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	0	9	0%
Non-Functional	8	0	9	0%

> System Test 및 Non-Functional 테스트는 시간의 이유로 진행하지 못함

3. Unit 테스트

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-8을 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 `asssertEquals` 통과 여부
- 테스트 케이스 및 결과

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

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정
		트			
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=SUCCESS`를 설정하는지,
  - 처리 중 예외 발생 시 `rowCount=0` 및 `status=FAILURE`로 안전하게 귀결하는지 확인
- 사전 조건:
  - `ResultSet#next()` 동작을 목 객체로 시뮬레이션
  - 판정 근거는 테스트 코드의 `asssertEquals` 통과 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` 10회 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` 통과 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

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정
UT-03-RE-D	SLEEP() → 매칭 성공, severity=LOW	`... OR SLEEP(5)`	매칭 있음, `severity=LOW`, 정규화 결과에 `SLEEP(` 포함	동일	Pass
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`	동잉ㄹ	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, `Type=PATTERN`, `Severity=HIGH`, `ev.sqlRaw == normalize(sql)`, `QueryLog.status=FAILURE`로 보정 및 `logRepo.save()` 1회, `notifier.onEvent()` 호출	전부 일치(정규화 결과에 `DROP TABLE`, 주석 제거 확인)	Pass (`assertEquals/True` + `verify(save, times(1))`, `verify(onEvent, times(1))`)
UT-03-DS-B	UNION SELECT → PATTERN/MEDIUM (보정 없음) + 알림	`... UNION SELECT ...`	이벤트 생성, `Severity=MEDIUM`, 상태 보정/저장 없음, 알림 호출	전부 일치 (`logRepo.save()` 호출 0)	Pass (`assert*` + `verify(never())/times(1)`)
UT-03-DS-C	OR 1=1 → PATTERN/MEDIUM (정규화로 0=0) + 알림	`... OR 1=1 -- bypass`	이벤트 생성, `Severity=MEDIUM`, 정규화 결과에 `OR 0=0` 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, `Severity=LOW`, 정규화 결과에 `SLEEP(` 포함, 보정 없음, 알림 호출	전부 일치	Pass
UT-04-DS-A	정상 SELECT → 이벤트/저장/알림 없음	`SELECT name FROM users;`	이벤트 없음 (`eventId.isEmpty()`), 레포/	전부 일치	Pass (`verify(never())` 일괄 확인)

TC ID	테스트명 (@DisplayName)	입력(요약)	기대 결과	실제 결과	판정/근거
			알림 호출 없음, 원상태 유지		

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 `Instant` (t0)에서 200건, +2초 후 flush 1건	`DetectionEvent.Type=BEHAVIOR`, `Severity ∈ {MEDIUM, HIGH}` 생성, `notifier.onEvent()` 호출	이벤트 저장 ≥1회, 마지막 캡처 `Type=BEHAVIOR`, `Severity` MED/HIGH, 알림 ≥1회	Pass (`verify(eventRepo, atLeastOnce)`, `verify(notifier, atLeastOnce)`)
UT-06	행 반환 수=100(저활동) → 이벤트/알림 없음 (임계↑로 강제 비발생)	임계치 Medium/High를 매우 크게 설정, 동일 시각 2건 + flush	이벤트 미발생, 알림 미호출	저장/알림 호출 0회	Pass (`verify(..., never())`)
UT-05-A	경계 하회: 낮은 활동량 → 이벤트 없음 (임계↑)	임계↑ 설정, 20건 + flush	이벤트/알림 없음	저장/알림 호출 0회	Pass
UT-05-B	경계 상회: 임계 이상 활동량 → 이벤트 발생	기본 임계 (0.8/0.95), 80건 + flush	이벤트 ≥1회, 알림 ≥1회	저장/알림 모두 호출됨	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 이벤트 생성 + 상태보정(FAILURE) + 알림	role=READ_ONLY, SQL=`DELETE FROM orders;`	이벤트 생성(Type=AUTHZ, Sev=HIGH), `QueryLog.status=FAILURE`, `eventRepo.save()` 1회, `notifier.onEvent()` 1회	전부 일치	Pass (`assert*`, `verify(...)`)
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 중 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)`, `assertArrayEquals(to)`)

TC ID	테스트명	입력(요약)	기대 결과	실제 결과	판정/근거
UT-10-B	sendTo() → from 없으면 springMailUsername 폴백	`from=null`, 리플렉션으로 `springMailUsername` 세팅 후 `sendTo(...)`	전송 1회(폴백 사용), 수신자는 인자로 지정한 값	동일	Pass (`verify(send)`, `assertArrayEquals(to)`)
UT-10-C	disabled=true → 전송 호출 없음	`enabled=false`	`sendTo` 호출해도 `JavaMailSender#send` 미호출	동일	Pass (`verify(never())`)
UT-10-D	toDefault 누락 → IllegalStateException	`enabled=true, toDefault=null` 후 `send(...)`	예외 발생, 전송 호출 없음	동일	Pass (`assertThrows`, `verify(never())`)

