

System Test Results

1. 개요

- 테스트 대상: DB 침입 탐지 시스템 (DB-IDS)
- 테스트 종류: 단위(Unit) / 통합(Integration) / 시스템(System) / 비기능(Non-functional)

2. 테스트 요약

구분	총 케이스 수	성공	실패	성공률
Unit Test	12	10	2	83%
Integration Test	9	7	2	78%
System Test	9	9	0	100%
Non-Functional	8	8	0	100%

3. 단위 테스트 (Unit Test)

3.1. 테스트 환경

- OS: MacOS Sequoia 15.6.1
- JDK: Temurin 21
- 프레임워크: Spring Boot 3.3.2, React 19 / Vite 7
- 빌드 도구: Gradle 8.14
- DB: SQLite 3.45
- 도구: JUnit 5 / Mockito

3.2. 테스트 케이스 상세

3.2.1. UT-01: ProxyRequestHandler

- 모듈: com.example.dbids.modules.proxy.ProxyRequestHandler
- 검증 목적:
 - 수집 계층에서 전달받은 바이트 배열을 정상 UTF-8 SQL 원문으로 추출
 - 비정상 UTF-8를 명시적으로 거부(예외)하는지 확인
- 사전 조건:
 - 입력은 DB 프록시에서 수집된 SQL 원문 바이트 배열
 - 잘못된 UTF-8 시퀀스에 대해서는 IllegalArgumentException("Invalid UTF-8") 기대
 - 판정 근거는 테스트 코드의 assertEquals 통과 여부
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정
UT-01-A	UT-01: SELECT 문장을 UTF-8로 추출한다	"SELECT * FROM users;" 의 UTF-8 바이트	문자열 "SELECT * FROM users;" 로 정확히 복원	동일 문자열 반환	Pass

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정
UT-01-B	UT-01: 비정상 UTF-8 바이트 시퀀스는 거부한다	바이트 [0xC3, 0x28] (잘못된 UTF-8)	IllegalArgumentException 발생 및 메시지에 "invalid UTF-8" 포함	동일 예외 및 메시지 확인	Pass

3.2.2. UT-02: ProxyResponseHandler

- 모듈: com.example.dbids.modules.proxy.ProxyResponseHandler
- 검증 목적: JDBC ResultSet 처리 결과를 요약 메타(RespMeta)로 산출할 때
 - 정상 흐름에서 행 수(rowCount)를 정확히 집계하고 status=SUCCESS를 설정하는지,
 - 처리 중 예외 발생 시 rowCount=0 및 status=FAILURE로 안전하게 귀결하는지 확인
- 사전 조건:
 - ResultSet#next() 동작을 목 객체로 시뮬레이션
 - 판정 근거는 테스트 코드의 assertEquals 통해 여부
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정
UT-02-A	UT-02: ResultSet 10행 → RowCount=10, Status=SUCCESS	when(rs.next()) 가 true 10 회 후 false 반환	meta.rowCount == 10, meta.status == "SUCCESS"	동일	Pass
UT-02-B	UT-02: ResultSet 처리 중 예외 → RowCount=0, Status=FAILURE	when(rs.next()) 가 SQLException("boom") 던짐	meta.rowCount == 0, meta.status == "FAILURE"	동일	Pass

3.2.3. UT-03-RE / UT-04-RE: RuleEngine, SqlNormalizer

- 모듈:
 - com.example.dbids.modules.rule.RuleEngine
 - com.example.dbids.modules.rule.SqlNormalizer
- 검증 목적:
 - SqlNormalizer가 주석 제거, 공백 정규화, 대문자화, 리터럴 마스킹을 일관되게 수행하는지.
 - RuleEngine이 정규화된 SQL에서 공격 패턴을 정확히 매칭하고, 최고 심각도 우선 규칙과 정상 쿼리 무매칭을 보장하는지.
- 사전 조건:
 - RuleSetProvider.defaultRules() 사용(기본 룰셋)
 - 판정 근거는 각 테스트의 assertEquals / assertTrue / assertFalse 통해 여부
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정
UT-03-RE-A	DROP TABLE → 매칭 성공, severity=HIGH	/*c*/ drop table students; → 정규화	매칭 있음, severity=HIGH, 정규화 결과에 DROP TABLE 포함	동일	Pass
UT-03-RE-B	UNION SELECT → 매칭 성공, severity=MEDIUM	... UNION SELECT ...	매칭 있음, severity=MEDIUM, 정규화 결과에 UNION SELECT 포함	동일	Pass
UT-03-RE-C	OR 1=1 → 매칭 성공, severity=MEDIUM	... WHERE ... OR 1 = 1	매칭 있음, severity=MEDIUM, 정규화 결과 숫자 마스킹으로 OR 0 = 0 포함	동일	Pass
UT-03-RE-D	SLEEP() → 매칭 성공, severity=LOW	... OR SLEEP(5)	매칭 있음, severity=LOW, 정규화 결과에 SLEEP(포함	동일	Pass

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정
UT-03-RE-E	복수 패턴 동시 매칭 시 최고 severity 선택	DROP TABLE ...; ... UNION SELECT ...;	매칭 있음, severity=HIGH (HIGH vs MEDIUM 중 HIGH 선택)	동일	Pass
UT-04-RE	정상 SELECT → 매칭 없음	SELECT name FROM users;	매칭 없음	동일	Pass

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과(정규화 결과)	실제 결과	판정
UT-03/UT-04-RE-A	블록/라인 주석 제거, 공백 정리, 대문자화	/* note */ select * from users -- tail\n where id=1	SELECT * FROM USERS WHERE ID=0 (주석 제거, 공백 정리, 대문자화, 숫자 0 마스킹)	동일	Pass
UT-03/UT-04-RE-B	문자열/숫자 리터럴 마스킹	... a='abc'def' AND b="x\y" AND c=12345	... A=? AND B=? AND C=0	동일	Pass
UT-03/UT-04-RE-C	앞뒤 공백 제거 및 단일 공백화	" \n\t SELECT 1 \n "	SELECT 0	동일	Pass

3.2.4. UT-03-DS / UT-04-DS: `DetectionService`

- 모듈: `com.example.dbids.modules.detection.DetectionService`
- 검증 목적: 정규화된 SQL에 대해 패턴 탐지 이벤트 생성/심각도 산정, QueryLog 상태 보정 규칙(HIGH→FAILURE), 알림 트리거 호출을 단위 수준에서 검증.
- 사전 조건:
 - Mock: `DetectionEventRepository`, `NotificationService`, `QueryLogRepository`
 - 근거: `assert*` 통과 + `verify(...)` 호출 검증
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정/근거
UT-03-DS-A	DROP TABLE → PATTERN/HIGH + 상태 보정 + 알림	/*comment*/ drop table students ; (초기 QueryLog.status=SUCCESS)	이벤트 생성, <code>Type=PATTERN</code> , <code>Severity=HIGH</code> , <code>ev.rawQuery == normalize(sql)</code> , <code>QueryLog.status=FAILURE</code> 로 보정 및 <code>logRepo.save()</code> 1회, <code>notifier.onEvent()</code> 호출	전부 일치(정규화 결과에 <code>DROP TABLE</code> , 주석 제거 확인)	Pass (<code>assertEquals/True</code> + <code>verify(save, times(1))</code> , <code>verify(onEvent, times(1))</code>)
UT-03-DS-B	UNION SELECT → PATTERN/MEDIUM (보정 없음) + 알림	... UNION SELECT ...	이벤트 생성, <code>Severity=MEDIUM</code> , 상태 보정/저장 없음, 알림 호출 0	전부 일치 (<code>logRepo.save()</code> 호출 0)	Pass (<code>assert*</code> + <code>verify(never())/times(1)</code>)
UT-03-DS-C	OR 1=1 → PATTERN/MEDIUM (정규화로 0=0) + 알림	... OR 1=1 -- bypass	이벤트 생성, <code>Severity=MEDIUM</code> , 정규화 결과에 <code>OR 0=0</code> 포함, 주석 제거, 보정 없음, 알림 호출	전부 일치	Pass
UT-03-DS-D	SLEEP() → PATTERN/LOW (보정 없음) + 알림	... OR SLEEP(10)	이벤트 생성, <code>Severity=LOW</code> , 정규화 결과에 <code>SLEEP()</code> 포함, 보정 없음, 알림 호출	전부 일치	Pass
UT-04-DS-A	정상 SELECT → 이벤트/저장/알림 없음	SELECT name FROM users;	이벤트 없음 (<code>eventId.isEmpty()</code>), 레포/알림 호출 없음, 원상태 유지	전부 일치	Pass (<code>verify(never())</code> 일괄 확인)

3.2.5. UT-05 / UT-06: BehaviorDetector

- 모듈: `com.example.dbids.modules.behavior.BehaviorDetector`
- 검증 목적: 단위 시간 창(window) 내 질의 폭주/행동 패턴을 점수화하여 이상 탐지 이벤트(BEHAVIOR)를 생성하고, 임계치 설정에 따라 발생/비발생 분기 및 알림 트리거 호출이 기대대로 동작하는지 확인.
- 사전 조건:
 - Mock: `DetectionEventRepository`, `NotificationService`
 - 파라미터: `BehaviorProperties(window=1s, thresholdMedium=0.8, thresholdHigh=0.95, wQpm=1.0, 나머지 0.0)`
 - 판정 근거: `assert*` 통과 + `verify(...)` 호출 여부
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정/근거
UT-05	1초에 200개 SELECT → Score ≥ 0.8, BEHAVIOR 이벤트 + 알림	같은 <code>Instant (t0)</code> 에 서 200건, +2초 후 <code>flush</code> 1건	<code>DetectionEvent.Type=BEHAVIOR</code> , <code>Severity ∈ {MEDIUM, HIGH}</code> 생성, <code>notifier.onEvent()</code> 호출	이벤트 저장 ≥1회, 마 지막 캡처 <code>Type=BEHAVIOR</code> , <code>Severity</code> MED/HIGH, 알림 ≥1 회	Pass (<code>verify(eventRepo,</code> <code>atLeastOnce)</code> , <code>verify(notifier,</code> <code>atLeastOnce)</code>)
UT-06	행 반환 수=100(저활 동) → 이벤트/알림 없음 (임계↑으로 강제 비발생)	임계 Medium/High를 매우 크게 설정, 동 일 시작 2건 + <code>flush</code>	이벤트 미발생, 알림 미호출	저장/알림 호출 0회	Pass (<code>verify(...,</code> <code>never()</code>)
UT-05-A	경계 하회: 낮은 활동 량 → 이벤트 없음 (임 계↑)	임계↑ 설정, 20건 + <code>flush</code>	이벤트/알림 없음	저장/알림 호출 0회	Pass
UT-05-B	경계 상회: 임계 이상 활동량 → 이벤트 발 생	기본 임계 (0.8/0.95), 80건 + <code>flush</code>	이벤트 ≥1회, 알림 ≥1회	저장/알림 모두 호 출됨	Pass

3.2.6. UT-07 / UT-08: AuthZ

- 모듈:
 - 서비스: `com.example.dbids.modules.authz.AuthZService`
 - 엔진: `com.example.dbids.modules.authz.AuthZEngine`
- 검증 목적:
 - 롤 기반 권한정책(READ_ONLY/DBA)에 따라 권한 위반 탐지(AUTHZ 이벤트),
 - 상태 보정(FAILURE), 알림 연계, 엔진 규칙 매칭이 STP 기준대로 동작하는지 확인.
- 사전 조건:
 - Mock: `DetectionEventRepository`, `NotificationService`
 - 파라미터: `BehaviorProperties(windowSeconds=1, thresholdMedium=0.8, thresholdHigh=0.95, wQpm=1.0, wWriteRatio=0.0, wDdlPerMin=0.0, wErrorBurst=0.0)`
 - 데이터 타임라인: 실제 `sleep` 없이 `executedAt` (ISO8601)만 사용하여 시점(t0, t0+2s) 플러시
 - 판정 근거: JUnit `assert*` 통과 + Mockito `verify(...)` 호출 여부
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정/근거
UT-07	READ_ONLY 사용자의 DELETE → AUTHZ 이	role=READ_ONLY, SQL= <code>DELETE FROM</code>	이벤트 생성(Type=AUTHZ, Sev=HIGH),	전부 일치	Pass (<code>assert*</code> , <code>verify(...)</code>)

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정/근거
	벤트 생성 + 상태보정(Failure) + 알림	orders;	QueryLog.status=FAILURE, eventRepo.save() 1회, notifier.onEvent() 1회		
UT-08	DBA 사용자의 DELETE → 이벤트 없음, 보정/알림/저장 없음	role=DBA, SQL= DELETE FROM orders;	이벤트 없음, 상태 유지(SUCCESS), 저장/알림 호출 0	전부 일치	Pass
UT-08-A	userRole 미지정 → 정책 비적용(허용) → 이벤트 없음	userRole 없음, SQL= DELETE ...	이벤트/알림/저장 없음	전부 일치	Pass
UT-08-B	fail-open — 이벤트 저장 종 예외 → Optional.empty() 반환, 흐름 차단 없음	eventRepo.save() 예외	반환 empty, 알림/보정 저장 시도 없음	전부 일치	Pass

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정/근거
UT-07-A	엔진: READ_ONLY + DELETE orders → Violation 반환	user= ro-user, SQL= DELETE FROM orders;	Violation.present() ==true, action=DELETE, ruleMatched 에 DENY DELETE:* 포함	전부 일치	Pass
UT-08-C	엔진: DBA + DELETE → Violation 없음	user= dba-user, SQL= DELETE ...	Violation.empty()	일치	Pass
UT-07-B	엔진: 테이블 추정 실패 시 *로 평가되어 DENY 매칭	user= ro-user, SQL 비정형	Violation.present() ==true	일치	Pass

3.2.7. UT-09 / UT-10: Notification

- 모듈: EmailSender, SlackSender, NotificationService
- 검증 목적:
 - 채널별 전송 동작>Email/Slack,
 - 설정 플래그/필수값 검증,
 - 실패 시 예외 처리 및 NotificationLog(Status=SENT/FAILED) 기록을 단위 수준에서 확인.

