🏗️ Jobdori Architecture Renovation Master Plan

Version: 1.0

Author: CTO (Global Monitoring Tech Lead)

Date: 2026-01-30

1. Executive Summary (개요)

현재 Jobdori 서비스는 초기 검증(MVP) 단계를 지나 데이터 축적 단계에 진입했습니다. 그러나 초기 설계의 구조적 한계(JSONB 남용, 모놀리식 구조, 스냅샷 기반 통계)로 인해 **"대시보드 속도 저하"**와 **"통계 데이터 불일치"**라는 치명적인 문제가 발생하고 있습니다.

본 계획은 데이터베이스를 정규화하여 **데이터 무결성**을 확보하고, 프론트엔드와 백엔드를 분리하여 **확장성**과 **성능**을 극대화하는 것을 목표로 합니다.

2. Problem Diagnosis (현상 진단)

A. 성능 저하 (Performance Bottleneck)

- **현상:** 데이터가 쌓일수록 대시보드 로딩 시간이 기하급수적으로 증가.
- **원인:**
  1. **Full Table Scan:** 페이지네이션 없이 모든 세션 데이터를 한 번에 조회.
  2. **Heavy Payload:** `sessions` 테이블에 포함된 거대 JSON/Blob 데이터까지 불필요하게 전송.
  3. **Client Rendering Load:** 브라우저가 수천 개의 DOM 요소를 한 번에 처리.

B. 데이터 불일치 (Data Discrepancy)

- **현상:** 월간 통계, 슬랙 알람, 대시보드 내 수치가 서로 다름.
- **원인:**
  1. **Snapshot vs Live:** 슬랙/세션 데이터는 '과거 시점'에 박제된 정적 데이터인 반면, 관리자 승인 처리는 이를 갱신하지 않음.
  2. **Duplicate Standards:** 통계 기준이 'URL 수'와 '도메인 수'로 혼재되어 있음.
  3. **No Single Source of Truth:** 통계가 원천 데이터(Raw Data)가 아닌, 여러 곳에 흩어진 요약 테이블(JSONB)에 의존함.

C. 아키텍처 한계 (Technical Debt)

- **Frontend:** `src/app.ts` 내 HTML 문자열 하드코딩 (유지보수 불가 수준).
- **Database:** 관계형 DB(PostgreSQL)를 NoSQL처럼 사용하여 JOIN, Indexing 불가.
- **Backend:** Serverless 환경에서 Stateful(In-Memory Session) 패턴 사용으로 인한 인증 불안정.

3. Database Strategy: Normalization (백엔드 전략)

핵심 목표: 모든 탐지 결과를 개별 행(Row)으로 저장하여 실시간 통계 산출 기반 마련.

3.1 New Schema Structure (V2)

기존 JSON 덩어리를 분해하고, 핵심 테이블인 `detection_results` 를 신설합니다.

테이블명	역할	주요 변경점
`detection_results`	**(핵심)** 탐지된 모든 URL, 도메인, 상태, 검색 정보를 개별 기록	`session_id` 와 FK 연결, 실시간 통계의 원천
`sessions`	모니터링 회차 정보 관리	`results_*` 컬럼은 유지하되, 값은 실시간 View로 대체 가능하도록 설계
`monthly_stats`	(레거시 제거) 월별 요약	`top_contents` 등 **JSONB 컬럼 제거**. View로 대체
`pending_reviews_v2`	도메인 단위 승인 대기열	JSONB 제거. 빠른 조회를 위한 큐 역할 수행

3.2 Key Logic Guidelines

1. **Dual Write (이중 저장):** 마이그레이션 과도기 동안, 기존 Blob 저장 방식과 신규 DB 저장 방식을 병행하여 데이터 유실 방지.
2. **Retroactive Update (소급 적용):** 특정 도메인을 '불법'으로 확정 시, 과거에 탐지된 동일 도메인의 `pending` 데이터도 일괄 업데이트하여 과거 통계 보정.
3. **Index Optimization:** 날짜 기반 통계 쿼리 시 `TO_CHAR` 대신 `DATE_TRUNC` 를 사용하여 인덱스 히트율 확보.

