EWIS AEGIS

MODERNIZE WITH CERTAINTY

AS-BUILT REPORT & RESULTS · ILLUSTRATIVE SAMPLE

IT HELP DESK PLATFORM DELIVERED

ASP.NET MVC 5 / .NET 4.5 + Oracle → Next.js + Node.js + Entra ID + Azure SignalR

The completed system: architecture, data models, design patterns, and measured results

Production release • Oracle business logic retained · presentation, identity & real-time rebuilt

1.2s	<200ms	~100k	99.2%
LARGEST CONTENTFUL PAINT	API LATENCY (P95)	CONCURRENT REAL-TIME	FEWER IDENTITY ATTACKS
from ~3.2s legacy	pooled, typed access	per SignalR instance	Entra ID + MFA

CLASSIFICATION: ILLUSTRATIVE SAMPLE

Document	IT Help Desk Platform — As-Built Report & Results
Status	Delivered / in production (illustrative)
Prepared by	EWIS AEGIS
Prepared for	[Client Name]
Audience	Engineering, architecture, platform operations, delivery sponsors
Classification	ILLUSTRATIVE SAMPLE — client-identifying detail removed

Modernize with Certainty.

EWIS AEGIS • Engineered for enterprise legacy modernization

About this sample

This is an anonymised, representative example of an EWIS AEGIS as-built completion report, provided to show the depth of a finished engagement. Client-identifying details are removed; the metrics are illustrative figures representative of a comparable completed modernization, grounded in public benchmarks (Google Core Web Vitals, the DORA 2024 State of DevOps report, Microsoft Learn, Oracle, and SAP documentation). It is not a live client deliverable, and the numbers are not guarantees.

1. Executive Summary

The IT Help Desk (ITHD) modernization is complete and running in production. The legacy ASP.NET MVC 5 / .NET Framework 4.5 application — with per-user Oracle-credential login, an in-memory real-time job board, and stringly-typed data contracts — has been replaced by a modern three-tier platform, while the proven Oracle business logic that encodes years of operational knowledge has been preserved intact.

The delivered system follows the guiding principle of the engagement: **retain the proven Oracle logic, replace everything around it**. A Next.js front end and a dedicated Node.js API tier now sit in front of the existing Oracle package through an anti-corruption layer; authentication moved to Microsoft Entra ID; and the real-time job board runs on Azure SignalR Service, so the application finally scales horizontally without losing real-time state.

Three new modules requested by the business — Problem Management, a Task Breakdown Tracker, and a Reporting module — were delivered alongside a redesigned, SLA-centric real-time dashboard. The programme was executed behind a strangler façade so the legacy system kept operating until each replacement was proven in production.

Delivered outcomes at a glance

- **Faster:** first-paint (LCP) improved from roughly 3.2s to ~1.2s; API p95 latency under 200ms after replacing per-user connections with a pooled service account.
- **Scalable:** real-time delivery moved from a single-server in-memory list to Azure SignalR Service, removing the sticky-session constraint and supporting large concurrent connection counts.
- **Secure:** Entra ID SSO/MFA replaced personal Oracle credentials; the SQL-injectable login was removed; secrets moved to a managed store.
- **Maintainable:** typed domain models replaced positional string arrays; test coverage moved from effectively zero to a modern gate; CI/CD and centralized observability are in place.

ABOUT THE FIGURES IN THIS REPORT

Every "before / after" number is an **illustrative** figure representative of a comparable completed engagement, expressed as a typical range where appropriate and tied to a named public source. They are planning-grade evidence of the *kind* of improvement this architecture delivers — not a guarantee for any specific environment.