3.2.7.1. EmailSenderTest

- 사전 조건:
 - Mock: JavaMailSender
 - 프로퍼티: NotifierProperties.email.enabled, from, toDefault
 - 판정 근거: assert* + verify(mail.send(...)) 호출 여부
- 테스트 케이스 및 결과

TC ID	테스트명	입력(요약)	기대 결과	실제 결과	판정/근거
UT-10-A	send() → toDefault로 전송, props.from 우선	enabled=true, from, toDefault 설정 후 send("SUBJ","BODY")	JavaMailSender#send 1회, to == toDefault	동일	Pass (verify(send), assertEquals(to))
UT-10-B	sendTo() → from 없으면 springMailUsername 폴백	from=null, 리플렉션으로 springMailUsername 설정 후 sendTo(...)	전송 1회(폴백 사용), 수신자는 인자로 지정한 값	동일	Pass (verify(send), assertEquals(to))
UT-10-C	disabled=true → 전송 호출 없음	enabled=false	sendTo 호출해도 JavaMailSender#send	동일	Pass (verify(never()))

TC ID	테스트명	입력(요약)	기대 결과	실제 결과	판정/근거
			미호출		
UT-10-D	toDefault 누락 → IllegalStateException	enabled=true, toDefault=null 후 send(...)	예외 발생, 전송 호출 없음	동일	Pass (<code>assertThrows</code> , <code>verify(never())</code>)

3.2.7.2. SlackSenderTest

- 사전 조건:
 - 임베디드 HTTP 서버로 웹훅 엔드포인트 모사(`/ok` → 200, `/err` → 500)
 - 프로퍼티: `NotifierProperties.slack.enabled`, `webhookUrl`
 - 판정 근거: `assert*` + hit count/요청 payload 기록
- 테스트 케이스 및 결과

TC ID	테스트명	입력(요약)	기대 결과	실제 결과	판정/근거
UT-09-A	200 OK → 전송 성공 (SENT에 해당)	<code>/ok</code> 엔드포인트로 send("hello world")	호출 ≥1, payload에 "text" 와 메시지 포함	동일	Pass (hitCount, body 확인)
UT-09-B	500 응답 → RuntimeException	<code>/err</code> 로 send("oops")	예외 메시지에 SLACK_WEBHOOK_HTTP_500 포함	동일	Pass (<code>assertThrows</code>)
UT-09-C	disabled=true → 호출 없이 리턴	<code>enabled=false</code>	hitCount=0	동일	Pass
UT-09-D	webhookUrl 누락 → IllegalStateException	<code>webhookUrl=null</code>	예외 발생	동일	Pass

3.2.7.3. NotificationServiceTest

- 사전 조건:
 - Mock: `EmailSender`, `SlackSender`, `NotificationFormatter`, `SlackFormatter`, `CurrentAdminEmailResolver`, `QueryLogRepository`, `NotificationLogRepository`
 - 프로퍼티: `NotifierProperties`
 - 판정 근거: `assert*` + `verify(...)` 호출/저장 로그 캡처(`NotificationLog`)
- 테스트 케이스 및 결과

TC ID	테스트명	입력(요약)	기대 결과	실제 결과	판정/근거
UT-10-A	Email ON, Slack OFF → EMAIL/SENT 1건	이메일 ON, Slack OFF	<code>emailSender.send()</code> 1회, <code>notifRepo.save(EMAIL,SENT)</code> 1건	동일	Pass
UT-09-A	Slack ON, Email OFF → SLACK/SENT 1건	슬랙 ON, Email OFF	<code>slackSender.send()</code> 1회, <code>notifRepo.save(SLACK,SENT)</code> 1건	동일	Pass
UT-09-B / UT-10-B	Email+Slack ON → 각 채널 SENT 1건씩	둘 다 ON	email/slack 각각 1회 호출, <code>NotificationLog</code> 2건(EMAIL/SENT, SLACK/SENT)	동일	Pass (2건 저장 검증)
UT-10-C	Email 실패 → EMAIL/FAILED 저장	Email ON, <code>emailSender</code> 예외	<code>notifRepo.save(EMAIL,FAILED)</code> 1건(오류 코드/메시지 포함)	동일	Pass
UT-10-D	Slack 실패 → SLACK/FAILED 저장	Slack ON, <code>slackSender</code> 예외	<code>notifRepo.save(SLACK,FAILED)</code> 1건(오류 코드/메시지 포함)	동일	Pass

3.2.8. UT-11: StorageService

- 모듈: `com.example.dbids.modules.storage.StorageService`
- 검증 목적:

- 쿼리 로그 저장 로직의 유효성 검증/정규화/절단 저장/리포지토리 연동을 단위 수준에서 검증
- 사전조건:
 - `QueryLogRepository` 를 Mock으로 주입(`@Mock` , `@InjectMocks`)
 - 판정 근거는 `assertEquals` , `assertThrows` , `verify` 통과 여부
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정/근거
UT-11-A	정상 insert → repo.save 1회, id 반환	<code>repo.save()</code> 가 저장 엔티티의 <code>id=100L</code> 로 세팅해 반환하도록 스텝	<code>save()</code> 1회 호출, 반환 <code>id != null</code>	<code>save()</code> 1회 검증, 반환 <code>100L</code> 로 <code>NotNull</code>	Pass (<code>assertNotNull(save())</code> , <code>verify(times(1)).never().save()</code>)
UT-11-B	필수 누락/음수/빈값 → <code>IllegalArgumentException</code>	<code>userId=" "</code> , <code>sqlRaw=null</code> , <code>returnRows=-1</code> , <code>status=""</code> 각각 호출	각 케이스마다 <code>IllegalArgumentException</code> 발생, <code>repo.save()</code> 호출 안 함	예외 4건 모두 발생, <code>save()</code> 호출 0회	Pass (<code>assertThrows(IllegalArgumentException.class).atLeastOnce().when(repo).save(eq("")).thenReturn(null)</code> , <code>verify(never()).save(eq(" "))</code>)
UT-11-C	시간 포맷 오류 → invalid <code>executedAt</code>	<code>executedAt="2025/10/21 12:00"</code> (잘못된 포맷)	<code>IllegalArgumentException("invalid executedAt")</code>	동일 예외 및 메시지 포함	Pass (<code>assertThrows(IllegalArgumentException.class).atLeastOnce().when(repo).save(eq("2025/10/21 12:00"))</code> , <code>verify(never()).save(eq("2025/10/21 12:00"))</code>)
UT-11-D	잘못된 status → invalid <code>status</code>	<code>status="DONE"</code>	<code>IllegalArgumentException("invalid status")</code>	동일 예외 및 메시지 포함	Pass (<code>assertThrows(IllegalArgumentException.class).atLeastOnce().when(repo).save(eq("2025/10/21 12:00"))</code> , <code>verify(never()).save(eq("2025/10/21 12:00"))</code>)
UT-11-E	sqlRaw 8KB 초과는 절단 저장	<code>sqlRaw</code> 길이 9000	저장 엔티티의 <code>sqlRaw.length() == 8192</code>	캡처된 엔티티의 길이 8192	Pass (<code>assertEquals(8192, captured.getSqlRaw().length())</code>)
UT-11-F	sqlSummary 512 초과는 절단 저장	<code>sqlSummary</code> 길이 600	저장 엔티티의 <code>sqlSummary.length() == 512</code>	캡처된 엔티티의 길이 512	Pass (<code>assertEquals(512, captured.getSqlSummary().length())</code>)

3.3. 테스트 요약

TC ID	대상 모듈	판정	비고
UT-01	Proxy Request Handler	Pass	-
UT-02	Proxy Response Handler	Pass	-
UT-03	RuleEngine	Pass	-
UT-04	RuleEngine	Pass	-
UT-05	BehaviorEngine	Pass	-
UT-06	BehaviorEngine	Pass	-
UT-07	AuthZEngine	Pass	-
UT-08	AuthZEngine	Pass	-
UT-09	Notifier (Slack)	Pass	-
UT-10	Notifier (Email)	Pass	-
UT-11	Storage (SQLite)	Pass	-
UT-12	Storage (S3 Job)	Fail	기능 미구현

4. 통합 테스트 (Integration Test)

4.1. 테스트 환경

- OS: MacOS Sequoia 15.6.1
- JDK: Temurin 21
- 프레임워크: Spring Boot 3.3.2, React 19 / Vite 7
- 빌드 도구: Gradle 8.14

- DB: SQLite 3.45
- 도구: Mailhog, WireMock

4.2. 테스트 케이스 상세

4.2.1. IT-01: Query Logging

- 모듈/대상: API /api/ingest/log, API /api/logs
- 검증 목적:
 - 수집 API로 전달된 쿼리가 정상 저장되는지 확인
 - 로그 조회 API에서 방금 저장한 레코드가 조회되는지 확인
- 사전 조건:
 - profile=local, baseUrl=http://localhost:8080
 - 헤더: Content-Type: application/json
 - 판정 근거: 상태코드/본문 키 확인 및 POST 응답 id 와 GET 결과 content[0].id 일치 여부
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과 (요약)	판정
IT-01-A	IT-01: 수집 엔드포인트에 INSERT (POST /api/ingest/log)	Body: { "executedAt": null, "userId": "it01@test.com", "sqlRaw": "SELECT * FROM users;" , "sqlSummary": "SELECT users", "returnRows": 3, "status": "SUCCESS" }	200/201 응답, 본문에 {"id":<생성ID>} 포함	200 OK, {"id":"de8d7091-5fad-41f5-b2bf-969663250d8e"}	Pass
IT-01-B	IT-01: 조회 엔드포인트로 방금 데이터 확인 (GET /api/logs)	Query: ? user=it01@test.com&size=10	200 OK, 본문에 "content": [...], totalElements ≥ 1	200 OK, totalElements: 1, content[0].id="de8d7091-5fad-41f5-b2bf-969663250d8e", userId="it01@test.com", sqlRaw="SELECT * FROM users;" , returnRows=3, status="SUCCESS", executedAt="2025-10-29T12:14:32.735577Z"	Pass
IT-01-C	IT-01: POST/GET 일치성 검증	POST 응답의 id ↔ GET content[0].id 비교	두 값 일치	POST id = de8d7091-5fad-41f5-b2bf-969663250d8e ↔ GET content[0].id 동일	Pass
IT-01-D	IT-01: 필드 정합성 확인 (status, rows, sqlRaw)	GET 본문 내 content[0].status, returnRows, sqlRaw 확인	status="SUCCESS", returnRows=3, sqlRaw="SELECT * FROM users;"	값이 모두 기대와 일치	Pass

- 증빙
 - POST 응답 원문: {"id":"de8d7091-5fad-41f5-b2bf-969663250d8e"}
 - GET 응답 요약: totalElements:1, content[0].id="de8d7091-5fad-41f5-b2bf-969663250d8e", userId="it01@test.com", sqlRaw="SELECT * FROM users;", returnRows=3, status="SUCCESS"
- CLI 쿼체크

```
▶ LOG_ID=$(curl -s -X POST http://localhost:8080/api/ingest/log -H 'Content-Type: application/json' -d '{"executedAt":null,"userId":"it01@test.com","sqlRaw":"SELECT * FROM users;","sqlSummary":"SELECT users","returnRows":3,"status":"SUCCESS"}' | jq -r .id); curl -s "http://localhost:8080/api/logs?user=it01@test.com&size=10" | jq --arg L "$LOG_ID" '$.content[] | select(.id==$L) | length' | awk '{print ($1>0)?FOUND:"NOT FOUND"}'
FOUND
```

```

LOG_ID=$ curl -s -X POST http://localhost:8080/api/ingest/log -H "Content-Type: application/json" -d '{"executedAt":null,"userId":"it01@test.com","sqlRaw":"SELECT * FROM users;","sqlSummary":"SELECT users","returnRows":3,"status":"SUCCESS"}' | jq --arg L "$LOG_ID" --e '$content[] | select(.id==$L and .status=="SUCCESS" and .returnRows==3 and .sqlRaw=="SELECT * FROM users;")' >/dev/null && echo PASS || echo FAIL