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)	`/ok` 엔드포인트로 `send("hello world")`	호출 ≥1, payload에 `"text"`와 메시지 포함	동일	Pass (hitCount, body 확인)
UT-09-B	500 응답 → RuntimeException	`/err`로 `send("oops")`	예외 메시지에 `SLACK_WEBHOOK_HTTP_500` 포함	동일	Pass (`assertThrows`)
UT-09-C	disabled=true → 호출 없이 리턴	`enabled=false`	hitCount=0	동일	Pass
UT-09-D	webhookUrl 누락 → IllegalStateException	`webhookUrl=null`	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	`emailSender.send()` 1회, `notifRepo.save(EMAIL,SENT)` 1건	동일	Pass
UT-09-A	Slack ON, Email OFF → SLACK/SENT 1건	슬랙 ON, Email OFF	`slackSender.send()` 1회, `notifRepo.save(SLACK,SENT)` 1건	동일	Pass
UT-09-B / UT-10-B	Email+Slack ON → 각 채널 SENT 1건씩	둘 다 ON	email/slack 각각 1회 호출, `NotificationLog` 2건(EMAIL/SENT, SLACK/SENT)	동일	Pass (2건 저장 검증)
UT-10-C	Email 실패 → EMAIL/FAILED 저장	Email ON, `emailSender` 예외	`notifRepo.save(EMAIL,FAILED)` 1건(오류 코드/메시지 포함)	동일	Pass

TC ID	테스트명	입력(요약)	기대 결과	실제 결과	판정/근거
UT-10-D	Slack 실패 → SLACK/FAILED 저장	Slack ON, `slackSender` 예외	`notifRepo.save(SLACK,FAILED)` 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	`repo.save()`가 저장 엔티티의 `id=100L`로 세팅해 반환하도록 스텁	`save()` 1회 호출, 반환 `id != null`	`save()` 1회 검증, 반환 `100L`로 NotNull	Pass (`assertNotN` `verify(times(1))`)
UT-11-B	필수 누락/음수/빈값 → IllegalArgumentException	`userId=" "`, `sqlRaw=null`, `returnRows=-1`, `status=""` 각각 호출	각 케이스마다 `IllegalArgumentException` 발생, `repo.save()` 호출 안 함	예외 4건 모두 발생, `save()` 호출 0회	Pass (`assertThrow` `verify(never())`)
UT-11-C	시간 포맷 오류 → invalid executedAt	`executedAt="2025/10/21 12:00"` (잘못된 포맷)	`IllegalArgumentException("invalid executedAt")`	동일 예외 및 메시지 포함	Pass (`assertThrow` `assertTrue(msg co`
UT-11-D	잘못된 status → invalid status	`status="DONE"`	`IllegalArgumentException("invalid status")`	동일 예외 및 메시지 포함	Pass (`assertThrow` `assertTrue(msg co`
UT-11-E	sqlRaw 8KB 초과는 절단 저장	`sqlRaw` 길이 9000	저장 엔티티의 `sqlRaw.length()==8192`	캡처된 엔티티의 길이 8192	Pass (`assertEqua` `captured.getSqlRaw`
UT-11-F	sqlSummary 512 초과는 절단 저장	`sqlSummary` 길이 600	저장 엔티티의 `sqlSummary.length()==512`	캡처된 엔티티의 길이 512	Pass (`assertEqua` `captured.getSqlSur`

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)	Pass	-

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

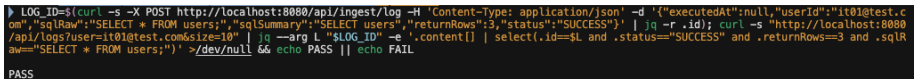
4.2.1. IT-01: Query Logging

- 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 퀵 체크



4.2.2. 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 O {"id":" 2e8192 c709-4 b4a9d
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 퀵 체크

4.2.3. BehaviorEvent

- 모듈: `com.example.dbids.modules.behavior.BehaviorDetector` , `com.example.dbids.modules.notify.NotificationService`

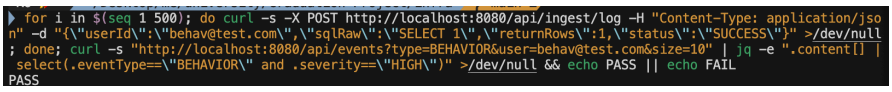
  엔드포인트: `POST /api/ingest/log` , `GET /api/events?type=BEHAVIOR`
- 검증 목적
  - 동일 사용자 고빈도 요청(스파이크) 시 `Type=BEHAVIOR` 이벤트가 생성되는지
  - 이벤트 심각도(Severity)가 정책 임계치 이상일 때 `HIGH` 로 표기되는지
  - 조회 API에서 해당 사용자 필터로 이벤트를 검색 가능하는지
- 사전 조건
  - `baseUrl=http://localhost:8080` (`profile=local`)
  - 행동 탐지 파라미터: `windowSeconds=60` , `thresholdMedium=1.0` , `thresholdHigh=2.0` , `wQpm=1.0` , `wWriteRatio=1.0` , `wDdlPerMin=1.0` , `wErrorBurst=0.5`
  - 판정 근거: 아래 Pass/Fail 기준 충족 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.com` 에 대해 1초 내 다건 `POST /api/ingest/log` , 이후 `GET /api/events?type=BEHAVIOR&user=...`	`200 OK` , `content[].eventType=="BEHAVIOR"` , 해당 사용자 이벤트 최소 1건	`content[0].eventType="BEHAVIOR"` , `userId="behav@test.com"` , 페이징 `totalElements=1`	Pass
IT-03-B	임계 초과(QPM)로 `HIGH` 확인	동일 입력; `sqlPreview` 내 QPM 수치 확인	`content[].severity=="HIGH"` , `sqlPreview` 내 `QPM` 임계 이상	`severity="HIGH"` , `sqlPreview="USER=behav@test.com QPM=200.00 ..."`	Pass