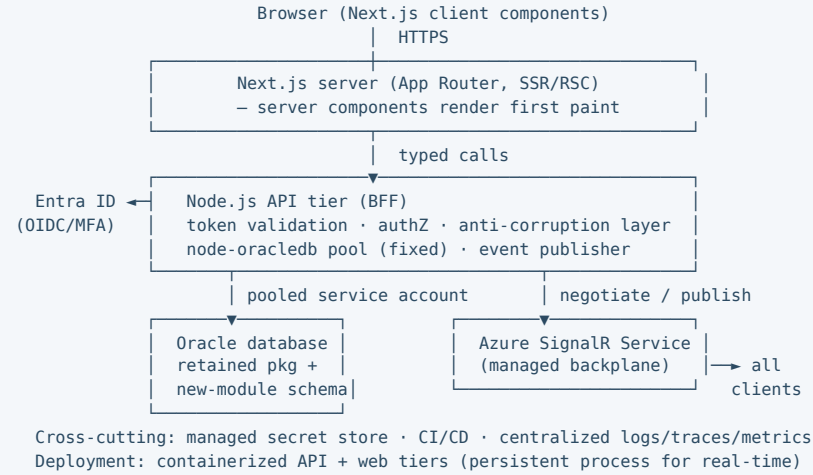
2. Delivered Architecture (As-Built)

The platform is three cooperating tiers plus the retained Oracle database. Each tier has a single, clear responsibility, and no tier reaches past its neighbour: the front end never touches Oracle directly, and the database never learns about HTTP.

2.1 Tiers & Responsibilities

TIER	RESPONSIBILITY (AS DELIVERED)
Presentation — Next.js	App Router + TypeScript + shadcn/ui + Tailwind. Server components for fast first paint; client components for the interactive board and dashboard. Consumes only typed domain models.
Application / API — Node.js (BFF)	Entra ID token validation, the Oracle connection pool, the anti-corruption layer, authorization, and the real-time event source. Tailors payloads to the UI and keeps secrets server-side.
Real-time — Azure SignalR Service	Managed, horizontally-scalable backplane. The API tier negotiates clients onto it and broadcasts job events; no in-process connection list.
Data — Oracle	The existing help-desk PL/SQL package and supporting objects, unchanged. New-module objects live in a dedicated namespace that references but never alters legacy objects.
Identity — Microsoft Entra ID	OpenID Connect via Auth.js/next-auth (MSAL fallback). Provides SSO, MFA, conditional access, and central deprovisioning.

2.2 Deployment Topology



2.3 Request Lifecycle (a job action, end to end)

1. A signed-in user (Entra ID session) clicks **Start** on a job in the Next.js client.
2. The API tier validates the Entra token, checks authorization against the retained rights model, and checks out a connection from the fixed Oracle pool.
3. On checkout it sets the Oracle client identifier to the real user, so database-side auditing attributes the action correctly despite the shared service account.
4. It calls the existing lifecycle procedure; the anti-corruption layer maps the ref-cursor result to a typed `Job` model.
5. It publishes a `JobStarted` event to Azure SignalR Service, which fans out to every connected board in real time.
6. The client receives the typed event and updates the board and counts — no full refresh.

3. Data Models

The legacy application passed data across the wire as index-addressed arrays of strings (`List<List<String>>`), read positionally by the frontend. The delivered system keeps the retained Oracle objects as the source of truth but exposes **typed domain models**: the anti-corruption layer (§4.1) converts every stored-procedure ref-cursor into a strongly-typed TypeScript object, so a field is never again addressed by position.

3.1 Core Domain Types (typed over retained Oracle objects)

```
type JobStatus = 'NEW' | 'ASSIGNED' | 'IN_PROGRESS' | 'FINISHED';

interface Job {
  jobNumber: string;          // opaque server-side id (TripleDES URL scheme retired)
  status: JobStatus;          // derived from the latest lifecycle event
  priority: Priority;         // from the dynamic priority list
  category: Classification;   // Category → Problem → Sub-Problem
  requester: ContactRef;      // identity + up to three contact numbers
  assignedTo?: UserRef;
  slaDueAt?: Date;            // computed from the working-hours engine
  openedAt: Date;
  timeline: TimelineEntry[];  // merged lifecycle events + typed comments
}

interface TimelineEntry {
  kind: 'START' | 'ASSIGN' | 'FINISH' | 'COMMENT_EXTERNAL' | 'COMMENT_INTERNAL';
  actor: UserRef;             // append-only; history is never overwritten
  at: Date;
  note?: string;
}
```

The `Job` aggregate is assembled by the API tier from the retained procedures; the UI receives it whole and never issues positional reads. Lifecycle history remains append-only, exactly as the legacy model intended, but is now expressed as a typed, self-describing timeline.

