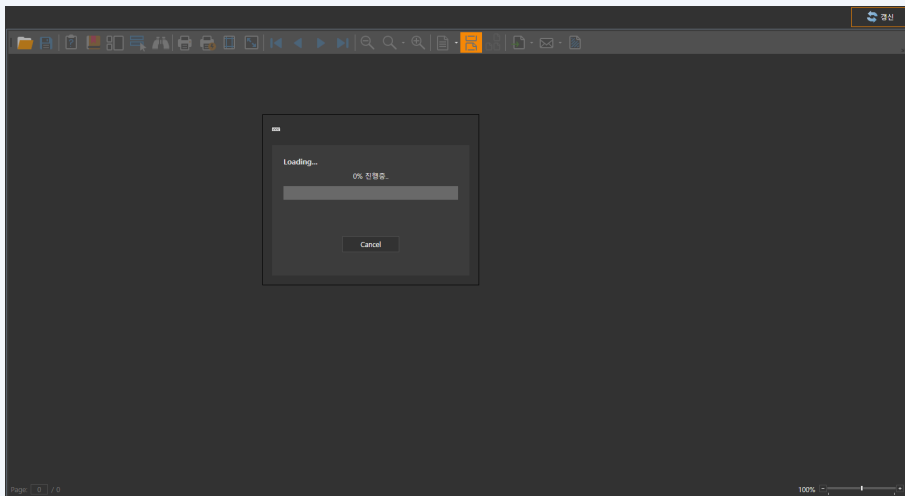
해석 · 완주와 속도 합격은 별개

최장 완료 시간은 70.000초였습니다. 보고서 성능이 개선됐어도 사용성에는 문제가 남을 수 있다고 판단했습니다. 보고서별 대기시간 메트릭을 만들고 기준을 초과한 보고서를 선별했습니다. 개선은 3단계 (Phase)로 나눠 진행하기로 했습니다.

내 기여: 측정 완료 기준, 조건 조합, 제외할 측정값과 재실행 범위를 결정했습니다. 성공 로그를 그대로 믿지 않고 실제 화면과 사용자 조작으로 판정기를 검증했습니다.

완료를 기록하기 전에, 화면을 다시 검증합니다

기존의 완료 로그를 화면과 대조한 뒤 판정 조건을 보강했습니다. 아래는 실제 자동화 코드와 Git 변경의 일부입니다.



실제 재실행 캡처 / 로딩 중

2026.09.03 최종 실행의 로딩 화면입니다.
5.91초 오판 당시 캡처가 아닙니다.

화면에서 출발한 보정

로딩 중인 상태를 성공으로 기록하지 않도록 응답·로딩·내보내기·실제 렌더 상태를 함께 확인합니다.

이 조건을 만족한 뒤에도 스크롤과 복원 후 응답을 검사하고 마지막에 완료 시간을 기록합니다.

화면 상태의 결합 조건

completion.py · L110~116

```
return bool(
    responsive
    and not busy
    and export_enabled
    and refresh_enabled is not False
    and rendered_report_visible(pixels)
)
```

응답·로딩·내보내기·렌더 상태를 함께 확인합니다. 갱신 상태가 False이면 차단하고 None이면 허용하는 원본 구현입니다.

탐색 범위도 다시 좁혔습니다

실제 Git 변경 발췌

aa0114b · uia_options.py

```
- self.control_index = UiaControlIndex(window)
+ self.control_index = UiaControlIndex(window, active_document)
```

이미 선택한 문서를 인덱스에 넘겨 다시 찾는 비용을 줄였습니다. 별도 옵션 창은 해당 창의 핸들로 찾도록 변경했습니다.

5.91초 오판과 11조건 진단 실행을 최종 성과에서 제외하고 보정된 판정기로 10중·55조건을 다시 실행했습니다. 자동화의 성공 메시지도 검증 대상으로 삼았습니다.