- 증빙
  - GET 응답

`{"id":"fe4f767e-ff4e-4467-a4e3-d60117f124d2","logId":"5c87191a-f0cb-4931-8fb8-3499a267ff53","eventType":"BEHAVIOR","severity":"HIGH","occurredAt":"2025-10-29T12:55:25.691259Z","userId":"behav@test.com","adminId":null,"sqlPreview":"USER=behav@test.com QPM=200.00 WRITE_RATIO=0.00 DDL/m=0.00 ERR=0 SCORE=200.00"}`

- CLI Quick Check



4.2.4. 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/DDL`
  - 판정 근거: 아래 Pass/Fail 충족 + 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
IT-04-B	심각도 HIGH	동일 입력	`severity="HIGH"`	이벤트 `severity="HIGH"` 확인	Pass
IT-04-C	logId 일치성	POST 응답의 `id` 와 이벤트 `logId`	`logId == id`	jq 필터로 동일성 확인됨	Pass

- 증빙
  - POST 응답

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

    `{"id":"0eb5c76b-4d11-4b59-a352-262238bd0735\n","logId":"5ac33925-d400-4685-8a66-20e80885fdb6\n","eventType":"AUTHZ","severity":"HIGH",...,"sqlPreview":"INSERT INTO users(id) VALUES (1)"}`
- CLI Quick Check

4.2.5. 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 훅):

```
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 id와 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	`userId="alert-ok@test.com"`, `sqlRaw="DROP TABLE important_table"`	`POST` 200/201, `GET`에서 이벤트 존재	`POST` 201, `GET` 200, PATTERN 이벤트 발견	**Pass**
IT-05-B	Email 발송 성공	IT-05-A와 동일	메일 발송 성공(실메일 수신 또는 NotificationLog `EMAIL/SENT`)	실메일 수신/또는 DB에서 `EMAIL/SENT` 확인	**Pass**
IT-05-C	Slack 발송 성공 (WireMock)	IT-05-A와 동일	WireMock `/slack`	`__admin/requests/mappings` 확인	**Pass**
IT-05-D	logId 연계 일치	POST `id` ↔ 이벤트 `logId`	`logId == id`	일치 확인	**Pass**
IT-05-E	NotificationLog 기록	DB 조회	`EMAIL/SENT` & `SLACK/SENT`	최근 로그에 두 채널 `SENT` 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_tabl
e","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-0fbd-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) 히트 ≥ 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-ffbb-489c-9bf7-da1498475551|EMAIL|SENT|2025-10-30T09:05:20.911324Z
fcc1bfc3-ffbb-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

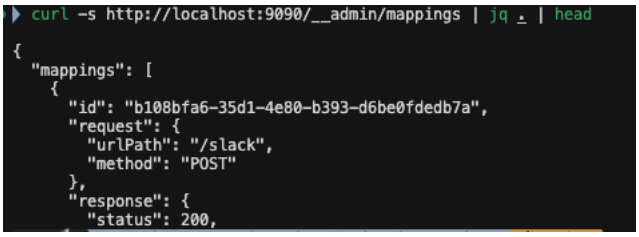
```
DB-IDS Detection Alert