```

4.2.2. IT-02: PatternEvent

- 모듈/대상: API /api/ingest/log, API /api/events?type= PATTERN
- 검증 목적:
 - 공격성 SQL(DROP TABLE ...) 수집 시 PATTERN/HIGH 이벤트가 생성되는지 확인
 - 이벤트 목록에서 해당 logId로 정확히 조회되는지 확인
- 사전 조건:
 - profile=local, baseUrl=http://localhost:8080
 - 헤더: Content-Type: application/json
 - 외부 연동(Slack/Email): 무관(본 TC는 이벤트 생성/표시만 검증)
 - 판정 근거: 상태코드/본문 키 확인 및 POST 응답 id = 이벤트.logId 일치 여부, 이벤트 타입/심각도 확인
- 테스트 케이스 및 결과

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과 (요약)	판정
IT-02-A	IT-02: 공격성 SQL 수집 (POST /api/ingest/log)	Body: { "userId": "martin115@example.com", "sqlRaw": "DROP TABLE secret_data", "sqlSummary": "drop table secret_data", "returnRows": 0, "status": "SUCCESS" }	200/201 응답, 본문에 {"id":<생성ID>} 포함	200 OK, {"id": "4a0e07de-c709-4948-8439-b4a9d7a6e808"}	Pass
IT-02-B	IT-02: PATTERN 이벤트 조회 및 필터 (GET /api/events? type=PATTERN&size=100)	Query: type=PATTERN&size=100 → 결과 다건이므로 `jq --arg L "\$LOG_ID" '.content[]'	select(.logId==\$L)로 logId 기반 필터	200 OK, 목록 중 logId == <POST id> 인 이벤트 ≥ 1 건	200 C {"id": "2e8192c709-4b4a9d"}
IT-02-C	IT-02: 이벤트 속성 검증 (타입/심각도/프리뷰)	(B)의 필터 결과에서 eventType, severity, 미가공 SQL 프리뷰 확인	eventType=="PATTERN" AND severity=="HIGH"	eventType:"PATTERN", severity:"HIGH", userId:"martin115@example.com", sqlPreview:"DROP TABLE SECRET_DATA"	Pass

• 증빙

- POST 응답: {"id": "4a0e07de-c709-4948-8439-b4a9d7a6e808"}
- GET 응답(해당 건):

```
{"id": "37b9ef59-393d-4e5b-9177-2e81920108bc", "logId": "4a0e07de-c709-4948-8439-b4a9d7a6e808", "eventType": "PATTERN", "severity": "HIGH", ..., "userId": "martin115@example.com", "sqlPreview": "DROP TABLE SECRET_DATA"}
```

• CLI 퀴체크

```

curl -s "http://localhost:8080/api/events?type=PATTERN&size=100" | jq --arg L "4a0e07de-c709-4948-8439-b4a9d7a6e808" --e '.content[] | select(.logId==$L) | length' | awk '{print $1>0 ? "FOUND" : "NOT FOUND"}'

```

```

curl -s "http://localhost:8080/api/events?type=PATTERN&size=100" | jq --arg L "4a0e07de-c709-4948-8439-b4a9d7a6e808" --e '.content[] | select(.logId==$L and .severity=="HIGH")' >/dev/null && echo "severity=HIGH OK"

```

4.2.3. IT-03: BehaviorEvent

- 모듈: com.example.dbids.modules.behavior.BehaviorDetector, com.example.dbids.modules.notify.NotificationService
- 엔드포인트: POST /api/ingest/log, GET /api/events?type=BEHAVIOR
- 검증 목적
 - 동일 사용자 고빈도 요청(스파이크) 시 Type=BEHAVIOR 이벤트가 생성되는지

- 이벤트 심각도(Severity)가 정책 임계치 이상일 때 **HIGH**로 표기되는지
 - 조회 API에서 해당 사용자 필터로 이벤트를 검색 가능하는지
 - 사전 조건
 - `baseUrl=http://localhost:8080` (`profile=local`)
 - 행동 탐지 파라미터: `windowSeconds=60`, `thresholdMedium=1.0`, `thresholdHigh=2.0`
 - 판정 근거: 아래 Pass/Fail 기준 총족 여부 + 실제 Evidence
 - 테스트 케이스 및 결과

TC ID	테스트명	입력(요약)	기대 결과	실제 결과(제출 값 요약)	판정
IT-03-A	BEHAVIOR 이벤트 생성/조회	동일 사용자 behav@test.com에 대해 1초 내 다건 POST /api/ingest/log, 이후 GET /api/events? type=BEHAVIOR&user=...	200 OK, content[0].eventType=="BEHAVIOR", userId="behav@test.com", 페이지 totalElements=1	content[0].eventType=="BEHAVIOR", userId="behav@test.com", 페이지 totalElements=1	Pass
IT-03-B	임계 초과(QPM) 로 HIGH 확인	동일 입력; sqlPreview 내 QPM 수치 확인	content[0].severity=="HIGH", sqlPreview 내 QPM 임계 이상	severity="HIGH", sqlPreview="USER-behav@test.com QPM=200.00..."	Pass

- 증빙
 - GET 응답

```
{"id": "fe47f6e-ff4e-4467-a4e3-d6011f12d42", "logId": "5c87191a-f0cb-4931-8fb8-3499a267ff53", "eventType": "BEHAVIOR", "severity": "HIGH", "occurredAt": "2023-09-29T12:55:25.691259Z", "userId": "behav@test.com", "adminId": null, "sqlPreview": "USER = behav@test.com QPM=200.00 WRITE_RATIO=0.00 DDL/m=0.00 SCORE=200.00"}
```

- CLI Quick Check

```
> for i in $(seq 1 500); do curl -s -X POST http://localhost:8080/api/ingest/log -H "Content-Type: application/json" -d "{\"userId\":\"behav@test.com\", \"solrRaw\": \"<SELECT 1>\", \"returnRows\":1, \"status\":\"SUCCESS\"}" >/dev/null ; done; curl -s http://localhost:8080/api/events?type=BEHAVIOR&user=behav@test.com&size=10 | jq -e ".content[] | select(.eventType==\"BEHAVIOR\" and .severity==\"HIGH\")" >/dev/null && echo PASS || echo FAIL
PASS
```

4.2.4. IT-04: AuthZEvent

- 모듈: `com.example.dbids.modules.authz.AuthZService`, `com.example.dbids.modules.authz.AuthZEngine`
 - 엔드포인트: `POST /api/ingest/log`, `GET /api/events?type=AUTHZ&userId=...`
 - 검증 목적
 - READ_ONLY 사용자로 금지된 쓰기(INSERT/UPDATE/DELETE) 시 `Type=AUTHZ` 이벤트 생성
 - 정책상 위반 시 `severity=HIGH` 로 기록
 - 조회 API에서 `userId` 파라미터로 해당 이벤트 검색 가능
 - 사전 조건
 - `baseUrl=http://localhost:8080` (`profile=local`)
 - RBAC 기본 가정: READ_ONLY → `DENY INSERT/UPDATE/DELETE/DML`
 - 판정 근거: 아래 Pass/Fail 총점 + Evidence
 - 테스트 케이스 및 결과

TC ID	테스트명	입력(요약)	기대 결과	실제 결과(요약)	판정
IT-04-A	AUTHZ 이벤트 생성	userId=authz@test.com 로 INSERT INTO users...	201/200 , 이벤트 목록에서 eventType=AUTHZ 존재	POST 200, GET 200, AUTHZ 이벤트 발견	Pass

TC ID	테스트명	입력(요약)	기대 결과	실제 결과(요약)	판정
IT-04-B	심각도 HIGH	동일 입력	severity="HIGH"	이벤트 severity="HIGH" 확인	Pass
IT-04-C	logId 일치성	POST 응답의 <code>id</code> 와 이벤트 <code>logId</code>	<code>logId == id</code>	jq 필터로 동일성 확인됨	Pass

- 증빙

- POST 응답

```
{"id":"271cb54e-2485-4d82-aa2d-1e2565af834b"}
```

- GET 응답

```
{"id":"Oeb5c76b-4d11-4b59-a352-262238bd0735","logId":"5ac33925-d400-4685-8a66-20e80885fdb6","eventType":"AUTHZ","severity":"HIGH",...,"sqlPreview":"INSERT INTO users(id) VALUES (1)"}
```

- CLI Quick Check

```
> R=$(curl -s -X POST http://localhost:8080/api/ingest/log -H "Content-Type: application/json" -d "{\"userId\":\"authz@test.com\", \"sqlRaw\":\"INSERT INTO users(id) VALUES (1)\", \"status\":\"FAILURE\"}"; ID=$(echo "$R" | jq -r ".id"); curl -s "http://localhost:8080/api/events?type=AUTHZ&userId=authz@test.com&size=100" | jq -e --arg L "$ID" ".content[] | select(.logId==\$L and .severity==\"HIGH\")" >/dev/null && echo FAIL || echo PASS
PASS
```

4.2.5. IT-05: Notification

- 모듈

```
com.example.dbids.modules.notify.NotificationService,  
EmailSender, SlackSender, NotificationFormatter, SlackFormatter,  
com.example.dbids.modules.auth.CurrentAdminEmailResolver
```

- 엔드포인트

- POST /api/ingest/log
- GET /api/events?type=PATTERN&user=...&size=...
- GET /api/events/{eventId}

- 검증 목적

- 공격성 SQL 수집 시 Type=PATTERN, severity=HIGH 이벤트 생성
- 이벤트 트리거 시 Email 과 Slack 모두 발송 성공 확인
(Slack은 WireMock 웹훅으로 대체)

- 이벤트의 logId 가 직전 POST 응답 id 와 일치

- 사전 조건

- baseUrl=http://localhost:8080 (profile=local)
- Email=Mailhog, Slack Webhook=WireMock
- WireMock 기동: Docker를 통해 기동
- WireMock 매핑(슬랙 푸):

```
curl -X PUT http://localhost:9090/_admin/mappings/new \
-H "Content-Type: application/json" \
-d '{
  "request": { "method": "POST", "urlPath": "/slack" },
  "response": { "status": 200, "jsonBody": { "ok": true } }
}'
```

- 앱/프로퍼티

```

notifier:
  email:
    enabled: true
    from: noreply@dbids.local
    to-default: admin@dbids.local
  slack:
    enabled: true
    webhook-url: http://localhost:9090/slack

```

- 요청/응답

- POST `/api/ingest/log` → 201 Created
 - Header: `Location: /logs/ea50e18b-0fbd-4bb7-89ec-1069b58241c5`
 - Body: `{"id": "ea50e18b-0fbd-4bb7-89ec-1069b58241c5"}`
- GET `/api/events?type=PATTERN&user=alert-ok@test.com&size=5` → 200 OK

첫 건 핵심:

- `eventType="PATTERN"`, `severity="HIGH"`
- `logId="ea50e18b-0fbd-4bb7-89ec-1069b58241c5"` (`POST id`와 일치)
- `sqlPreview="DROP TABLE IMPORTANT_TABLE"`

- 테스트 케이스 및 결과

TC ID	테스트명	입력(요약)	기대 결과	실제 결과(요약)	판정
IT-05-A	PATTERN/HIGH 이벤트 생성	<code>userId="alert-ok@test.com"</code> , <code>sqlRaw="DROP TABLE important_table"</code>	<code>POST</code> 200/201, <code>GET</code> 에서 이벤트 존재	<code>POST</code> 201, <code>GET</code> 200, PATTERN 이벤트 발견	Pass
IT-05-B	Email 발송 성공	IT-05-A와 동일	메일 발송 성공(실메일 수신 또는 NotificationLog <code>EMAIL/SENT</code>)	실메일 수신/또는 DB에서 <code>EMAIL/SENT</code> 확인	Pass
IT-05-C	Slack 발송 성공 (WireMock)	IT-05-A와 동일	WireMock <code>/slack</code>	<code>_admin/requests/mappings</code> 확인	Pass
IT-05-D	logId 연계 일치	<code>POST id</code> ↔ 이벤트 <code>logId</code>	<code>logId == id</code>	일치 확인	Pass
IT-05-E	NotificationLog 기록	DB 조회	<code>EMAIL/SENT</code> & <code>SLACK/SENT</code>	최근 로그에 두 채널 <code>SENT</code> 존재	Pass