빨라진 화면과 여전히 느린 화면을 함께 남겼습니다

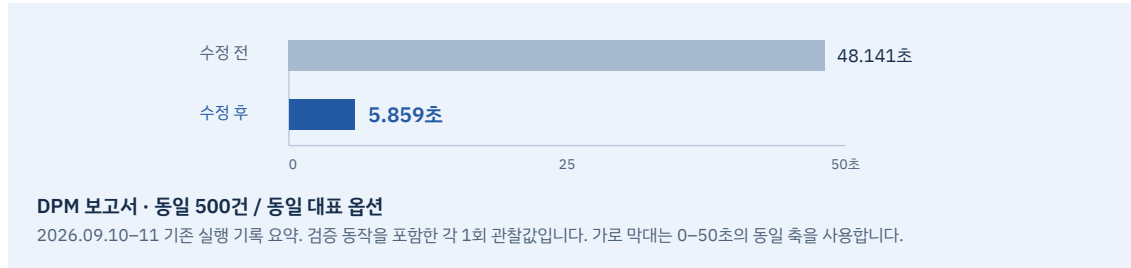
초기 대량 조화가 600초 제한을 넘어 중단된 뒤, 비교 조건부터 다시 맞췄습니다. 동일 500건·같은 대표 옵션으로 개발 수정 전후를 재검증하고 남은 느린 경로를 분리했습니다.

시간 초과·미실행 기록

동일 500건 입력 대조

패치 전후 같은 조건

개선·느린 경로 분리



36개

화면·흐름 분류

30개

화면 시간 측정

6개

조건 제한·별도 흐름

실패 → 비교 조건 복구

600초 중단을 개선 전 시간으로 쓰지 않았습니다

초기 13,405건의 대량 선택 실행은 제한 시간을 넘었습니다. 나머지 60개 시나리오는 미실행으로 남겼고 이후의 500건 실행과 직접 전후 비교하지 않았습니다.

같은 입력과 대표 옵션을 맞췄습니다

입력 식별을 대조한 동일 500건으로 수정 전후를 비교했습니다. DPM 보고서는 48.141초와 5.859초로 관찰했습니다. 각 1회 관찰값을 구분해 남겼습니다.

제품 수정과 QA의 책임을 구분했습니다

성능 패치는 개발 담당자의 수정입니다. 저는 비교 입력, 완료 기준, 중단 조건과 재검증 범위를 정하고 실행 결과를 대조했습니다.

관찰 결과 → 남은 위험

종아진 결과만 고르지 않았습니다

수정 후에도 Yield는 48.766초, NG Export는 337.188초로 관찰됐습니다. 느린 경로를 후속 분석 대상으로 남겨 별도로 관리했습니다.

제외한 6개를 0초나 통과로 바꾸지 않았습니다

건수 제한이 있는 3개와 실시간·파일 등 별도 흐름 3개를 시간 측정과 분리했습니다. 30개 측정을 전체 화면의 데이터 정합성 통과로 집계하지 않았습니다.

반복 측정과 정합성 검사는 후속 과제입니다

관찰 시간에는 검증 동작이 포함됩니다. 반복 실행, 구간별 병목 분석, 데이터 정합성 확인을 별도 과제로 유지했습니다.

내 기여: 동일 입력인지 확인하고 실행 중단·미실행을 처리했습니다. 비교 범위를 설계해 패치를 재검증하고 남은 병목을 보고했습니다. 개선된 화면과 후속 분석이 필요한 화면을 함께 제시했습니다.

문서를 다시 쓰다가 검증의 의미가 달라졌습니다

AI가 새 문서를 만드는 동안 기존 설명·이미지·입력 결과가 약해지거나 빠질 수 있었습니다. 생성 전후뿐 아니라 실제 게시 후에도 보존할 사실과 표 구조를 대조했습니다.

변경 집합·최신 문서

AI 초안·독립 검토

의미·이미지·P/F 대조

게시 후 다시 읽기



AI 초안과 실제 게시 결과를 각각 검증

문장의 자연스러움과 별도로, 지켜야 할 사실과 표 구조를 보존 기준으로 정했습니다.
설명용 도식·도형 개수는 측정값을 나타내지 않습니다.

22 / 22

이미지 보존

30 / 30

입력된 P/F 보존

19 / 19

체크리스트 표 구조

실제 실패 → 보존 기준 보강

공통 코드를 거친 변경이 빠졌습니다

제품 경로만으로 모은 후보에서 공통 코드 변경이 누락됐습니다. 실제 변경 집합과 후보 집합을 대조하고 공통 영역을 거치는 변경까지 추적하도록 보완했습니다.

AI가 검증 조건의 의미를 약하게 만들었습니다

진행 상태를 표시할 시점과 특정 검증 경로가 축약됐습니다. 독립 검토에서 게시를 막고 기존 의미·이미지·미결 질문·입력 결과를 보존 대상으로 고정했습니다.

문서를 읽는 변환기도 검증 대상이었습니다

변환기가 병합 셀의 colspan·rowspan을 버리는 문제가 있었습니다. 이를 보존하도록 수정하고 게시 직전 HTML과 게시 후 표 구조를 대조했습니다.