Event ID : fcc1bfc3-ffbb-489c-9bf7-da1498475551
Log ID   : ea50e18b-0fbd-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
```



4.2.6. 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

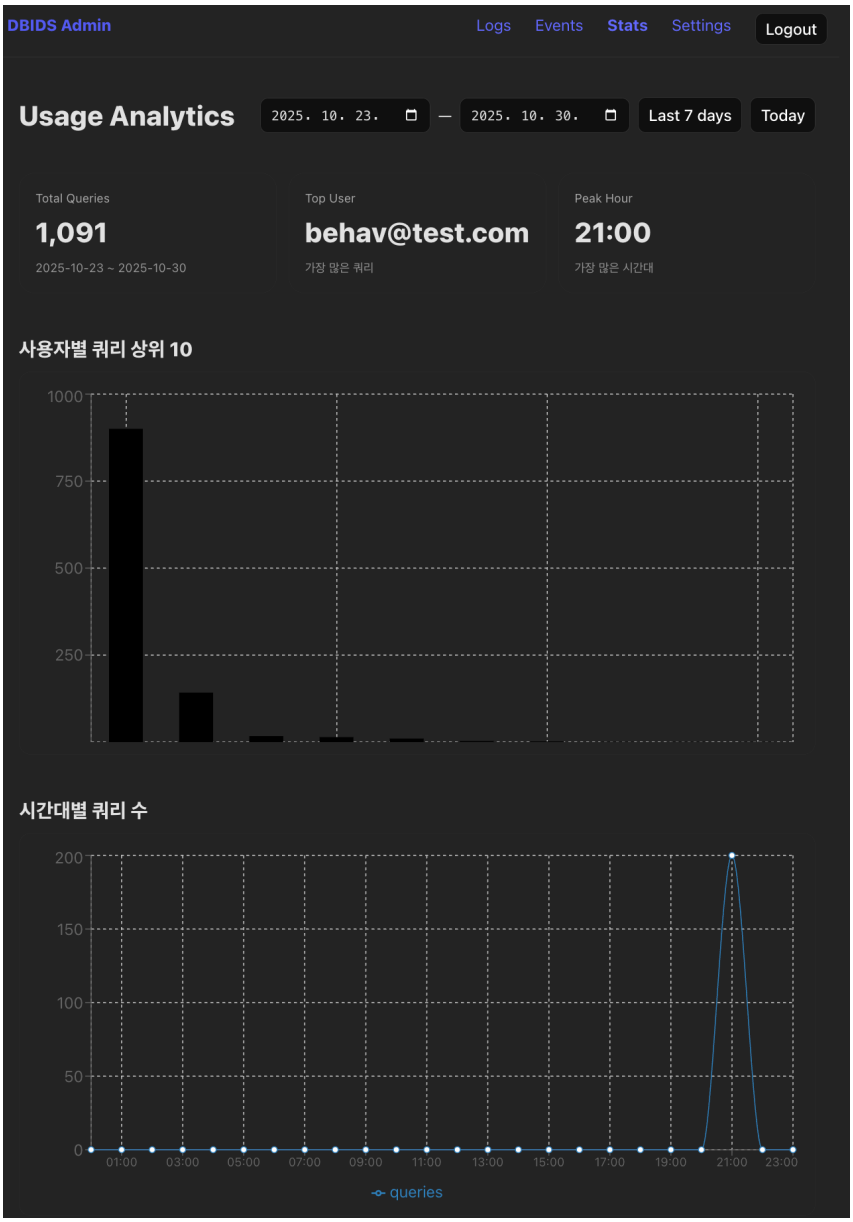
```
PATTERN_ID=$(curl -s -X POST "http://localhost:8080/api/ingest/log" \
  -H "Content-Type: application/json" \
  -d '{"userId":"dash@test.com","sqlRaw":"DROP TABLE t_dash","sqlSummary":"drop table t_dash","returnRows":
```

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

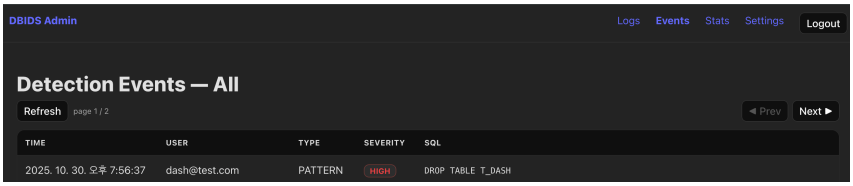
- 테스트 케이스 및 결과

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

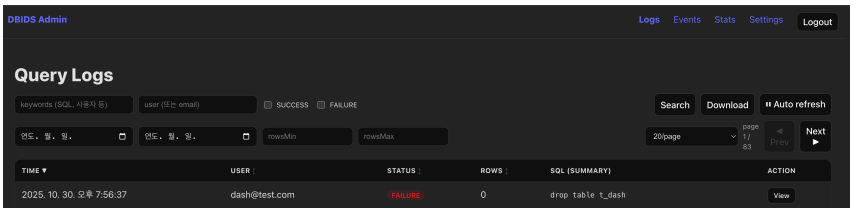
- 증빙
  - `/stats`

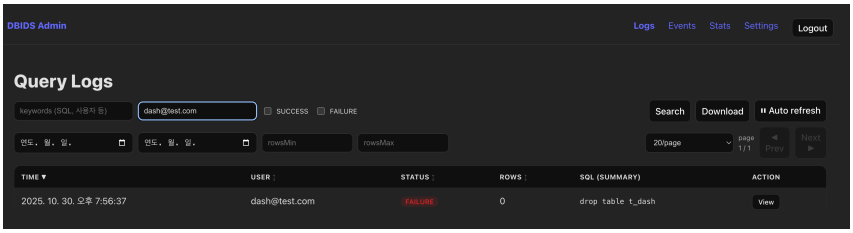

DBIDS Admin
Logs
Events
Stats
Settings
Logout
Usage Analytics
2025. 10. 23.
—
2025. 10. 30.
Last 7 days
Today
Total Queries
1,091
2025-10-23 ~ 2025-10-30
Top User
behav@test.com
가장 많은 쿼리
Peak Hour
21:00
가장 많은 시간대
사용자별 쿼리 상위 10
1000
750
500
250
시간대별 쿼리 수
200
150
100
50
0
01:00
03:00
05:00
07:00
09:00
11:00
13:00
15:00
17:00
19:00
21:00
23:00
queries

- /events

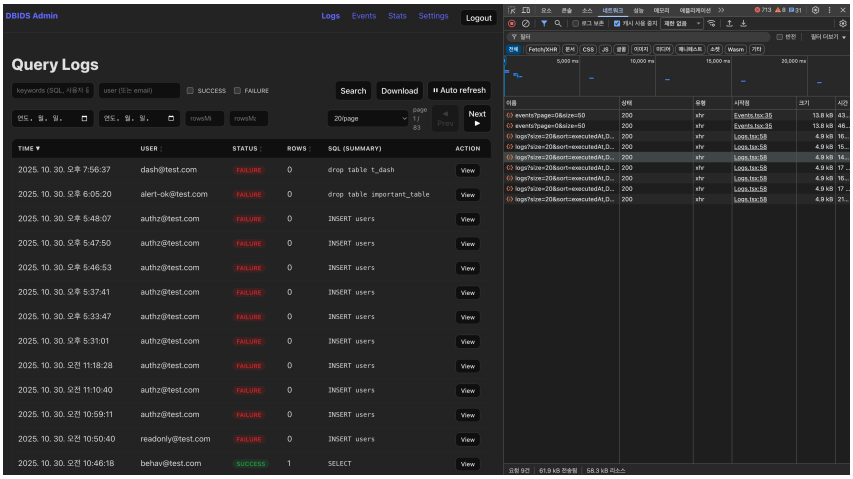

DBIDS Admin
Logs
Events
Stats
Settings
Logout
Detection Events — All
Refresh
page 1 / 2
◀ Prev
Next ▶
TIME
USER
TYPE
SEVERITY
SQL
2025. 10. 30. 오후 7:56:37
dash@test.com
PATTERN
HIGH
DROP TABLE T_DASH

- /logs





- 자동 갱신



`/logs` api가 주기적으로 실행됨

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

3.3. 시스템 테스트 (System Test)