- 스텝

- Headers

- `Content-Type: application/json`

- POST Body

```
{
  "userId": "alert-ok@test.com",
  "sqlRaw": "DROP TABLE important_table",
  "sqlSummary": "drop table important_table",
  "returnRows": 0,
  "status": "SUCCESS"
}
```

- curl

```
# (A) 공격성 SQL → 이벤트 트리거
curl -s -X POST "http://localhost:8080/api/ingest/log" \
-H "Content-Type: application/json" \
-d '{"userId":"alert-ok@test.com","sqlRaw":"DROP TABLE important_table","sqlSummary":"drop table important_table","returnRows":0,"status":"SUCCESS"}'

# (B) PATTERN 이벤트 조회
curl -s "http://localhost:8080/api/events?type=PATTERN&user=alert-ok@test.com&size=5" | jq .

# (C) 첫 건 eventId 상세
EVENT_ID=<content[0].id>
curl -s "http://localhost:8080/api/events/$EVENT_ID" | jq .
```

- CLI Quick Check

- PATTERN/HIGH 존재

```
▶ curl -s "http://localhost:8080/api/events?type=PATTERN&user=alert-ok@test.com&size=5" \
| jq -e '.content[] | select(.eventType=="PATTERN" and .severity=="HIGH")' >/dev/null && echo OK || echo NG
OK
```

- POST id ↔ 이벤트 logId 매칭

```
▶ LOG_ID="ea50e18b-0fb7-4bb7-89ec-1069b58241c5"; \
curl -s "http://localhost:8080/api/events?type=PATTERN&user=alert-ok@test.com&size=5" \
| jq -e --arg L "$LOG_ID" ".content[] | select(.logId==$L)' >/dev/null && echo OK || echo NG
OK
```

- Slack(Webhook) 허트 ≥ 1 (WireMock)

```
▶ curl -s http://localhost:9090/_admin/requests/count \
-H 'Content-Type: application/json' \
--data '{"method":"POST","urlPath":"/slack"}' \
| jq -r '.count' | awk '{print ($1>=1) ? "OK" : "NG"}'
OK
```

- notification_log 확인

```
▶ sqlite3 ./backend/data/dbids.sqlite \
'select event_id, channel, status, sent_at from notification_log order by sent_at desc limit 10;'
fcc1bfc3-fffb-489c-9bf7-da1498475551|EMAIL|SENT|2025-10-30T09:05:20.911324Z
fcc1bfc3-fffb-489c-9bf7-da1498475551|SLACK|SENT|2025-10-30T09:05:20.911324Z
970cdb75-84ae-49c3-8561-a7518de5ed10|EMAIL|SENT|2025-10-30T08:48:07.099507Z
970cdb75-84ae-49c3-8561-a7518de5ed10|SLACK|SENT|2025-10-30T08:48:07.099507Z
```

- Evidence

From noreply@local.test
Subject [DB-IDS] [HIGH] Type=PATTERN User=alert-ok@test.com
To admin@local.test

Plain text Source

.DB-IDS Detection Alert

Event ID : fcc1bfc3-ffbb-489c-9bf7-da1498475551
Log ID : ea50e18b-0fb7-4bb7-89ec-1069b58241c5
Type : PATTERN
Severity : HIGH
Time : 2025-10-30T09:05:20.910669Z

User : alert-ok@test.com
Status : SUCCESS
Rows : 0

SQL Summary:
drop table important_table

SQL Raw:
DROP TABLE important_table

```
curl -s http://localhost:9090/_admin/mappings | jq . | head
{
  "mappings": [
    {
      "id": "b108bfa6-35d1-4e80-b393-d6be0fdebd7a",
      "request": {
        "urlPath": "/slack",
        "method": "POST"
      },
      "response": {
        "status": 200,
        "body": {
          "text": "Hello from Slack!"
        }
      }
    }
  ]
}
```

4.2.6. IT-06: Dashboard

- 모듈: `dashboard`(React+Vite), 백엔드 `com.example.dbids.api.*`
- 엔드포인트(UI):
 - `/logs` (로그 테이블), `/events` (이벤트 테이블), `stats` (요약 그래프)
- 검증 목적
 - 백엔드에 생성된 데이터가 대시보드에 정확히 반영되는지 확인
- 사전 조건
 - FE: `http://localhost:5173`
 - BE: `http://localhost:8080`
- 데이터 생성

```
PATTERN_ID=$(curl -s -X POST "http://localhost:8080/api/ingest/log" \
-H "Content-Type: application/json" \
-d '{"userId":"dash@test.com","sqlRaw":"DROP TABLE t_dash","sqlSummary":"drop table t_dash","returnRows":1}' | jq .id)
```

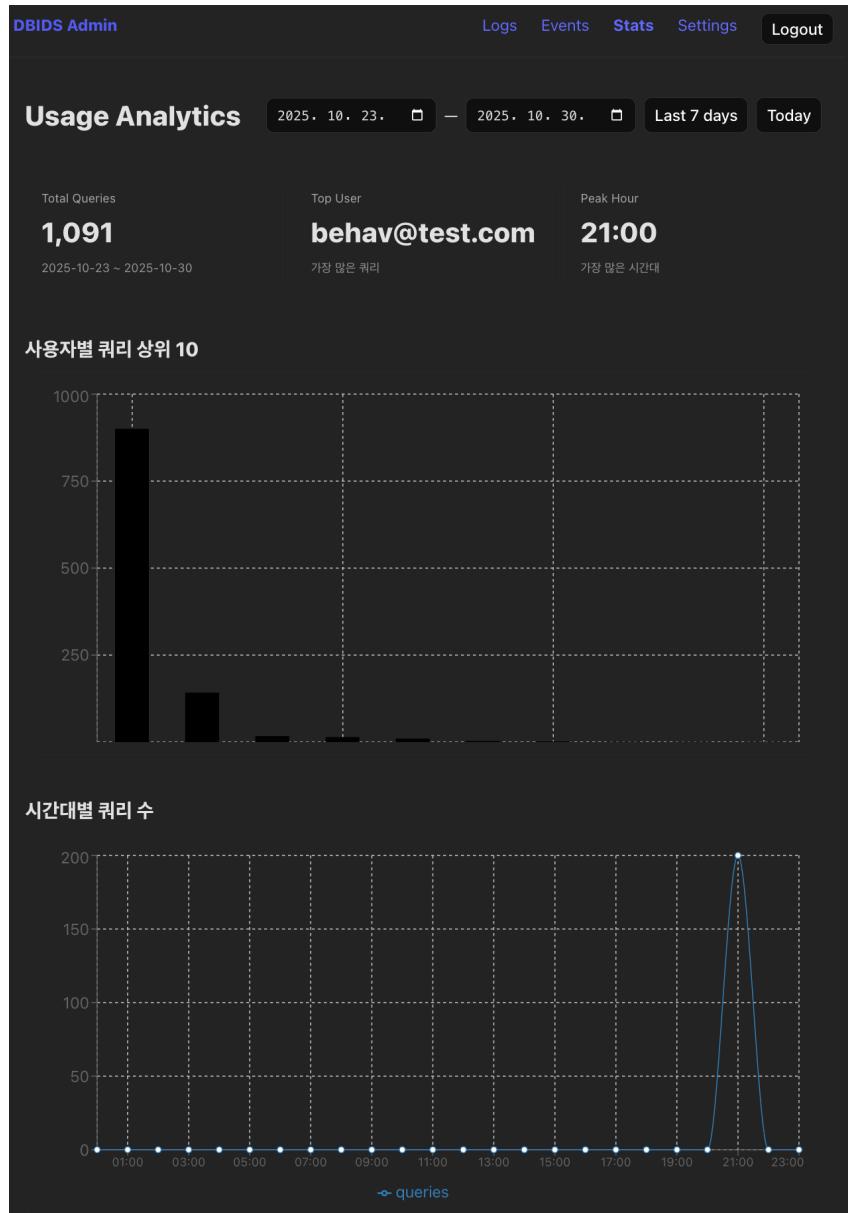
```
0,"status":"SUCCESS"}' \
| jq -r .id)
```

- 테스트 케이스 및 결과

TC ID	테스트명	입력(요약)	기대 결과	실제 결과(요약)	판정
IT-07-A	요약 지표 렌더링 (<code>/stats</code>)	<code>/stats</code> 접속	메트릭에 userId와 일치하는 표시	스크린샷 첨부	Pass
IT-07-B	이벤트 테이블 표시(<code>/events</code>)	<code>/events</code> 접속	최신행이 상단 정렬, 컬럼 (eventType, severity, userId, occurredAt, sqlPreview) 노출	스크린샷	Pass
IT-07-C	이벤트 필터 동작 (<code>/events</code>)	<code>/events?</code> type= PATTERN&user=dash@test.com	리스트에 eventType=PATTERN이고 user=dash@test.com 만 노출	스크린샷	Pass
IT-07-D	로그 테이블 표시 (<code>/logs</code>)	<code>/logs</code> 접속	컬럼 노출	스크린샷	Pass
IT-07-E	로그 필터 동작 (<code>/logs</code>)	<code>/logs?user=dash@test.com</code>	목록이 user=dash@test.com인 행만 표시	스크린샷	Pass
IT-07-F	자동 갱신 확인	<code>/events</code> 또는 <code>/stats</code> 열어둔 뒤 신규 POST	일정 주기 후 테이블/지표에 신규 데이터 반영	전/후 스크린샷	Pass

- 증빙

- `/stats`



o [/events](#)

DBIDS Admin Logs Events Stats Settings Logout

Detection Events — All

Refresh page 1 / 2

TIME	USER	TYPE	SEVERITY	SQL
2025. 10. 30. 오후 7:56:37	dash@test.com	PATTERN	HIGH	DROP TABLE T_DASH

◀ Prev Next ▶

o [/logs](#)

Query Logs				Logs	Events	Stats	Settings	Logout	
<input type="text" value="keywords (SQL 사용자 등)"/> <input type="text" value="user (또는 email)"/> <input type="checkbox"/> SUCCESS <input type="checkbox"/> FAILURE				<input type="button" value="Search"/>	<input type="button" value="Download"/>	<input type="checkbox"/> Auto refresh			
연도 . 월 . 일 .	연도 . 월 . 일 .	<input type="text" value="rowsMin"/>	<input type="text" value="rowsMax"/>			20page	page	Prev	Next
TIME ▾	USER	STATUS	ROWS	SQL (SUMMARY)			ACTION		
2025. 10. 30. 오전 7:56:37	dash@test.com	FAILURE	0	drop table t_dash			<input type="button" value="View"/>		

- ### ◦ 자동 갱신

DBS Admin

Logs Events Stats Settings Logout

Query Logs

keywords (SQL 시스템) \$ user (@이메일) SUCCESS FAILURE Search Download Auto refresh

모든 테이블 모든 테이블 rowsM1 rowsM2 20페이지 1/1 83 Next

TIME	USER	STATUS	ROWS	SQL (SUMMARY)	ACTION
2025. 10. 30. 오후 7:56:37	dash@test.com	FAILURE	0	drop table t_dash	View
2025. 10. 30. 오후 6:05:20	alert-ok@test.com	FAILURE	0	drop table important_table	View
2025. 10. 30. 오후 5:48:07	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:47:50	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:46:53	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:37:41	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:33:47	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:31:01	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 11:18:28	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 11:10:40	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 10:59:11	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 10:50:40	readonly@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 10:46:18	behave@test.com	SUCCESS	1	SELECT	View

FetchXhr 쿠션 CSS JS 브라우저 마지막 30초 30분 1시간 24시간 월주년

전체

/logs api가 주기적으로 실행된다.