게시 보존 검증 → 미결 유지

쓴 뒤에 다시 읽는 검사를 추가했습니다

실제 게시 직전 HTML을 기준으로 이미지를 누락·중복 없이 유지하고 기존 P/F와 체크리스트의 논리 열 구조가 같은지 확인했습니다.

한 Staging 실행의 결과를 확인했습니다

15개 항목에서 이미지 22개, 입력 P/F 30개, 체크리스트 표 19개의 보존을 확인했습니다. 게시 전후 비교와 실제 화면 검사도 통과했습니다.

질문이 남은 상태를 완료로 올리지 않았습니다

개발자 확인 질문 6개는 미결로 유지했습니다. 게시·보존 검증의 통과와 최종 릴리즈 확정은 다른 판단으로 관리했습니다.

내 기여: AI가 지켜야 할 사실, 게시 차단 기준, 재시도 증거와 게시 후 대조 규칙을 설계했습니다. 생성 결과를 검토 가능한 문서로 만들고 남은 판단은 해당 항목에 유지했습니다.

TC를 만들고 실행 누락을 막았습니다

릴리즈 근거, TC, 증거, Jira, 변경 이력을 연결했습니다. AI가 제안한 내용이 실제 제품·버전·실행 차수에서 유효한지 확인하는 절차를 설계했습니다.

릴리즈·소스 근거

AI 초안·QA 검토

TC·실행 차수 연결

실제 결과·Jira 대조



284 → 287

진행 중 실행 범위 확장

245건 유지

기존 실행 결과 보존

3건 미실행

새 TC의 시작 상태

문제와 첫 번째 보정

파일에는 있는데 실행 웹에는 없었습니다

TC가 저장된 작업 브랜치와 실제 웹의 데이터 위치가 달랐습니다. 저장소 반영만으로 작업 완료를 판단할 수 없었습니다.

실제 운영 대상을 확인했습니다

제품, 릴리즈, 데이터 위치, 실행 차수를 다시 대조했습니다. 실제 웹에서 TC·FAIL·Jira 연결을 확인하는 완료 검사를 추가했습니다.

정의와 실행 결과를 분리했습니다

TC의 절차와 기대 결과는 정의로, PASS·FAIL과 증거는 실행 이력으로 관리했습니다. AI가 작성한 TC 수를 실행 성공 건수로 계산하지 않았습니다.

두 번째 보정 · 늦게 추가된 TC

실패 · 새 TC가 기존 실행 범위에 없음

실행 차수의 범위는 생성 시점에 고정되어 뒤늦게 만든 TC의 결과를 바로 저장할 수 없었습니다.

해결 · 최신 진행 차수에만 추가 허용

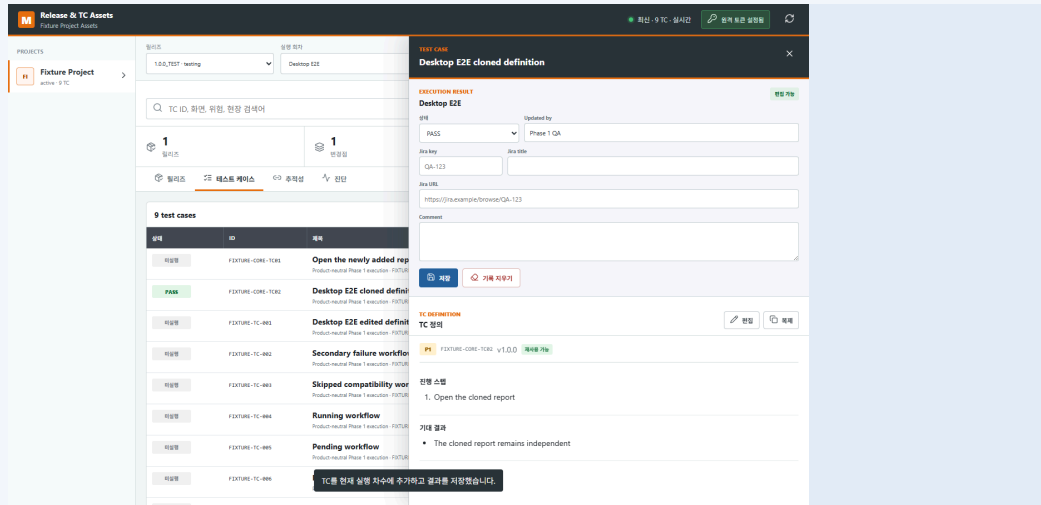
현재 제품·릴리즈의 활성 TC만 추가하고 기존 범위와 결과는 보존했습니다. 동시 수정 여부를 확인하고 종료·과거 차수의 변경은 막았습니다.