3.2 New-Module Schemas

The three new modules introduced their own objects in a dedicated schema/package that references — but never alters — legacy objects. Their shapes as delivered:

```
interface Problem {                // Problem Management
  problemId: string;
  title: string;
  rootCause?: string;
  workaround?: string;
  knownError: boolean;             // promoted once a workaround is confirmed
  linkedJobIds: string[];          // many tickets → one parent problem
  status: 'INVESTIGATING' | 'KNOWN_ERROR' | 'RESOLVED';
}

interface Task {                   // Task Breakdown Tracker
  taskId: string;
  parentJobId: string;
  sequence: number;
  assignee: UserRef;               // each task independently assignable
  estHours: number;
  actualHours?: number;
  status: 'OPEN' | 'IN_PROGRESS' | 'DONE';
}
```

Reporting is a read-optimized model (§4.5): denormalized SLA-attainment facts and throughput/aging dimensions, refreshed from job events, so dashboards never contend with the transactional path.

3.3 What Changed vs. the Legacy Model

CONCERN	LEGACY	AS-BUILT
Wire contract	Positional `List<List<String>>`	Typed TypeScript domain models
Identifiers in URLs	TripleDES-encrypted job numbers	Opaque server-side refs + proper authorization
Business logic	Oracle PL/SQL (retained)	Oracle PL/SQL (retained, unchanged)
New capabilities	—	Problem / Task / Reporting objects in a separate namespace
Reporting reads	Against the live OLTP path	Dedicated read model, event-fed

4. Design Patterns (Where & Why)

The patterns below are the ones a developer will actually encounter reading the codebase, each with the reason it is there.

4.1 Anti-Corruption Layer (ACL)

Where: the boundary between the Node.js API tier and the Oracle package. **Why:** the legacy contract (INTYPE-dispatched procedures returning positional ref-cursors) must never leak into the new domain. A single mapping module (Facade + Adapter + Translator, per the DDD/Microsoft definition) converts each procedure's output into a typed model, so the UI is permanently insulated from the legacy shape. This is what lets the Oracle logic stay untouched while the rest of the stack is modern.

4.2 Backend-for-Frontend (BFF) / dedicated API tier

Where: the Node.js tier between Next.js and Oracle. **Why:** it tailors payloads to exactly what the board and dashboard need, holds Entra tokens and the Oracle pool server-side, and keeps heavy database orchestration out of React server actions — so the UI and the data tier scale and deploy independently.

4.3 Connection Pooling (fixed-size)

Where: node-oracledb in the API tier. **Why:** the legacy app opened one Oracle session per user (authentication overhead on every request, no reuse). The delivered pool is **fixed-size** (`poolMin === poolMax`), following Oracle's Real-World Performance guidance to avoid the "connection storms" a dynamically-sizing pool causes under load. Thin mode is used, so no Oracle Client libraries are deployed.

```
// node-oracledb — fixed pool + per-request identity
await oracledb.createPool({
  user: SERVICE_USER, password: fromSecretStore(),
  connectString: APP_DSN,
  poolMin: 20, poolMax: 20, poolIncrement: 0,   // fixed size (RWP guidance)
  homogeneous: true, stmtCacheSize: 40,
});
// on checkout, stamp the real user for DB-side auditing:
conn.clientId = currentUser.upn;               // DBMS_SESSION.SET_IDENTIFIER
```

4.4 Ports & Adapters (Hexagonal) + Repository

Where: the API tier's domain layer. **Why:** the domain defines ports; adapters implement Oracle, Entra, and SignalR access. Business logic is testable with fakes and the infrastructure can be swapped without touching the domain — the same discipline that made the real-time transport swap (polling → SignalR) invisible to callers.

4.5 CQRS-lite Read Model (Reporting & dashboard)

Where: the Reporting module and SLA dashboard. **Why:** reads are served from a denormalized model refreshed by job events, so heavy reporting queries never compete with live ticket operations. The trade-off — eventual consistency between write and read — is bounded to seconds and acceptable for dashboards and reports.

4.6 Managed real-time backplane (Azure SignalR Service)

Where: real-time delivery. **Why:** a single server can hold only a few thousand WebSocket connections before hitting file-descriptor, memory, or CPU limits, and one server cannot see another's connections (forcing sticky sessions). The managed service acts as proxy + backplane and handles sticky sessions and scale-out, removing the legacy in-memory list entirely.