TC ID	요구사항	시나리오	예상 결과	실제 결과	판정
ST-01	FR-1	Admin → SELECT 실행	QueryLog 저장, Dashboard 조회 가능	테스트 비진행	
ST-02	FR-2	UNION SELECT password FROM users	Event.Type=PATTERN, 알림 발송	테스트 비진행	
ST-03	FR-3	10,000건 SELECT / 1분간	Event.Type=BEHAVIOR, Severity=HIGH	테스트 비진행	
ST-04	FR-4	READ_ONLY → DELETE	Event.Type=AUTHZ, 알림 발송	테스트 비진행	
ST-05	FR-5	High Severity Event	Slack+Email 알림 도착	테스트 비진행	
ST-06	FR-6	Admin 로그인 후 이벤트 조회	UI-DB 값 일치	테스트 비진행	
ST-07	FR-6	로그인 실패	오류 메시지 표시	테스트 비진행	
ST-08	FR-7	30일 경과 로그 존재	ArchiveLog 생성	테스트 비진행	
ST-09	FR-7	S3 접근 불가	실패 후 재시도, 오류 로그 기록	테스트 비진행	

3.4. 비기능 테스트 (Non-Functional Test)

TC ID	항목	시나리오	예상 결과	실제 결과	판정
NFT-01	성능(탐지 지연)	1000건/초 SELECT	탐지 ≤ 1초	테스트 비진행	
NFT-02	성능(처리량)	1000TPS 부하	TPS ≥ 1000 유지	테스트 비진행	
NFT-03	성능(동시성)	Dashboard 100명 동시 접속	응답 ≤ 2초	테스트 비진행	
NFT-04	보안(SQL Injection)	`OR '1'='1'`	탐지 성공	테스트 비진행	
NFT-05	보안(권한 제어)	READ_ONLY → UPDATE	Event.Type=AUTHZ	테스트 비진행	
NFT-06	보안(비밀번호 저장)	Admin PW 확인	SHA-256 해시 저장	테스트 비진행	
NFT-07	신뢰성(탐지율)	정상1000, 공격50	오탐≤5%, 미탐≤2%	테스트 비진행	
NFT-08	신뢰성(알림 성공률)	Event 100건	성공률 ≥ 99%	테스트 비진행	

3.5. 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)	—

4. 주요 결함 (Defect Summary)

ID	유형	심각도	발생 TC	설명	조치
D-01	시스템 테스트	High	ST	시스템 테스트 진행하지 못함	이후 빠른 시일 내에 진행
D-02	비기능 요구사항 테스트	High	NFT	비기능 테스트 진행하지 못함	이후 빠른 시일 내에 진행

5. 첨부 자료

5.1 Unit Test 로그

5.1.1. 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.AuthZEngineTest" 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.modules.authz.AuthZEngineTest" time="0.001"/>
</testsuite>
```