검증 · 보존할 값과 바뀔 값 비교

기존 245개 결과를 유지한 채 범위를 284개에서 287개로 늘렸습니다. 추가한 3개에는 결과를 넣지 않았고 관련 변경 시 회귀 115개와 데스크톱 E2E가 통과했습니다.

내 기여: QA 운영 흐름, 제품별 경계, 실행 상태와 완료 기준을 설계하고 AI 구현을 검토했습니다. 실제 실행 기록을 확인한 뒤 이슈와 변경 이력을 마무리했습니다.

기존 결과를 보존하는지, 코드로 확인합니다



실제 E2E 테스트 화면

화면의 9개 TC는 테스트 전용 데이터입니다.
본문의 운영 범위 287개와 다른 데이터입니다.

화면과 코드를 함께 설명할 수 있는 기록

새 항목을 현재 실행 차수에 먼저 추가한 뒤 결과를 저장하도록 UI 흐름도 연결했습니다.

변경 전의 제약

나중에 정의한 TC는 현재 실행 범위에 없어 결과 저장이 거부됐습니다.

변경 후의 기준

최신 진행 차수에만 추가를 허용하고 기존 결과를 보존합니다. 새 TC는 미실행으로 시작합니다.

새 TC를 실행 범위에 넣는 동작과 결과를 저장하는 동작을 분리했습니다. 기존 PASS는 유지되고 새 TC는 결과가 없어야 합니다.

범위만 추가하는 코드

run-store.mjs · L250~254

```
const existingRefs = new Set(record.run.test_case_refs);
const addedRefs = requestedRefs.filter((tcId) => !existingRefs.has(tcId));
if (addedRefs.length === 0) {
  return { record, added_test_case_refs: [] };
}
```

중간 검증 절차 뒤에 실행

run-store.mjs · L263

```
for (const tcId of addedRefs) scope.add(tcId);
```

재검증에서 지킨 두 조건

실제 테스트 코드 발췌

run-scope-admission.test.mjs · L107~108

```
assert.equal(
  resultByTestCase(afterAdmission.run,
    fixture.testCaseIds[0]).status,
  "PASS"
);
assert.equal(
  resultByTestCase(afterAdmission.run, newTcId),
  null
);
```

위 테스트는 기존 TC의 PASS 유지와 새 TC의 결과 없음(null)을 각각 확인합니다. 읽기 편하도록 줄바꿈만 조정했습니다.

범위 추가와 결과 기록을 분리해 테스트를 새로 만들었다는 사실이 실행 성공으로 바뀌지 않게 했습니다. 본문에 제시한 245건 보존은 당시 운영 검증 기록입니다.

게시자는 보는데 독자는 볼 수 없는 이미지

CMS 게시물의 본문은 열리지만 독자에게 이미지가 모두 404로 보였습니다. 문서 게시 성공을 실제 독자가 읽을 수 있는 상태까지 확장해 검증했습니다.

독자 계정으로 재현

파일·첨부 소유권 확인

게시 공간에 복제·연결

독자 화면 재검증

초안 공간

이미지 참조



게시 공간

공간에 복제



읽기 전용 독자

실제 로드 확인



최종 사용자 권한으로 다시 확인

게시 작업의 성공 응답과 독자가 콘텐츠를 읽는 성공을 함께 검증합니다.
설명용 도식. 도형 개수는 측정값을 나타내지 않습니다.

7 / 7

최초 진단 이미지 404

18 / 18

후속 게시 이미지 로드

오류 0

후속 게시 요청·콘솔 검사

문제 → 원인 구분 → 해결

첫 재현 · 본문 성공, 이미지만 실패

독자 권한으로 접속해 7개 이미지 요청이 모두 실패함을 확인했습니다. 페이지가 열리는 것과 첨부가 보이는 것을 나눠 검사했습니다.

원인 · 파일 삭제가 아닌 소유권 불일치

파일과 첨부 레코드는 존재했습니다. 게시 본문은 초안 공간 소유 이미지를 참조했고 초안 접근 권한이 없는 독자는 이미지를 읽을 수 없었습니다.