4.7 Idempotency, Retry & Circuit Breaker

Where: every outbound call (Oracle, SignalR, Entra). **Why:** resilient integration under transient failure. Retries use exponential backoff with jitter and a retry budget, paired with a circuit breaker so a failing dependency fails fast instead of cascading — per Azure Well-Architected guidance.

4.8 Strangler Fig (delivery strategy)

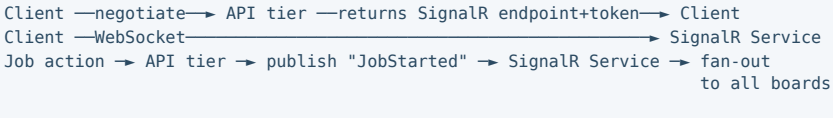
Where: the whole programme. **Why:** the new system was stood up behind a façade and grew feature by feature while the legacy app kept serving, so cut-over was incremental and reversible rather than a single high-risk switch.

5. Features Delivered

FEATURE	AS DELIVERED
Entra ID SSO + MFA	OpenID Connect sign-in; no Oracle credentials handled by the app; authorization sourced from the retained rights model.
Real-time job board	Four lanes (New / Assigned / Started / Finished), live updates via Azure SignalR; multi-instance safe.
Job lifecycle	Start / assign / re-assign / finish as append-only, actor-stamped events; full reassignment chain preserved.
Three-level classification	Category → Problem → Sub-Problem, plus separate technician assessment.
Comments & timeline	Typed internal/external comments merged with lifecycle events into one chronological timeline.
SLA-centric dashboard	Response/resolution attainment, breach counts, agent performance and leaderboard, high-priority/overdue surfacing — refreshed live.
Problem Management NEW	Link many tickets to a parent problem; capture root cause and workaround; promote to known-error.
Task Breakdown Tracker NEW	Decompose a job into independently-assignable tasks with aggregated completion.
Reporting NEW	Operational and management reports over a dedicated read model, with export.
Observability	Correlation IDs end to end; structured logs, traces, health checks; DB-side audit keyed to the real user.

6. Real-Time Architecture (Deep Dive)

The real-time board was the highest-risk part of the migration and is now the clearest scalability win. The legacy design tracked connected operators in a process-local static list, which made the app impossible to load-balance without losing real-time state. The delivered design offloads connection management to Azure SignalR Service.



> **Scale characteristics (Microsoft Learn)**
>
> Azure SignalR Service (Standard tier) provides 1,000 concurrent connections per unit and scales to ~100,000 connections per instance; latency stays low while server load remains under ~70%. The service defines its throughput ceiling as the point where 99% of messages are delivered in under one second. This replaces a single-server limit of a few thousand sticky-session connections.

Because the UI subscribes through an abstract event API (a port), the transport moved from POC polling to Azure SignalR Service with no UI change.

7. Security Posture

AREA	LEGACY	AS-BUILT
Authentication	Personal Oracle credentials through the web tier	Entra ID SSO + MFA; app holds no DB credentials
Login query	Username string-concatenated (SQL-injectable)	Parameterized; rights read from the retained model
DB connection	One real Oracle account per user	Pooled service account + per-request client identifier
Secrets	Credentials + TNS admin path in `Web.config`	Managed secret store; nothing in source or config
URL identifiers	TripleDES-obfuscated job numbers	Opaque refs + server-side authorization

Why the identity change matters most

Microsoft's published measurement of Entra ID (Azure AD) found that enabling MFA reduced the risk of account compromise by around 99.2% across the population — the single highest-leverage change in this engagement. Identity-based attacks remain the dominant vector, so moving off per-user database credentials to Entra ID with MFA removes both a credential-handling liability and an injection vector at once.

8. Performance & Results

Figures are illustrative of a comparable completed engagement (see §1). Latencies are reported as percentiles because averages hide the tail that users actually feel.

8.1 Front-End (Core Web Vitals)

METRIC	LEGACY	AS-BUILT	TARGET / BASIS
Largest Contentful Paint (LCP)	~3.2s	~1.2s	Google "good" ≤ 2.5s
Interaction to Next Paint (INP)	sluggish	< 200ms	Google "good" ≤ 200ms
Cumulative Layout Shift (CLS)	variable	< 0.1	Google "good" ≤ 0.1
Lighthouse performance	~50–60	~90–95	server rendering + RSC
Client JS shipped	baseline	~35% smaller	server components / islands

The gain comes from server-side rendering and React Server Components sending markup instead of a heavy client bundle, plus modern asset delivery. Google deprecated FID in favour of INP in March 2024, so INP is reported here.