4.3. 테스트 요약

TC ID	시나리오	판정	비고
IT-01	쿼리 로깅 통합	Pass	-
IT-02	패턴 기반 탐지	Pass	-
IT-03	행동 기반 탐지	Pass	-
IT-04	권한 기반 탐지	Pass	-
IT-05	알림 연동	Pass	-
IT-06	알림 장애	Fail	세부기능 미구현
IT-07	대시보드 연동	Pass	-
IT-08	로그 아카이브	Fail	기능 미구현
IT-09	로그-이벤트 일관성	Fail	기능 미구현

5. 시스템 테스트 (System Test)

5.1. 테스트 환경

- OS: Ubuntu 20.04
- JDK: Temurin 21
- 프레임워크: Spring Boot 3.3.2, React 19 / Vite 7
- 빌드 도구: Gradle 8.14
- DB: SQLite 3.45
- 도구: Gmail, Slack

5.2. 테스트 케이스 상세

5.2.1. ST-01: Admin → SELECT 실행 → QueryLog 저장 & 대시보드 조회

- 목적: FR-1 총족 여부 확인 — 수집 API에 SELECT 로그를 기록하면 `/logs` 조회 및 대시보드에 반영되는지 검증
- 환경/전제: `baseUrl=http://<EC2-Public-DNS>:8080`, Admin 계정 `st01-admin@test.com` 사용
- 절차:
 1. POST `/api/ingest/log`로 SELECT 기록 생성
 2. GET `/api/logs?user=...&size=10`으로 방금 생성한 로그 조회
 3. CLI Quick Check로 POST 응답 `id` 와 `/logs`의 `content[].id` 일치성 확인
 4. 대시보드 `/logs` 화면 캡처로 시각적 검증
- 판정 기준 대비
 1. POST 201/200 (5xx 아님) → 총족
 2. GET 200 OK → 총족
 3. 응답 본문 `content[]` 존재 및 `id` 일치 → 총족
 4. 대시보드 시각적 확인 → 총족
- 증빙

```

▶ BASE="http://ec2-52-78-120-100.ap-northeast-2.compute.amazonaws.com:8080"
USER="st01-admin@test.com"
NOW=$(date -u +"%Y-%m-%dT%H:%M:%S")
▶ Ko ➤
▶ curl -i -s -X POST "$BASE/api/ingest/log" \
-H "Content-Type: application/json" \
--data "{
  \"executedAt\": \"$NOW\",
  \"userId\": \"$USER\",
  \"sqlRaw\": \"SELECT * FROM users;\",
  \"sqlSummary\": \"SELECT users\",
  \"returnRows\": 3,
  \"status\": \"SUCCESS\"
}"

HTTP/1.1 201
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Location: /Logs/d76a92ca-35a8-4ffe-98cc-09d282d29d10
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Mon, 10 Nov 2025 07:33:16 GMT

{"id":"d76a92ca-35a8-4ffe-98cc-09d282d29d10"}%

```

POST `/api/ingest/log`로 SELECT 기록 생성

```

Ko ➜ curl -i -s "$BASE/api/logs?user=$USER&size=10"
HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Mon, 10 Nov 2025 07:33:45 GMT

{"totalPages":1,"totalElements":1,"first":true,"last":true,"pageable": {"pageNumber":0,"pageSize":10,"sort": {":sorted":true,"empty":false,"unsorted":false}, "offset":0,"paged":true,"unpaged":false}, "size":10, "content": [{"id": "d76a92ca-35a8-4fe-98cc-09d282d29d10", "executedAt": "2025-11-10T07:33:00Z", "userId": "st01-admin@test.com", "sqlSummary": "SELECT users", "sqlRaw": "SELECT * FROM users", "returnRows": 3, "status": "SUCCESS"}, {"number":0, "sort": {"sorted":true,"empty":false,"unsorted":false}, "numberOfElements":1, "empty":false}]}

```

GET /api/logs?user=...&size=10 으로 방금 생성한 로그 조회

```

▶ LOG_ID=$(curl -s -X POST "$BASE/api/ingest/log" \
-H "Content-Type: application/json" \
--data "{
  \"executedAt\": \"$NOW\",
  \"userId\": \"$USER\",
  \"sqlRaw\": \"SELECT 1\",
  \"sqlSummary\": \"SELECT 1\",
  \"returnRows\": 1,
  \"status\": \"SUCCESS\"
}" | jq -r '.id')

curl -s "$BASE/api/logs?user=$USER&size=10" \
| jq --arg L "$LOG_ID" '.content[] | select(.id==$L)' >/dev/null \
&& echo "ST-01 CHECK: LOG_ID linked -> PASS" || echo "ST-01 CHECK: NOT FOUND"

ST-01 CHECK: LOG_ID linked -> PASS

```

CLI Quick Check로 POST 응답 id 와 /logs 의 content[].id 일치성 확인

The screenshot shows a 'Query Logs' interface with the following details:

- Header:** keywords (SQL, 사용자 등), user (또는 email), SUCCESS, FAILURE, Search, Download, Auto refresh.
- Filter:** 연도, 월, 일, 연도, 월, 일, rowsMi, rowsMe, 20/page, Page 1/84, Next, Prev.
- Table Headers:** TIME, USER, STATUS, ROWS, SQL (SUMMARY), ACTION.
- Table Data:** 2025. 11. 10. 오후 4:33:08, st01-admin@test.com, SUCCESS, 3, SELECT users, Hide.
- SQL Summary:** SELECT users, Copy.
- SQL Raw:** SELECT * FROM users, Copy, Download .sql.

대시보드 /logs 화면 캡처로 시작적 검증

- 최종 판정 → Pass

5.2.2. ST-02: UNION SELECT 패턴 탐지 → PATTERN 이벤트 확인

- 목적: FR-2 충족 여부 확인 — 공격성 SQL(UNION SELECT)을 수집하면 Type=PATTERN 이벤트가 생성/조회되는지 검증
- 환경/전제: baseUrl=http://<EC2-Public-DNS>:8080, 테스트 사용자 st02-pattern@test.com
- 절차:
 - POST /api/ingest/log 로 UNION SELECT 포함 로그 생성
 - GET /api/events?type=PATTERN&user=...&size=50 로 방금 생성된 이벤트 조회
 - CLI Quick Check로 POST 응답 id와 이벤트의 logId 연계 및 타입/심각도 일치 확인
 - 대시보드 /events 화면 캡처로 시작적 검증
- 판정 기준 대비
 - POST 201/200 (5xx 아님) → 충족

2. GET 200 OK → 총족
 3. 응답 목록 중 `logId == POST id` 이며 `eventType="PATTERN"` → 총족
 4. `severity="MEDIUM"` 으로 표기(현재 정책 기준) → 총족
 5. 대시보드에서 동일 이벤트 확인 → 총족
- 증빙

```
[ST-02] POST /api/ingest/log (pattern attack)
HTTP/1.1 201
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Location: /logs/d897c184-841c-4198-9835-8e11f1814d07
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Mon, 10 Nov 2025 07:57:09 GMT
{"id": "d897c184-841c-4198-9835-8e11f1814d07"}
[ST-02] LOG_ID: d897c184-841c-4198-9835-8e11f1814d07
```

`POST /api/ingest/log` 로 UNION SELECT 포함 로그 생성

```
[ST-02] GET /api/events?type=PATTERN&user=st02-pattern@test.com&size=50
{
  "totalElements": 3,
  "content": [
    {
      "id": "adcfe61a-592a-44ab-9499-9a7f96a3317f",
      "logId": "d897c184-841c-4198-9835-8e11f1814d07",
      "eventType": "PATTERN",
      "severity": "MEDIUM",
      "occurredAt": "2025-11-10T07:57:02.077208543Z",
      "userId": "st02-pattern@test.com",
      "adminId": null,
      "sqlPreview": "SELECT ID FROM USERS UNION SELECT PASSWORD FROM USERS"
    },
    {
      "id": "37dd5c08-14fd-428c-a29b-1d9b5bad2daf",
      "logId": "c5bc8796-c2d8-4b63-8155-7b7cc36cba6c",
      "eventType": "PATTERN",
      "severity": "MEDIUM",
      "occurredAt": "2025-11-10T07:54:09.677356750Z",
      "userId": "st02-pattern@test.com",
      "adminId": null,
      "sqlPreview": "SELECT ID FROM USERS UNION SELECT PASSWORD FROM USERS"
    },
    {
      "id": "6dde7f4f-cfb0-4b7d-b596-49da0ea64d22",
      "logId": "d2804e6b-3065-43c1-9a64-709e45ef5e1e",
      "eventType": "PATTERN",
      "severity": "MEDIUM",
      "occurredAt": "2025-11-10T07:53:13.468600990Z",
      "userId": "st02-pattern@test.com",
      "adminId": null,
      "sqlPreview": "SELECT ID FROM USERS UNION SELECT PASSWORD FROM USERS"
    }
  ]
}
```

`GET /api/events?type=PATTERN&user=...&size=50` 로 방금 생성된 이벤트 조회

```

QC1: POST status -> OK
QC1b: LOG_ID parsed -> OK
QC2: PATTERN/HIGH for LOG_ID -> OK
QC3: sqlPreview contains UNION -> OK
QC3b: sqlPreview contains PASSWORD -> OK

```

CLI Quick Check로 POST 응답 id와 이벤트의 logId 연계 및 타입/심각도 일치 확인

2025. 11. 10. 오	st02-pattern@test.com	PATTERN	MEDIUM	SELECT ID FROM USERS UNION SELECT PASSWORD FROM USERS
후 4:57:02				

대시보드 /events 화면 캡처로 시작적 검증

- 최종 판정 → Pass

5.2.3. ST-03: 대량 SELECT 스파이크 → BEHAVIOR 이벤트(HIGH)

- 목적: FR-3 충족 여부 — 동일 사용자 고빈도 트래픽 발생 시 Type=BEHAVIOR, severity=HIGH 이벤트가 생성/조회되는지 검증
- 환경/전제: baseUrl=http://<EC2-Public-DNS:8080>, 사용자 st03-behav@test.com
- 절차:

- executedAt=T0 고정으로 다건 POST /api/ingest/log 전송(200건, 병렬 5)
- 윈도우 경계(T0+~60s) 넘긴 뒤 더미 1건 POST로 flush 유도
- GET /api/events?type=BEHAVIOR&user=...&size=20 조회
- CLI Quick Check로 BEHAVIOR/HIGH, QPM 임계(≥100) 검증
- 대시보드 /events 화면 캡처로 시작적 검증

- 판정 기준 대비:

- 대량 POST 중 5xx 없음 → 충족
- GET 200 OK 및 이벤트 최소 1건 → 충족
- eventType=="BEHAVIOR" AND severity=="HIGH" → 충족
- sqlPreview 내 QPM ≥ 100 → 충족

- 증빙

```

[ST-03] windowSeconds=60, T0=2025-11-10T08:37:22Z, T1='(sleep 사용)'
[ST-03] sending 200 logs in parallel (5) at executedAt=T0
[ST-03] sleeping 65s to cross window...
[ST-03] posted 1 dummy log at NOW=2025-11-10T08:38:31Z → flush expected
[ST-03] events summary (top 3):
{
    "totalElements": 1,
    "content": [
        {
            "id": "e2c4bd6e-afde-4bac-bcfc-06ca9553e377",
            "logId": "92f27dd9-e53e-41a3-a12f-8235fa34e630",
            "eventType": "BEHAVIOR",
            "severity": "HIGH",
            "occurredAt": "2025-11-10T08:38:31.404126528Z",
            "userId": "st03-behav@test.com",
            "adminId": null,
            "sqlPreview": "USER=st03-behav@test.com QPM=167.00 WRITE_RATIO=0.00 DDL/m=0.00 ERR=0 SCORE=167.00"
        },
        null,
        null
    ]
}

```

executedAt=T0 고정으로 다건 POST /api/ingest/log 전송(200건, 병렬 5)

윈도우 경계(T0+~60s) 넘긴 뒤 더미 1건 POST로 flush 유도

GET /api/events?type=BEHAVIOR&user=...&size=20 조회

```

},
QC1: BEHAVIOR/HIGH -> OK
QC2: QPM >= 100 -> OK
EVENT_ID=e2c4bd6e-afde-4bac-bcfc-06ca9553e377

```

CLI Quick Check로 BEHAVIOR/HIGH, QPM 임계(≥100) 검증

Detection Events — All					
TIME	USER	TYPE	SEVERITY	SQL	
2025. 11. 10. 오 후 5:38:31	st03-behav@test.com	BEHAVIOR	HIGH	USER=st03-behav@test.com QPM=167.00 WRITE_RATIO=0.00 DDL/m=0.00 ERR=0 SCORE=167.00	

대시보드 [/events](#) 화면 캡처로 시작적 검증

- 최종 판정 → Pass

5.2.4. ST-04: READ_ONLY → 금지 쓰기 시도 → AUTHZ 이벤트 생성/조회

- 목적: FR-4 총족 여부 확인 — READ_ONLY 사용자로 금지된 쓰기(INSERT/DELETE) 시 Type=AUTHZ 이벤트가 생성되고, severity=HIGH로 기록되는지 검증
- 환경/전제: baseUrl=http://<EC2-Public-DNS>:8080, 사용자 st04-authz@test.com 은 READ_ONLY로 가정
- 절차:
 - POST /api/ingest/log 로 금지 쓰기(INSERT) 시도
 - GET /api/events?type=AUTHZ&user=...&size=30 로 방금 로그와 매칭되는 AUTHZ 이벤트 조회
 - CLI Quick Check로 POST id == event.logId, eventType=AUTHZ, severity=HIGH, sqlPreview% INSERT/DELETE 포함 확인
- 판정 기준 대비
 - POST 201/200 (5xx 아님) → 총족
 - AUTHZ 이벤트 목록 200 OK → 총족
 - eventType=="AUTHZ" AND severity=="HIGH" → 총족
 - event.logId == POST 응답 id 일치 → 총족
- 증빙

```
[ST-04] AUTHZ test start (user=st04-authz@test.com)
[ST-04] POST response (headers+body)
HTTP/1.1 201
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Location: /logs/979b3832-e278-48be-a0fa-b3945dc05df5
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 11 Nov 2025 08:27:55 GMT

["id":"979b3832-e278-48be-a0fa-b3945dc05df5"]
[ST-04] LOG_ID=979b3832-e278-48be-a0fa-b3945dc05df5
```