해결 · 게시 공간 소유 첨부로 재생성

이미지를 게시 대상 공간에 복제하고 본문 참조를 새 첨부로 교체했습니다. 모든 독자에게 초안 권한을 주는 방식은 채택하지 않았습니다.

실패를 다음 배포의 기준으로

이전 접근의 빈틈

본문을 복사하면서 이미지 주소만 유지하면 게시자 확인은 통과해도 독자 경험이 깨질 수 있었습니다.

배포 검사에 권한과 자산을 포함

초안이 보이지 않는 읽기 전용 계정으로 검사했습니다. 이미지 수·소유권·화면 로드·요청 오류·편집 불가를 함께 확인했습니다.

후속 운영에서 재검증

후속 18개 이미지 게시에서 전체 이미지 로드와 요청·콘솔 오류 0건을 확인했습니다. 중간 실패 후 중복 게시가 생기지 않도록 재시도 기준도 관리했습니다.

내 기여: 독자 관점에서 재현하고 파일·권한을 분석해 원인을 찾았습니다. 게시 절차와 재검증 기준도 정립했습니다. 라프텔의 어드민·CMS에서도 운영자와 최종 사용자의 경험을 함께 보겠습니다.

이미 쌓인 행동 기록을 QA에서도 쓰고 싶습니다

기존 MarTech·Analytics 자산을 재사용해 사용자 실패를 찾고 다시 확인하고 다음 릴리즈의 검증으로 남기고 싶습니다.



행동 기록을 사용자가 지나온 길로 보고 멈춘 지점의 조건과 결과를 덧붙입니다.

먼저 있는 것을 확인

이벤트 목록 · 수집 경로 · 담당자

재생 흐름 하나부터

실제 이벤트와 플레이어 상태 대조

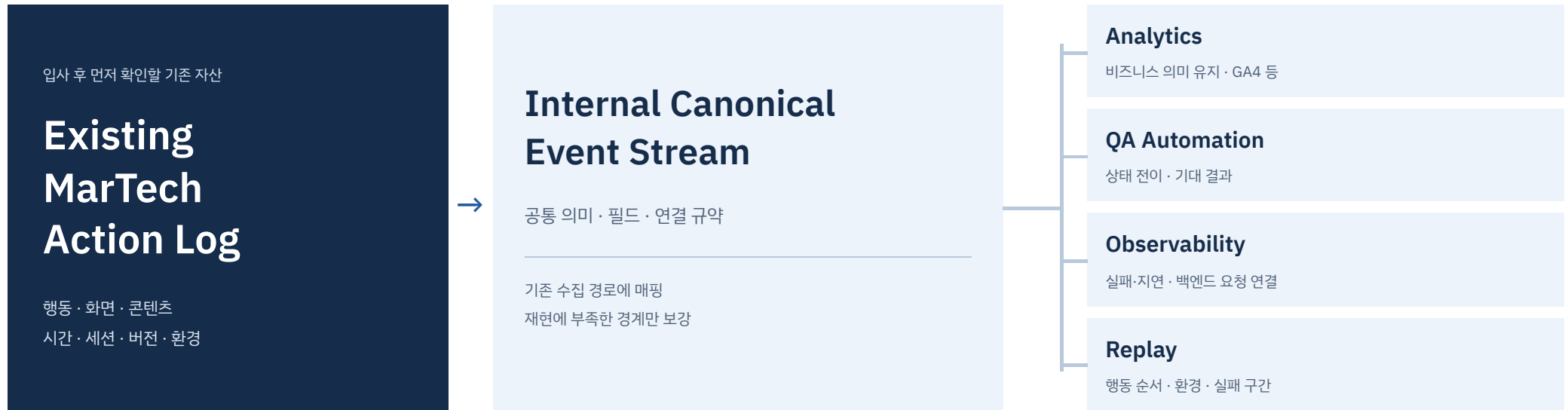
실패를 반복 검사로

재현 · 수정 확인 · 회귀 · 배포 판단

기존 자산은 최대한 재사용하고 실제 사용자 실패가 품질 자산으로 돌아오게 하겠습니다.

하나의 이벤트 자산을 네 가지 목적에 연결합니다

기존 이벤트에 공통 의미를 맞추고 분석·자동화·관측·재현에 필요한 정보를 각 경로에서 활용하겠습니다.



같은 이벤트 자산에서 **용도별 필드와 의미를 선택**합니다. 기술 이벤트 전체를 Analytics에 전달할 필요는 없습니다.