5.1.2. 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.example.dbids.modules.authz.AuthZServiceTest" time="0.58"/>
  <testcase name="UT-08-A: userRole 미지정 → 정책 비적용(허용) → 이벤트 없음" classname="com.example.dbids.modules.authz.AuthZServiceTest" time="0.004"/>
  <testcase name="UT-08: DBA 사용자의 DELETE → 이벤트 없음, 상태보정/알림/저장 호출 없음" classname="com.example.dbids.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>
```


5.1.3. BehaviorDetectorTest


```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.behavior.BehaviorDetectorTest" tests="4" skipped="0" failures="0" errors="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.behavior.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.dbids.modules.behavior.BehaviorDetectorTest" time="0.001"/>
  <testcase name="UT-05: 1초에 200개 SELECT → Score ≥ 0.8, BEHAVIOR 이벤트 + 알림" classname="com.example.dbids.modules.behavior.BehaviorDetectorTest" time="0.019"/>
</testsuite>
```


5.1.4. DetectionServiceTest


```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.detection.DetectionServiceTest" tests="5" skipped="0" failures="0" errors="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.modules.detection.DetectionServiceTest" time="0.006"/>
  <testcase name="UT-03-DS: OR 1=1 → PATTERN/MEDIUM 이벤트(정규화로 0=0) + 알림 호출" classname="com.example.dbids.modules.detection.DetectionServiceTest" time="0.005"/>
  <testcase name="UT-03-DS: SLEEP() → PATTERN/LOW 이벤트(상태 보정 없음) + 알림 호출" classname="com.example.dbids.modules.detection.DetectionServiceTest" time="0.004"/>
  <testcase name="UT-03-DS: UNION SELECT → PATTERN/MEDIUM 이벤트(상태 보정 없음) + 알림 호출" classname="com.example.dbids.modules.detection.DetectionServiceTest" time="0.004"/>
  <testcase name="UT-03-DS: DROP TABLE → PATTERN/HIGH 이벤트 + QueryLog.status=FAILURE 보정 + 알림 호출" classname="com.example.dbids.modules.detection.DetectionServiceTest" time="0.004"/>
</testsuite>
```

5.1.5. 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="com.example.dbids.modules.notify.EmailSenderTest" time="0.023"/>
  <testcase name="UT-10-D: EmailSender send() - toDefault 누락 시 IllegalStateException" classname="com.example.dbids.modules.notify.EmailSenderTest" time="0.001"/>
  <testcase name="UT-10-C: EmailSender disabled=true → sendTo 호출해도 메일 전송 안 함" classname="com.example.dbids.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>
```

5.1.6. 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>
```

51.7. 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>
```

5.1.8. 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.ProxyRequestHandlerTest" time="0.002"/>
</testsuite>
```

5.1.9. 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>
```