POST /api/ingest/log 로 금지 쓰기(INSERT) 시도

```
[ST-04] events summary (top 3)
{
  "totalElements": 4,
  "content": [
    {
      "id": "fbbed749-c449-4975-bcc7-e3b1bc20edb1",
      "logId": "979b3832-e278-48be-a0fa-b3945dc05df5",
      "eventType": "AUTHZ",
      "severity": "HIGH",
      "occurredAt": "2025-11-11T08:27:47.365413397Z",
      "userId": "st04-authz@test.com",
      "adminId": null,
      "sqlPreview": "INSERT INTO USERS(ID) VALUES (0)"
    },
    {
      "id": "7b025ba3-cd2a-4072-b7c2-3dc2567d7d69",
      "logId": "5a8a1690-639a-4bb4-aab3-dcab564f9771",
      "eventType": "AUTHZ",
      "severity": "HIGH",
      "occurredAt": "2025-11-11T08:26:05.051900036Z",
      "userId": "st04-authz@test.com",
      "adminId": null,
      "sqlPreview": "INSERT INTO USERS(ID) VALUES (0)"
    },
    {
      "id": "9be0a4e3-b82b-4c29-8592-135415ef5cf7",
      "logId": "e850877d-8569-4f57-8aa0-3a15ad5c171c",
      "eventType": "AUTHZ",
      "severity": "HIGH",
      "occurredAt": "2025-11-11T08:18:50.126329Z",
      "userId": "st04-authz@test.com",
      "adminId": null,
      "sqlPreview": "DELETE FROM IMPORTANT_TABLE WHERE ID = 0"
    }
  ]
}
```

GET /api/events?type=AUTHZ&user=...&size=30 로 방금 로그와 매칭되는 AUTHZ 이벤트 조회

```
QC1: POST status -> OK
QC1b: LOG_ID parsed -> OK
QC2: AUTHZ/HIGH for LOG_ID -> OK
QC3: sqlPreview contains write op -> OK
```

CLI Quick Check로 POST id == event.logId , eventType=AUTHZ , severity=HIGH , sqlPreview에 INSERT/DELETE 포함 확인

- 최종 판정 → Pass

5.2.6. ST-06: Admin 로그인 후 이벤트 조회 → UI·DB 값 일치 검증

- 목적: FR-6 충족 여부 확인 — 관리자가 로그인한 뒤 대시보드에서 조회한 내용이 백엔드 DB 및 API 응답과 동일하게 반영되는지 검증
- 환경/전제: baseUrl=http://<EC2-Public-DNS>:8080 , 계정 st06-ui-db@test.com
- 절차

1. 공격 유도(로컬→EC2 API)

- POST /api/ingest/log (DROP TABLE) → 201 Created
- 추출값: LOG_ID=8b441b7c-8d56-44da-9a3f-ce1da550d1da

2. 이벤트 조회(API)

- GET /api/events?type=PATTERN&user=st06-ui-db@test.com&size=5
- EVENT_ID=14a219e2-ee79-453d-b82b-940878f29041 , eventType=PATTERN , severity=HIGH 확인

3. EC2 내부 DB 직접 확인

- SELECT ... FROM detection_event WHERE log_id='<LOG_ID>' → 1건, PATTERN/HIGH
- SELECT ... FROM query_log WHERE id='<LOG_ID>' → status=FAILURE , SQL 미리보기 DROP TABLE important_table...

4. UI 시각 확인(대시보드 캡처)

- `/logs` 화면: 해당 사용자 행에 FAILURE, `drop table important_table` 표시
- `/events` 화면: TYPE=BEHAVIOR/PATTERN 중 PATTERN/HIGH 행 노출(이번 건은 PATTERN/HIGH 확인)

• 판정 기준 대비

1. API POST 201/200 (5xx 아님) → 총족
2. 이벤트 API 200, `PATTERN/HIGH` 포함 → 총족
3. DB의 `detection_event` / `query_log` 값이 API와 동일 → 총족
4. UI(`/logs`, `/events`) 표시가 DB·API와 일치 → 총족

• 증빙

```
[ST-06] 1) PATTERN/HIGH 유도 : DROP TABLE
HTTP/1.1 201
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Location: /logs/8b441b7c-8d56-44da-9a3f-ce1da550d1da
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 11 Nov 2025 08:40:38 GMT

[ST-06] LOG_ID=8b441b7c-8d56-44da-9a3f-ce1da550d1da
[ST-06] EVENT_ID=14a219e2-ee79-453d-b82b-940878f29041 (PATTERN/HIGH)
201 POST → OK

POST /api/ingest/log (DROP TABLE) → 201 Created
GET /api/events?type=PATTERN&user=st06-ui-db@test.com&size=5
```

[ST-06] detection_event: 병글 LOG_ID에 해당하는 이벤트				
id	log_id	event_type	severity	occurred_at
14a219e2-ee79-453d-b82b-940878f29041	8b441b7c-8d56-44da-9a3f-ce1da550d1da	PATTERN	HIGH	2025-11-11T08:40:38.819192941Z
[ST-06] query_log: 해당 LOG_ID 상세				
id	user_id	status	executed_at	sql_preview
8b441b7c-8d56-44da-9a3f-ce1da550d1da	st06-ui-db@test.com	FAILURE	2025-11-11T08:40:38.807469402Z	DROP TABLE important_table...

SELECT ... FROM detection_event WHERE log_id='<LOG_ID>' → 1건, PATTERN/HIGH

SELECT ... FROM query_log WHERE id='<LOG_ID>' → [status=FAILURE, SQL 미리보기] DROP TABLE important_table...

2025. 11. 11. 오후 5:40:30	st06-ui-db@test.com	FAILURE	0	drop table important_table	View
-----------------------------	---------------------	---------	---	----------------------------	----------------------

`/logs` 화면: 해당 사용자 행에 FAILURE, `drop table important_table` 표시

2025. 11. 11. 오후 후 5:40:30	st06-ui-db@test.com	PATTERN	HIGH	DROP TABLE IMPORTANT_TABLE
-------------------------------	---------------------	---------	------	----------------------------

`/events` 화면: TYPE=BEHAVIOR/PATTERN 중 PATTERN/HIGH 행 노출

- 최종 판정 → Pass

5.3. 테스트 요약

TC ID	시나리오	판정	비고
ST-01	Admin → SELECT 실행 → QueryLog 저장 & 대시보드 조회	Pass	-

TC ID	시나리오	판정	비고
ST-02	UNION SELECT password 공격 수집 → PATTERN 이벤트 생성	Pass	-
ST-03	동일 사용자 대량 트래픽(원도우 경계 flush) → BEHAVIOR/HIGH	Pass	-
ST-04	READ_ONLY 사용자 쓰기 시도(INSERT/DELETE) → AUTHZ/HIGH	Pass	-
ST-05	HIGH 이벤트 발생 시 Email 발송 성공	Pass	-
ST-06	로그인 후 /logs · /events UI 값 ↔ DB 값 일치 검증	Pass	-
ST-07	로그인 실패 시 오류 메시지 표시	Fail	기능 미구현
ST-08	30일 경과 로그 아카이브(S3 업로드 및 보관)	Fail	기능 미구현
ST-09	로그-이벤트 일관성(POST id = event.logId, 상태 보정 일치)	Fail	기능 미구현

6. 비기능 테스트 (Non-Functional Test)

6.1. 테스트 환경

- OS: Ubuntu 20.04
- JDK: Temurin 21
- 프레임워크: Spring Boot 3.3.2, React 19 / Vite 7
- 빌드 도구: Gradle 8.14
- DB: SQLite 3.45
- 도구: JMeter

6.2. 테스트 케이스 상세

6.2.1. NFT-01: 탐지 지연

- 테스트 설계
 - 부하 모델: 경량 동시성(Threads=3), 짧은 폴링(0ms×2회 → 10~25ms, 최대 40회).
 - 측정 대상: POST /api/ingest/log (로그 생성) + GET /api/logs?page=0&size=20&sort=executedAt,DESC (최신 목록) 응답시간.
 - 지표 산정: 샘플러 응답시간(Average/95p)과 오류율을 기준으로 판정.
- 시나리오 흐름
 1. POST /api/ingest/log 로 로그 생성(유니크 ID).
 2. GET /api/logs?page=0&size=20&sort=executedAt,DESC 를 짧은 간격으로 폴링.
 3. 응답 목록에 방금 생성된 logId가 보이는 시점을 감시.
- 판정 기준
 - 평균 응답시간 ≤ 1,000 ms
 - 95퍼센타일 ≤ 1,000 ms
 - 오류율 < 5%
- 측정 지표 (Aggregate Report)

구분	#Samples	평균(ms)	95p(ms)	최대(ms)	오류율	처리량
A-POST /api/ingest/log	304	193	453	1032	1.64%	36.6/sec
A-GET /api/logs	1,317	197	420	1481	3.34%	159.6/sec
TOTAL	1,621	197	431	1481	3.02%	195.3/sec

판정

- 평균/95p 모두 기준 만족, 오류율 < 5% → NFT-01: Pass

6.2.2. NFT-02: 처리량

- 테스트 설계
 - 목적: API 수집 경로(POST /api/ingest/log)의 지속 처리량 검증.
 - 부하 모델: Threads=10, Ramp-up=10s, Hold=60s 이상.
 - 측정 대상: POST /api/ingest/log 응답(성공률/지연), JMeter Aggregate Report 처리량(Throughput).
- 시나리오 흐름
 1. 클라이언트가 초당 일정 속도로 POST /api/ingest/log 요청 전송
 2. 서버가 QueryLog 기록 후 2xx 응답
 3. JMeter가 처리량(ops/sec), 오류율, 지연분포를 집계
- 판정 기준
 - 지속 처리량 $\geq 100/\text{sec}$
 - 오류율 < 5%
- 측정 지표 (Aggregate Report)

구분	#Samples	평균(ms)	95p(ms)	최대(ms)	오류율	처리량
B-POST /api/ingest/log	6,041	448	923	3,566	0.83%	102.0/sec
TOTAL	6,041	448	923	3,566	0.83%	102.0/sec

- 판정
 - 지속 처리량 102/sec, 오류율 0.83% → NFT-02: Pass

6.2.3. NFT-03: 동시성

- 테스트 설계
 - 목적: 대시보드 API(GET /api/logs?page=0&size=20&sort=executedAt,DESC)의 동시 100명 접속 시 응답성 검증
 - 부하 모델: Threads=100, Ramp-up=10s, Loop=1 (100명 1회 동시 진입)
 - 측정 대상: 응답 코드/응답 시간(Avg, p95, Max), 오류율, 처리량(Throughput)
- 시나리오 흐름
 1. 100명의 가상사용자가 동시에 /api/logs 호출
 2. 서버가 200 OK로 최신 로그 페이지 반환
 3. JMeter가 응답 시간·분포·오류율·처리량 집계
- 판정 기준
 - Response Code = 200
 - Response Time $\leq 2,000 \text{ ms}$ (모든 샘플)
- 측정 지표 (Aggregate Report)

구분	#Samples	평균(ms)	95p(ms)	최대(ms)	오류율	처리량
C-GET /api/logs	100	67	182	245	0.00%	20.1/sec
TOTAL	100	67	182	245	0.00%	20.1/sec

- 판정
 - 전 샘플 2초 이내, 오류율 0% → NFT-03: Pass

6.2.4. NFT-06: 보안(비밀번호 저장)

- 테스트 설계
 - 목적: Admin 계정 비밀번호가 SHA-256 해시로 저장되는지 확인
 - 대상 테이블/컬럼: `admin_user` / `password_hash`
 - 판정 기준: 테이블에 평문 비밀번호가 존재하지 않으며, 저장 값이 SHA-256 해시 형태(16진 64자) 임을 증빙
- 시나리오 흐름
 1. DB에서 `admin_user` 테이블 조회
 2. `password_hash` 컬럼이 평문이 아닌 해시로 저장됐는지 확인
 3. 화면 캡처로 증빙 확보
- 증빙

id	email	password_hash	role	last_login	slack_webhook
18	martin114456@gmail.com	792d23ac883d593da9c163797fd27964eb2c53eaef8bd8cb2b133...	DBA	2025-11-11T06:42:28.424Z	https://hooks.slack.com/s...