기존 필드 재사용

행동 / 화면·콘텐츠 / 시각 / 사용자·세션
앱·웹 버전 / 기기·OS / 파라미터 / 행동 순서
현재 수집 여부와 의미를 먼저 대조합니다.

QA에 필요한 경계만 보강

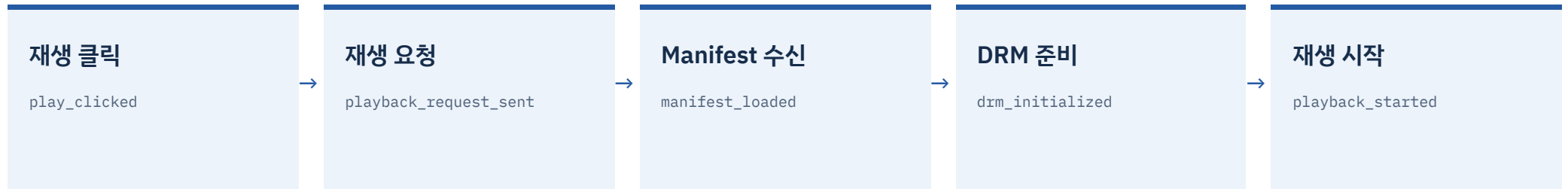
action_id · event_id · trace_id / API·응답 / Player State
Error Code / Feature·Release Version / Correlation ID
schema_version · 순번 · 발생/수집 시각으로 중복·순서·누락도 검증합니다.

공통 규약은 기존 도구 사이를 잇는 기준입니다. 이미 운영 중인 수집기·저장소·러너부터 활용합니다.

사용자와 자동화가 같은 상태 전이를 검증합니다

Play Click부터 실제 Playback까지. 임의의 대기를 줄이고 이벤트가 설명하는 상태와 실제 동작을 함께 확인하겠습니다.

사용자와 테스트가 공유할 관측 기준 `session_id + action_id + release_version`



DRM을 사용하는 재생 경로의 설명용 예시입니다. 실제 플레이어·콘텐츠 경로에 맞춰 상태 계약을 정합니다.



둘을 대조한 뒤 판정 일치 → 결과 기록 / 불일치·누락 → 제품·계측·자동화 원인 구분

이벤트 기반 대기과 실제 상태 검증을 연결해 테스트가 왜 멈췄는지 설명할 근거를 남깁니다.

CS의 실패 기록을 재현과 회귀 후보로 돌려보냅니다

Trace를 실행 가능한 시나리오로 정리하고 QA 환경에서 재현을 시도하겠습니다. CS·QA·개발이 같은 근거를 보게 하겠습니다.

“재생하다가 멈췄어요.” Play → Seek → Subtitle On → Buffering → Stall

CS Issue

신고 시점·세션 연결

→ Existing Trace

행동·상태·오류 구간

→ Replay Scenario

입력·환경·기대 결과

→ QA Reproduction

같은 조건에서 재현 시도

→ Bug Evidence

재현 결과·화면·Trace

재현 성공 → 수정 후 같은 조건 확인

Fix → Retest → Regression Candidate

기대 결과·안정성 검토 후 회귀 세트에 편입

필요 조건이 없거나 재현되지 않으면

재현 보류 · 추가 조건 요청

관찰 기록과 미확인 범위를 함께 보존

실행할 입력

Play · Seek · 자막 변경 + 시험 환경·데이터

관찰할 결과

Buffering · Stall + 화면·플레이어 상태

실제 장애를 일회성 보고서에서 끝내지 않고 수정 검증을 거쳐 다음 릴리즈의 검사로 남깁니다.

실제 사용자 실패가 다음 배포의 검사로 돌아오게

이미 있는 이벤트 자산을 분석·자동화·장애 재현·회귀·배포 판단에 연결하는 QA를 하고 싶습니다.



같은 행동 경로의 오류율

Stall 세션 / 해당 경로 진입 세션

대조 버전 ↔ 카나리 버전

동일 관측 창 · 기기·OS·환경 비교

증거가 모이면

변화·절대 기준·표본·계측 품질 확인

Replay로 원인 후보 좁히기

확대 · 중단 · 롤백 판단 지원

먼저 한 흐름에서 시작

기존 자산 확인 → 재생 Trace

CS 재현 → 회귀 편입

합의한 기준으로 카나리 비교

기존 자산은 최대한 재사용하고 실제 사용자 실패가 자동화된 품질 자산으로 다시 돌아오는 구조를 만들고 싶습니다.