5.1.10. 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>
```

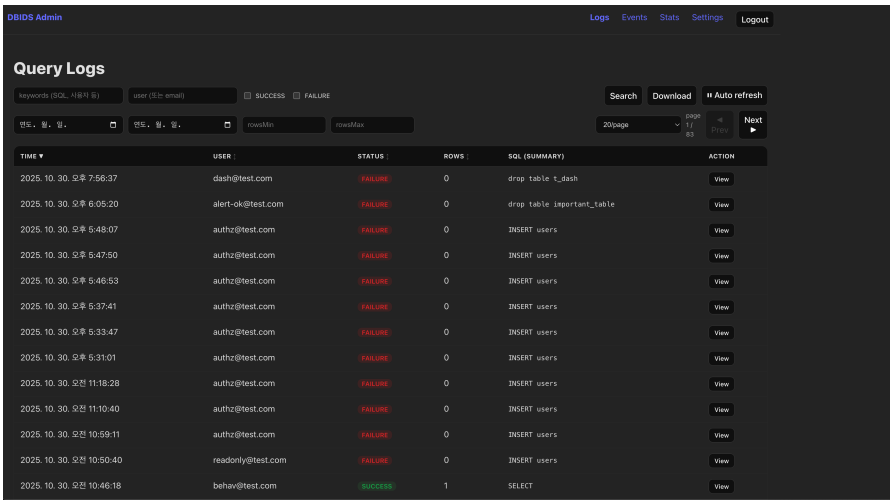

5.1.11. SqlNormalizerTest

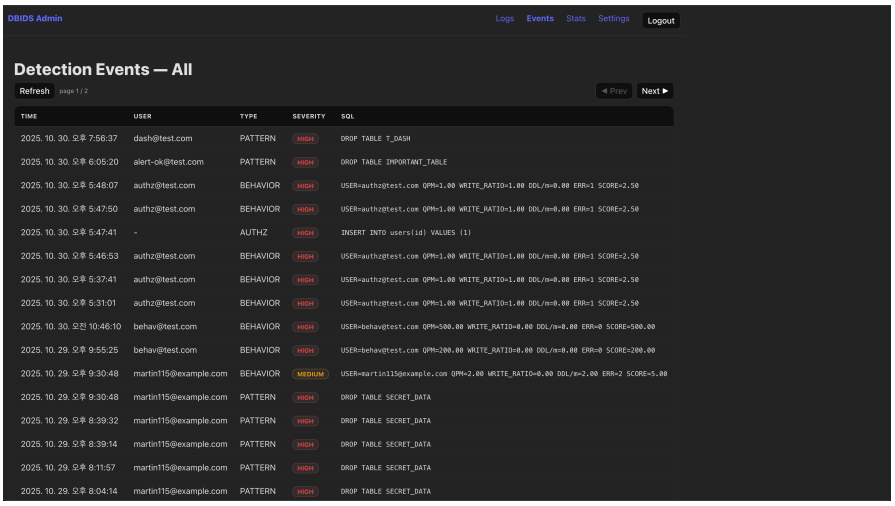

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.rule.SqlNormalizerTest" tests="3" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.002">
  <properties/>
  <testcase name="UT-03-RE/UT-04-RE 정규화: 블록/라인 주석 제거, 공백 정리, 대문자화" classname="com.example.dbids.modules.rule.SqlNormalizerTest"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

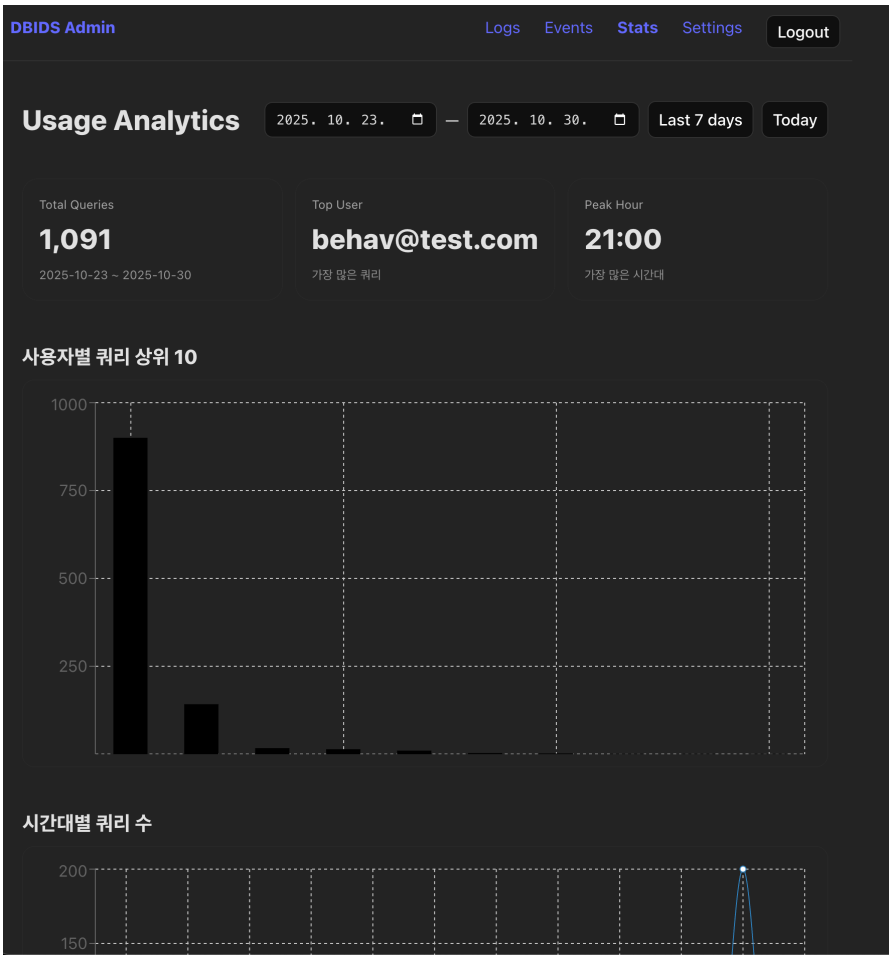

5.1.12. StorageServiceTest

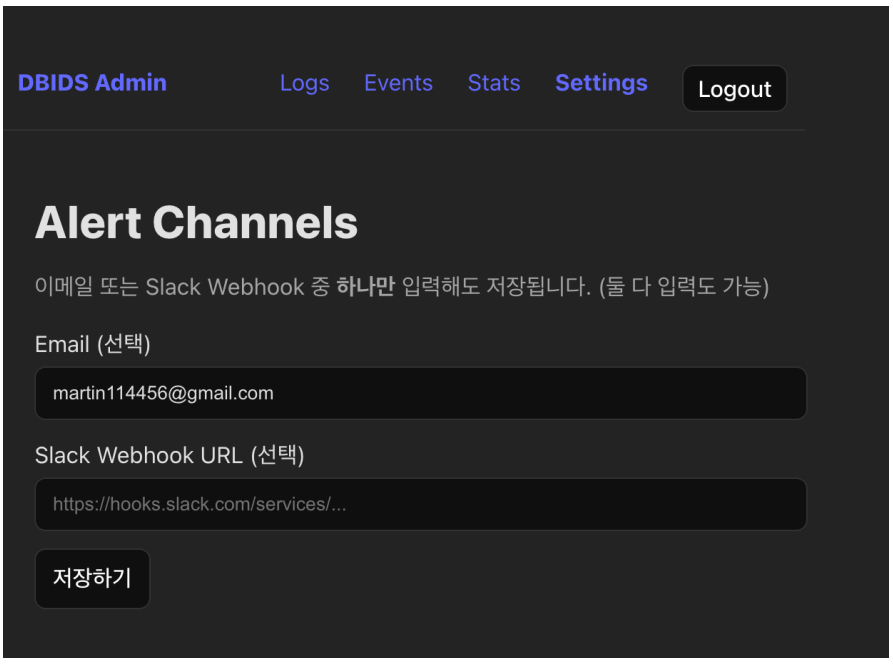

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="com.example.dbids.modules.storage.StorageServiceTest" tests="7" skipped="0" failures="0" errors="0" hostname="Ko-MacBookAir.local" time="0.042">
  <properties/>
  <testcase name="UT-11: 잘못된 status → invalid status" classname="com.example.dbids.modules.storage.StorageServiceTest" 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 누락/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>
```