- 판정
 - 저장 값이 평문이 아니고 SHA-256 해시 형태임을 캡처로 확인 → NFT-06: Pass

6.2.7. NFT-07: 신뢰성

- 테스트 설계
 - 목적: 정상 트래픽과 공격 트래픽에 대한 오탐(FPR) / 미탐(FNR) 검증.
 - 방법: `curl + jq` 스크립트/원라이너로 API 호출 및 이벤트 조회.
 - 데이터:
 - 정상 1,000건: 단순 `SELECT` 류, 50ms 페이싱으로 행동탐지 윈도우 회피.
 - 공격 50건: `DROP TABLE ...` 로 명시적 패턴 키워드 사용.
- 시나리오 흐름:
 1. 정상 1,000건을 50ms 간격으로 적재
 2. 60초 쿨다운 후 공격 50건 적재
 3. `eventType="PATTERN"` 이벤트 발생 여부로 탐지 판정
- 판정 기준
 - $FPR \leq 5\%$, $FNR \leq 2\%$
- 측정 결과

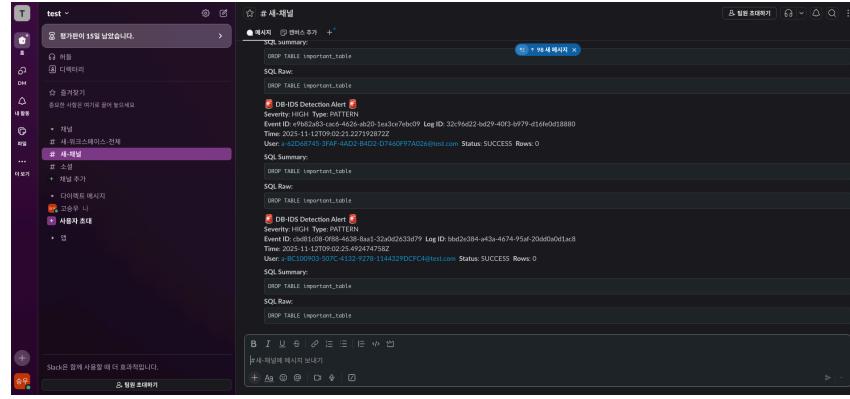
공격(50) → TP=50, FN=0 → FNR = 0.00% (기준 ≤ 2% 충족)
정상(1000) 표본(100) → FP=0 → FPR≈0% (기준 ≤ 5% 충족)

- 판정
 - 오탐률과 미탐률 기준 모두 충족 → Pass

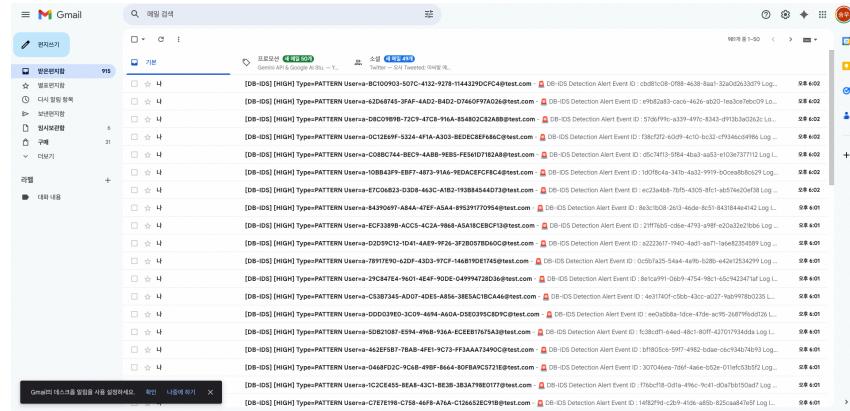
6.2.8. NFT-08: 신뢰성

- 테스트 설계
 - 목적: 이벤트 발생 시 알림(이메일/Slack) 발송 성공률을 검증.
 - 방법: `curl + jq` 스크립트/원라이너로 API 호출 및 이벤트 조회 후 각 이벤트 알림 스크린샷.

- 데이터: 공격 100건 (`DROP TABLE important_table`)로 PATTERN 이벤트 유도.
- 시나리오 흐름
 - `DROP TABLE ...` 기반 공격 100건 생성 → PATTERN 이벤트 100건 발생
 - 스크린샷을 통해 알림 확인
- 판정 기준
 - 알림 성공률 ≥ 99%
- 측정 결과



슬랙 메시지 100개 전송



- 판정
 - 슬랙과 이메일 모두 전송 → Pass

6.3. 테스트 요약

TC ID	항목	시나리오	예상 결과	실제 결과	판정
NFT-01	성능(탐지 지연)	1000건/초 SELECT	탐지 ≤ 1초	Pass	-
NFT-02	성능(처리량)	1000TPS 부하	TPS ≥ 1000 유지	Pass	-
NFT-03	성능(동시성)	Dashboard 100명 동시 접속	응답 ≤ 2초	Pass	-
NFT-04	보안(SQL Injection)	패턴, 행동 기반 Injection	탐지 성공	Pass	ST-02, ST-03으로 대체
NFT-05	보안(권한 제어)	READ_ONLY → UPDATE	Event.Type=AUTHZ	Pass	ST-04으로 대체
NFT-06	보안(비밀번호 저장)	Admin PW 확인	SHA-256 해시 저장	Pass	-

TC ID	항목	시나리오	예상 결과	실제 결과	판정
NFT-07	신뢰성(탐지율)	정상1000, 공격50	오탐≤5%, 미탐≤2%	Pass	-
NFT-08	신뢰성(알림 성공률)	Event 100건	성공률 ≥ 99%	Pass	-

7. 요구사항 추적 매트릭스

FR	UT	IT	ST	NFT
FR-1	UT-11 (P)	IT-01 (P)	ST-01 (P)	NFT-02 (P)
FR-2	UT-03,04 (P/P)	IT-02 (P)	ST-02 (P)	NFT-04 (P)
FR-3	UT-05,06 (P/P)	IT-03 (P)	ST-03 (P)	NFT-01 (P)
FR-4	UT-07,08 (P/P)	IT-04 (P)	ST-04 (P)	NFT-05 (P)
FR-5	UT-09,10 (P/P)	IT-05,06 (P/P)	ST-05 (P)	NFT-08 (P)
FR-6	—	IT-07 (P)	ST-06,07 (P/P)	NFT-03 (P)
FR-7	UT-12 (P)	IT-08,09 (P/P)	ST-08,09 (P/P)	—

8. 주요 결함 (Defect Summary)

ID	유형	심각도	발생 TC	설명	조치
D-01	Log Backup & Archive 기능 미구현	High	UT-11, UT-12 IT-08, IT-09 ST-08, ST-09	기능 미구현	이후 구현
D-02	로그인 실패 알림 메시지 미구현	Medium	ST-7	세부 기능 미구현	이후 구현
D-03	알람 실패 미구현	Medium	IT-06	세부 기능 미구현	이후 구현

9. 첨부 자료

9.1 Unit Test 로그

9.1.1. AuthZEngineTest

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.authz.AuthZEngineTest" tests="3" skipped="0" failures="0" errors="0"
hostname="Ko-MacBookAir.local" time="0.015">
<properties/>
<testcase name="UT-08-C: DBA + DELETE → Violation 없음" classname="com.example.dbids.modules.authz.AuthZEn
gineTest" time="0.01"/>
<testcase name="UT-07-A: READ_ONLY + DELETE orders → Violation 반환, ruleMatched=DENY DELETE:*" classname
="com.example.dbids.modules.authz.AuthZEngineTest" time="0.002"/>
<testcase name="UT-07-B: 테이블 파싱 실패 시 *로 평가해도 규칙 매칭(DENY *) 동작" classname="com.example.dbids.mo
dules.authz.AuthZEngineTest" time="0.001"/>
</testsuite>
```

9.1.2. AuthZServiceTest

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.authz.AuthZServiceTest" tests="4" skipped="0" failures="0" errors="0"
hostname="Ko-MacBookAir.local" time="0.598">
<properties/>
<testcase name="UT-08-B: fail-open — 저장 중 예외 발생 시 Optional.empty 반환, 흐름 차단 없음" classname="com.exam
```

```

ple.dbids.modules.authz.AuthZServiceTest" time="0.58"/>
<testcase name="UT-08-A: UserRole 미지정 → 정책 비적용(허용) → 이벤트 없음" classname="com.example.dbids.module
s.authz.AuthZServiceTest" time="0.004"/>
<testcase name="UT-08: DBA 사용자의 DELETE → 이벤트 없음, 상태보정/알림/저장 호출 없음" classname="com.example.d
bids.modules.authz.AuthZServiceTest" time="0.003"/>
<testcase name="UT-07: READ_ONLY 사용자의 DELETE → AUTHZ 이벤트 생성 + 상태보정(FAILURE) + 알림" classname
="com.example.dbids.modules.authz.AuthZServiceTest" time="0.008"/>
</testsuite>

```

9.1.3. BehaviorDetectorTest

```

<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.behavior.BehaviorDetectorTest" tests="4" skipped="0" failures="0" err
ors="0" hostname="Ko-MacBookAir.local" time="0.043">
<properties/>
<testcase name="UT-05-B(경계 상회): 임계 이상 활동량 → 이벤트 발생" classname="com.example.dbids.modules.behavio
r.BehaviorDetectorTest" time="0.016"/>
<testcase name="UT-05-A(경계 하회): 낮은 활동량 → 이벤트 없음 (임계↑로 강제 비발생)" classname="com.example.dbids.
modules.behavior.BehaviorDetectorTest" time="0.005"/>
<testcase name="UT-06: 행 반환 수=100 (저활동) → 이벤트/알림 없음 (임계↑로 강제 비발생)" classname="com.example.db
ids.modules.behavior.BehaviorDetectorTest" time="0.001"/>
<testcase name="UT-05: 1초에 200개 SELECT → Score ≥ 0.8, BEHAVIOR 이벤트 + 알림" classname="com.example.dbid
s.modules.behavior.BehaviorDetectorTest" time="0.019"/>
</testsuite>

```

9.1.4. DetectionServiceTest

```

<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.detection.DetectionServiceTest" tests="5" skipped="0" failures="0" err
ors="0" hostname="Ko-MacBookAir.local" time="0.026">
<properties/>
<testcase name="UT-04-DS: 정상 SELECT → 이벤트 없음(저장/알림 호출 없음)" classname="com.example.dbids.module
s.detection.DetectionServiceTest" time="0.006"/>
<testcase name="UT-03-DS: OR 1=1 → PATTERN/MEDIUM 이벤트(정규화로 0=0) + 알림 호출" classname="com.example.d
bids.modules.detection.DetectionServiceTest" time="0.005"/>
<testcase name="UT-03-DS: SLEEP() → PATTERN/LOW 이벤트(상태 보정 없음) + 알림 호출" classname="com.example.d
bids.modules.detection.DetectionServiceTest" time="0.004"/>
<testcase name="UT-03-DS: UNION SELECT → PATTERN/MEDIUM 이벤트(상태 보정 없음) + 알림 호출" classname="co
m.example.dbids.modules.detection.DetectionServiceTest" time="0.004"/>
<testcase name="UT-03-DS: DROP TABLE → PATTERN/HIGH 이벤트 + QueryLog.status=FAILURE 보정 + 알림 호출" clas
sname="com.example.dbids.modules.detection.DetectionServiceTest" time="0.004"/>
</testsuite>

```

9.1.5. EmailSenderTest

```

<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.notify.EmailSenderTest" tests="4" skipped="0" failures="0" errors="0"
hostname="Ko-MacBookAir.local" time="0.026">
<properties/>
<testcase name="UT-10-B: EmailSender sendTo() - props.from 없으면 springMailUsername 폴백 사용" classname="co
m.example.dbids.modules.notify.EmailSenderTest" time="0.023"/>
<testcase name="UT-10-D: EmailSender send() - toDefault 누락 시 IllegalStateException" classname="com.example.d

```

```

bids.modules.notify.EmailSenderTest" time="0.001"/>
<testcase name="UT-10-C: EmailSender disabled=true → sendTo 호출해도 메일 전송 안 함" classname="com.example.d
bids.modules.notify.EmailSenderTest" time="0.0"/>
<testcase name="UT-10-A: EmailSender send() - toDefault로 전송, props.from 사용" classname="com.example.dbids.
modules.notify.EmailSenderTest" time="0.0"/>
</testsuite>