4. Frontend Strategy: Modernization (프론트엔드 전략)

핵심 목표: 백엔드와 강결합된 HTML 코드를 제거하고, 독립적인 Next.js 애플리케이션 구축.

4.1 Tech Stack

- **Framework: Next.js 14+ (App Router)** - Vercel 배포 최적화 및 성능 확보.
- **Language: TypeScript** - 안정성 확보.
- **State/Fetching: TanStack Query (React Query)** - 서버 상태 동기화 및 캐싱 필수.
- **Styling: Tailwind CSS + Shadcn UI** - 빠른 개발 속도와 표준화된 디자인.
- **Charts: Recharts** - 경량화된 차트 라이브러리.

4.2 Architecture Rules

1. **Decoupling:** 프론트엔드 코드는 백엔드 DB에 절대 직접 접근하지 않음. 오직 JSON API만 호출.
2. **API Only:** 백엔드(`src/app.ts`)는 더 이상 HTML(`c.html`)을 반환하지 않고 JSON(`c.json`)만 반환하도록 수정.
3. **No Login (Yet):** 1단계 개발에서는 로그인 기능을 제외하되, 추후 도입을 고려한 Layout 구조 유지.

4.3 Development Checklist

- [ ] **Pagination:** 모든 리스트 조회 시 `page`, `limit` 파라미터 사용 필수.
- [ ] **Timezone:** 서버의 UTC 시간을 클라이언트 브라우저 시간(KST)으로 변환하여 표시.
- [ ] **Empty States:** 데이터가 없을 때의 UI(Skeleton, Empty Placeholder) 구현.
- [ ] **Optimistic Updates:** 승인/반려 버튼 클릭 시, UI를 먼저 갱신하고 API 요청을 백그라운드에서 처리 (UX 향상).

5. Implementation Roadmap (실행 로드맵)

우선순위에 따라 순차적으로 진행하며, 각 단계는 독립적으로 배포 가능해야 합니다.

🥇 Priority 1: The Foundation (DB 정규화)

- **목표:** 데이터 구조를 바로잡아 통계 불일치 원천 차단.
- **작업:**
  - `schema-v2.sql` 적용 (`detection_results` 테이블 생성).
  - `src/lib/db.ts` 에 Drizzle ORM 도입.
  - `run-pipeline.ts` 수정하여 탐지 결과를 DB에 적재 (Dual Write 시작).

🥈 Priority 2: The Performance (API 최적화)

- **목표:** 대시보드 로딩 속도 1초 이내 진입.
- **작업:**
  - `/api/sessions` 등 목록 조회 API에 페이지네이션(`LIMIT`, `OFFSET`) 적용.
  - DTO 패턴 적용: 목록 조회 시 무거운 컬럼(`file_final_results` 등) 제외하고 요약 정보만 전송.

🥉 Priority 3: The Accuracy (동적 통계 구현)

- **목표:** "내가 보고 있는 숫자가 진실임"을 보장 (SSOT).
- **작업:**
  - `monthly_stats` 테이블 의존성 제거.
  - `detection_results` 기반의 실시간 집계 쿼리(`GROUP BY`) API 구현.
  - 승인(`review`) 액션 시 트랜잭션을 통해 상태값 일괄 갱신 로직 구현.

🔒 Priority 4: Authentication (추후 진행)

- 로그인/보안 기능은 위 1~3단계 안정화 후 진행. (현재는 후순위)

6. Cost Analysis (비용 분석)

결론: 추가 비용은 거의 없으며, 오히려 장기적으로 감소 예상.

1. **Database (Neon):**

- 정규화로 행(Row) 수는 증가하나, 불필요한 거대 JSON I/O가 줄어들어 전체 부하는 상쇄됨. (Free Tier 유지 가능)

2. **Serverless (Vercel):**
   - API 최적화(Priority 2)를 통해 함수 실행 시간(Duration)과 전송량(Bandwidth)이 감소하여 비용 절감 효과.
3. **Frontend:**
   - Vercel 정적 호스팅은 기본 무료 제공 범위 내에서 충분함.

승인자: Jobdori CEO

작성자: CTO