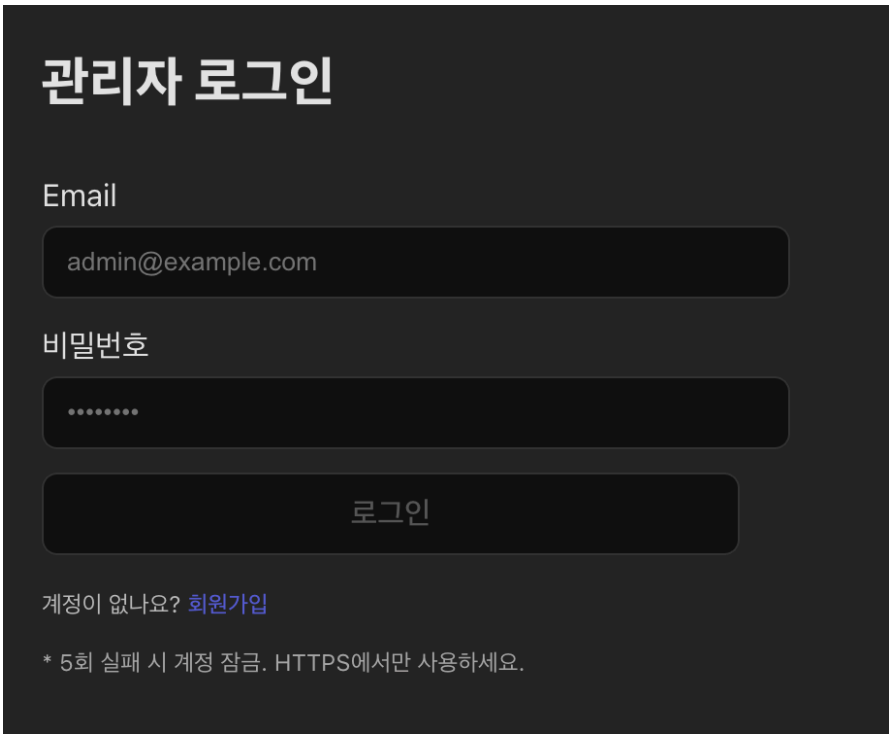

5.2. DashBoard 스크린샷


DBIDS Admin
Logs
Events
Stats
Settings
Logout
Query Logs
keywords (SQL, 사용자 등)
user (또는 email)
SUCCESS
FAILURE
Search
Download
Auto refresh
연도. 월. 일.
연도. 월. 일.
rowsMin
rowsMax
20/page
page 1 / 83
◀ Prev
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. 오전 11:18:28 authz@test.com FAILURE 0 INSERT users View
2025. 10. 30. 오전 11:10:40 authz@test.com FAILURE 0 INSERT users View
2025. 10. 30. 오전 10:59:11 authz@test.com FAILURE 0 INSERT users View
2025. 10. 30. 오전 10:50:40 readonly@test.com FAILURE 0 INSERT users View
2025. 10. 30. 오전 10:46:18 behav@test.com SUCCESS 1 SELECT View


DBIDS Admin
Logs
Events
Stats
Settings
Logout
Detection Events — All
Refresh
page 1 / 2
◀ Prev
Next ▶
TIME
USER
TYPE
SEVERITY
SQL
2025. 10. 30. 오후 7:56:37 dash@test.com PATTERN HIGH DROP TABLE T_DASH
2025. 10. 30. 오후 6:05:20 alert-ok@test.com PATTERN HIGH DROP TABLE IMPORTANT_TABLE
2025. 10. 30. 오후 5:48:07 authz@test.com BEHAVIOR HIGH USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ERR=1 SCORE=2.50
2025. 10. 30. 오후 5:47:50 authz@test.com BEHAVIOR HIGH USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ERR=1 SCORE=2.50
2025. 10. 30. 오후 5:47:41 - AUTHZ HIGH INSERT INTO users(id) VALUES (1)
2025. 10. 30. 오후 5:46:53 authz@test.com BEHAVIOR HIGH USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ERR=1 SCORE=2.50
2025. 10. 30. 오후 5:37:41 authz@test.com BEHAVIOR HIGH USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ERR=1 SCORE=2.50
2025. 10. 30. 오후 5:31:01 authz@test.com BEHAVIOR HIGH USER=authz@test.com QPM=1.00 WRITE_RATIO=1.00 DDL/m=0.00 ERR=1 SCORE=2.50
2025. 10. 30. 오전 10:46:10 behav@test.com BEHAVIOR HIGH USER=behav@test.com QPM=500.00 WRITE_RATIO=0.00 DDL/m=0.00 ERR=0 SCORE=500.00
2025. 10. 29. 오후 9:55:25 behav@test.com BEHAVIOR HIGH USER=behav@test.com QPM=200.00 WRITE_RATIO=0.00 DDL/m=0.00 ERR=0 SCORE=200.00
2025. 10. 29. 오후 9:30:48 martin115@example.com BEHAVIOR MEDIUM USER=martin115@example.com QPM=2.00 WRITE_RATIO=0.00 DDL/m=2.00 ERR=2 SCORE=5.00
2025. 10. 29. 오후 9:30:48 martin115@example.com PATTERN HIGH DROP TABLE SECRET_DATA
2025. 10. 29. 오후 8:39:32 martin115@example.com PATTERN HIGH DROP TABLE SECRET_DATA
2025. 10. 29. 오후 8:39:14 martin115@example.com PATTERN HIGH DROP TABLE SECRET_DATA
2025. 10. 29. 오후 8:11:57 martin115@example.com PATTERN HIGH DROP TABLE SECRET_DATA
2025. 10. 29. 오후 8:04:14 martin115@example.com PATTERN HIGH DROP TABLE SECRET_DATA


DBIDS Admin
Logs
Events
Stats
Settings
Logout
Usage Analytics
2025. 10. 23.
—
2025. 10. 30.
Last 7 days
Today
Total Queries
1,091
2025-10-23 ~ 2025-10-30
Top User
behav@test.com
가장 많은 쿼리
Peak Hour
21:00
가장 많은 시간대
사용자별 쿼리 상위 10
1000
750
500
250
시간대별 쿼리 수
200
150


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.