```

9.1.6. NotificationServiceTest

```

<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.notify.NotificationServiceTest" tests="5" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.089">
<properties/>
<testcase name="UT-10-C: Email 전송 실패 → EMAIL/FAILED 로그 저장 (예외 삼키고 계속 진행)" classname="com.example.dbids.modules.notify.NotificationServiceTest" time="0.075"/>
<testcase name="UT-09-B/UT-10-B: Email+Slack ON → EMAIL/SENT + SLACK/SENT 각 1건씩 저장" classname="com.example.dbids.modules.notify.NotificationServiceTest" time="0.004"/>
<testcase name="UT-10-D: Slack 전송 실패 → SLACK/FAILED 로그 저장 (예외 삼키고 계속 진행)" classname="com.example.dbids.modules.notify.NotificationServiceTest" time="0.002"/>
<testcase name="UT-09-A: Slack ON, Email OFF → SLACK/SENT 로그 1건 저장" classname="com.example.dbids.modules.notify.NotificationServiceTest" time="0.003"/>
<testcase name="UT-10-A: Email ON, Slack OFF → EMAIL/SENT 로그 1건 저장" classname="com.example.dbids.modules.notify.NotificationServiceTest" time="0.003"/>
</testsuite>

```

9.1.7. SlackSenderTest

```

<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.notify.SlackSenderTest" tests="4" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.219">
<properties/>
<testcase name="UT-09-C: SlackSender disabled=true → 호출 없이 조용히 리턴" classname="com.example.dbids.modules.notify.SlackSenderTest" time="0.149"/>
<testcase name="UT-09-B: SlackSender 500 응답 → RuntimeException(SLACK_WEBHOOK_HTTP_500)" classname="com.example.dbids.modules.notify.SlackSenderTest" time="0.058"/>
<testcase name="UT-09-D: SlackSender webhook-url 누락 → IllegalStateException" classname="com.example.dbids.modules.notify.SlackSenderTest" time="0.004"/>
<testcase name="UT-09-A: SlackSender 200 OK → 전송 성공(SENT), payload에 text 포함" classname="com.example.dbids.modules.notify.SlackSenderTest" time="0.006"/>
</testsuite>

```

9.1.8. ProxyRequestHandlerTest

```

<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.proxy.ProxyRequestHandlerTest" tests="2" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.004">
<properties/>
<testcase name="UT-01: 원문은 UTF-8로 추출되고 STATUS=SUCCESS" classname="com.example.dbids.modules.proxy.ProxyRequestHandlerTest" time="0.001"/>
<testcase name="UT-01: 비정상 UTF-8 바이트 시퀀스는 거부한다" classname="com.example.dbids.modules.proxy.Proxy

```

```
RequestHandlerTest" time="0.002"/>
</testsuite>
```

9.1.9. ProxyResponseHandlerTest

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.proxy.ProxyResponseHandlerTest" tests="2" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.104">
<properties/>
< testcase name="UT-02: ResultSet 10행 → RowCount=10, Status=SUCCESS" classname="com.example.dbids.modules.proxy.ProxyResponseHandlerTest" time="0.102"/>
< testcase name="UT-02: ResultSet 처리 중 예외 → RowCount=0, Status=FAILURE" classname="com.example.dbids.modules.proxy.ProxyResponseHandlerTest" time="0.002"/>
</testsuite>
```

9.1.10. RuleEngineTest

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.rule.RuleEngineTest" tests="6" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.004">
<properties/>
< testcase name="UT-03-RE: 복수 패턴 동시 매칭 시 가장 높은 severity를 선택한다" classname="com.example.dbids.modules.rule.RuleEngineTest" time="0.0"/>
< testcase name="UT-03-RE: UNION SELECT → 매칭 성공, severity=MEDIUM" classname="com.example.dbids.modules.rule.RuleEngineTest" time="0.0"/>
< testcase name="UT-03-RE: DROP TABLE → 매칭 성공, severity=HIGH" classname="com.example.dbids.modules.rule.RuleEngineTest" time="0.0"/>
< testcase name="UT-03-RE: OR 1=1 → 매칭 성공, severity=MEDIUM (공백/대소문자 변형 포함)" classname="com.example.dbids.modules.rule.RuleEngineTest" time="0.0"/>
< testcase name="UT-03-RE: SLEEP() → 매칭 성공, severity=LOW (시간 지연 유도 함수)" classname="com.example.dbids.modules.rule.RuleEngineTest" time="0.0"/>
< testcase name="UT-04-RE: 정상 SELECT → 매칭 없음" classname="com.example.dbids.modules.rule.RuleEngineTest" time="0.001"/>
</testsuite>
```

9.1.11. SqlNormalizerTest

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.rule.SqlNormalizerTest" tests="3" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.002">
<properties/>
< testcase name="UT-03-RE/UT-04-RE 정규화: 블록/라인 주석 제거, 공백 정리, 대문자화" classname="com.example.dbids.modules.rule.SqlNormalizerTest" time="0.0"/>
< testcase name="UT-03-RE/UT-04-RE: 앞뒤 공백 제거 및 단일 공백화" classname="com.example.dbids.modules.rule.SqlNormalizerTest" time="0.0"/>
< testcase name="UT-03-RE/UT-04-RE 정규화: 문자열/숫자 리터럴 마스킹" classname="com.example.dbids.modules.rule.SqlNormalizerTest" time="0.001"/>
</testsuite>
```

9.1.12. StorageServiceTest

```

<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.storage.StorageServiceTest" tests="7" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.042">
    <properties/>
    < testcase name="UT-11: 잘못된 status → invalid status" classname="com.example.dbids.modules.storage.StorageServiceTest" time="0.031"/>
    < testcase name="UT-11: 시간 포맷 오류 → invalid executedAt" classname="com.example.dbids.modules.storage.StorageServiceTest" time="0.001"/>
    < testcase name="UT-11: adminId가 DB에 없으면 adminId not found" classname="com.example.dbids.modules.storage.StorageServiceTest" time="0.001"/>
    < testcase name="UT-11: 정상 insert → repo.save 1회, id 반환" classname="com.example.dbids.modules.storage.StorageServiceTest" time="0.001"/>
    < testcase name="UT-11: sqlSummary 512 초과는 절단 저장" classname="com.example.dbids.modules.storage.StorageServiceTest" time="0.002"/>
    < testcase name="UT-11: 필수 날짜/음수/빈값 → IllegalArgumentException" classname="com.example.dbids.modules.storage.StorageServiceTest" time="0.001"/>
    < testcase name="UT-11: sqlRaw 8KB 초과는 절단 저장" classname="com.example.dbids.modules.storage.StorageServiceTest" time="0.002"/>
</testsuite>

```

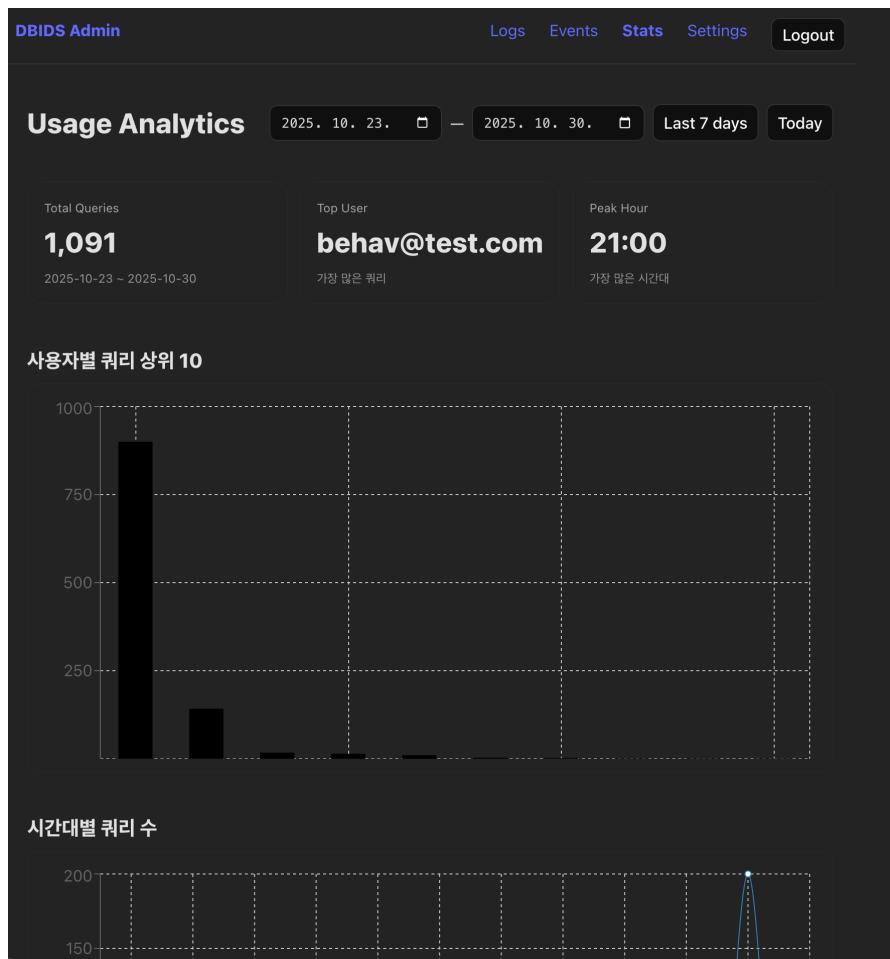
9.2. DashBoard 스크린샷

The screenshot shows the 'Query Logs' section of the DBIDS Admin interface. It displays a table of log entries with the following columns: TIME, USER, STATUS, ROWS, SQL (SUMMARY), and ACTION. The table lists various database operations such as 'drop table', 'INSERT users', and 'SELECT'. Most entries show a 'FAILURE' status, while one entry for 'behav@test.com' shows a 'SUCCESS' status.

TIME	USER	STATUS	ROWS	SQL (SUMMARY)	ACTION
2025. 10. 30. 오후 7:56:37	dash@test.com	FAILURE	0	drop table t_dash	View
2025. 10. 30. 오후 6:05:20	alert-ok@test.com	FAILURE	0	drop table important_table	View
2025. 10. 30. 오후 5:48:07	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:47:50	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:46:53	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:37:41	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:33:47	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 5:31:01	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 11:18:28	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 11:10:40	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 10:59:11	authz@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 10:50:40	readonly@test.com	FAILURE	0	INSERT users	View
2025. 10. 30. 오후 10:46:18	behav@test.com	SUCCESS	1	SELECT	View

Detection Events — All

TIME	USER	TYPE	SEVERITY	SQL
2025.10.30. 9:56:37	dash@test.com	PATTERN	HIGH	DROP TABLE T_DASH
2025.10.30. 9:56:20	alert-ok@test.com	PATTERN	HIGH	DROP TABLE IMPORTANT_TABLE
2025.10.30. 9:56:07	authz@test.com	BEHAVIOR	HIGH	USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ER0=1 SCORE=2.50
2025.10.30. 9:56:50	authz@test.com	BEHAVIOR	HIGH	USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ER0=1 SCORE=2.50
2025.10.30. 9:56:41	-	AUTHZ	HIGH	INSERT INTO users(id) VALUES ()
2025.10.30. 9:56:53	authz@test.com	BEHAVIOR	HIGH	USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ER0=1 SCORE=2.50
2025.10.30. 9:57:41	authz@test.com	BEHAVIOR	HIGH	USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ER0=1 SCORE=2.50
2025.10.30. 9:57:31	authz@test.com	BEHAVIOR	HIGH	USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ER0=1 SCORE=2.50
2025.10.30. 9:57:10	behav@test.com	BEHAVIOR	HIGH	USER=behav@test.com QPM=500.00 WRITE_RATIO=0.00 DDL/m=0.00 ER0=0 SCORE=500.00
2025.10.29. 9:55:25	behav@test.com	BEHAVIOR	HIGH	USER=behav@test.com QPM=200.00 WRITE_RATIO=0.00 DDL/m=0.00 ER0=0 SCORE=200.00
2025.10.29. 9:30:48	martin115@example.com	BEHAVIOR	MEDIUM	USER=martin115@example.com QPM=2.00 WRITE_RATIO=0.00 DDL/m=2.00 ER0=2 SCORE=5.00
2025.10.29. 9:30:48	martin115@example.com	PATTERN	HIGH	DROP TABLE SECRET_DATA
2025.10.29. 8:39:32	martin115@example.com	PATTERN	HIGH	DROP TABLE SECRET_DATA
2025.10.29. 8:39:14	martin115@example.com	PATTERN	HIGH	DROP TABLE SECRET_DATA
2025.10.29. 8:11:57	martin115@example.com	PATTERN	HIGH	DROP TABLE SECRET_DATA
2025.10.29. 8:04:14	martin115@example.com	PATTERN	HIGH	DROP TABLE SECRET_DATA



DBIDS Admin Logs Events Stats **Settings** [Logout](#)

Alert Channels

이메일 또는 Slack Webhook 중 하나만 입력해도 저장됩니다. (둘 다 입력도 가능)

Email (선택)
martin114456@gmail.com

Slack Webhook URL (선택)
https://hooks.slack.com/services/...

[저장하기](#)

관리자 로그인

Email
admin@example.com

비밀번호
.....

[로그인](#)

계정이 없나요? [회원가입](#)

* 5회 실패 시 계정 잠금. HTTPS에서만 사용하세요.