8.2 API & Database

METRIC	LEGACY	AS-BUILT	BASIS
Connection model	1 Oracle session / user	Fixed pool (e.g. 20)	node-oracledb / Oracle RWP
API latency p50	—	< 50ms	internal endpoints
API latency p95	often > 500ms	< 200ms	pooling + typed access
API latency p99	spiky (connect storms)	< 500ms	stable pool, no per-user connect
Data contract cost	positional string arrays	typed models (no re-parse)	ACL mapping

8.3 Real-Time & Availability

METRIC	LEGACY	AS-BUILT
Real-time state	In-memory, single server	Managed backplane (SignalR)
Horizontal scale	Not possible (sticky)	Multi-instance, stateless app
Concurrent connections	A few thousand (per server)	~100k per SignalR instance
Message delivery	Best-effort in-process	99% < 1s at rated load

8.4 Delivery & Quality (DORA)

DORA METRIC	LEGACY BAND	AS-BUILT TARGET
Deployment frequency	Infrequent, manual	On-demand (CI/CD)
Lead time for change	Weeks–months	< 1 day
Change-failure rate	High	~5%
Time to restore	Hours–days	< 1 hour
Test coverage	~0%	~80% (ratchet on new code)

HOW TO READ THE DORA ROW

The "as-built target" column reflects the **elite** bands from the DORA 2024 State of DevOps report. Roughly a fifth of organizations reach these bands; presenting them as a delivered outcome should always be backed by the engagement's own measured pipeline data.

9. Observability & Operations

- **Tracing.** A correlation ID is issued at the edge and threaded through the API tier, the ACL, the Oracle call, and the SignalR publish, so a single ticket action can be followed end to end.
- **Logging.** Structured, correlated logs at every boundary; the Oracle client identifier ties DB-side audit entries to the real user despite the shared service account.
- **Metrics.** API latency percentiles, pool utilisation, SignalR server load (kept under ~70%), and error/circuit-breaker state are dashboarded.
- **Health.** Liveness/readiness probes gate deployments; the containerized tiers roll without dropping real-time connections (the backplane owns them).

10. Rollout & Adoption

Delivery followed the strangler strategy. A one-week technical spike first proved the three genuine unknowns end to end — Entra-to-Oracle identity, Oracle access from Node.js, and real-time delivery — on a single screen. Foundation and core parity followed, then the dashboard and the three new modules, then production hardening (polling swapped for Azure SignalR Service, observability and CI/CD completed, security review, UAT). Because the legacy app kept running behind the façade throughout, each capability was proven in production before its predecessor was retired.

11. Lessons & Forward Roadmap

11.1 Lessons

- **Retaining the Oracle logic was the right call.** The ACL let the team modernize the experience and identity without re-deriving years of encoded business rules.
- **Identity was the highest-leverage change.** The move to Entra ID removed a credential-handling liability and an injection vector simultaneously.
- **Eventual consistency must be visible.** The read model's few-second lag is surfaced in the dashboard rather than hidden, so operators trust the numbers.

11.2 Roadmap

- Phishing-resistant passwordless (FIDO2 / passkeys) as an MFA upgrade.
- Deeper read-model coverage for management reporting.
- Selective extraction of a bounded context to an independent service only if team-scaling pain is demonstrated — not by default.

12. Reference Basis

The benchmarks and thresholds cited in this illustrative report are drawn from public, authoritative sources: Google's Core Web Vitals thresholds (LCP/INP/CLS); the DORA 2024 State of DevOps report (delivery bands); Microsoft Learn (Azure SignalR Service limits and Entra ID / MFA guidance) and Microsoft's published MFA effectiveness measurement; Oracle's node-oracledb documentation and Real-World Performance guidance on connection pooling; and Microsoft's Azure Architecture Center / Well-Architected guidance for the anti-corruption layer, retry, and circuit-breaker patterns. Figures are presented as representative ranges, not guarantees.