

## Technical Architecture — Query Responses

City of Johannesburg | 2026-02-18

Document	Chain360 Architecture — Technical Query Responses
Prepared for	City of Johannesburg
System	Chain360
Revision	1.0
Scope	Technology Stack · Performance & Scalability · Security · Integration · Hosting

**POSITION STATEMENT:** Chain360 was developed using PHP 8.2 MVC and MySQL 8.3 The BRS was written pre-development as a technology hypothesis. During implementation, the technology stack was deliberately revised based on concrete technical and operational reasons documented in Section 1.1. This document reflects the as-built system and provides the rationale for all technology decisions. Where gaps exist (security hardening, HTTPS enforcement), they are identified with clear remediation actions.

## Executive Status Summary

Aspect	Current State	Action Required	Status
PHP Framework	PHP 8.2 MVC — deliberate decision away from Laravel/Vue. Fully documented rationale.	None — decision finalised and justified	IMPLEMENTED
MySQL 3NF	Core tables fully comply; JSON schema blobs are intentional	Document intentional denorm decisions	IMPLEMENTED
3-Tier Architecture	Full layer separation: views / controllers / DB	None — fully implemented	IMPLEMENTED
Concurrency / Cache	3 dedicated VMs (App, DB, Storage)	Configure VMs per	PLANNED

Aspect	Current State	Action Required	Status
	per environment. Nginx + PHP-FPM + Redis architecture defined.	documented topology	
<b>Horizontal Scaling</b>	Storage VM NFS-mounted so App VM is stateless — second App VM can be added with no code change	Implement Nginx upstream pool when needed	<b>PLANNED</b>
<b>AJAX / Lazy Load</b>	ApiController JSON endpoints exist; PHP renders full pages	Implement DataTables lazy loading on asset list	<b>PARTIAL</b>
<b>AES-256 at Rest</b>	No column-level encryption; bcrypt for passwords only	Encrypt PII columns + LDAP bind password	<b>PLANNED</b>
<b>SSL/TLS in Transit</b>	LDAPS port 636 confirmed; web HTTPS not yet enforced	Enable HTTPS on web server — URGENT	<b>PARTIAL</b>
<b>AD / RBAC</b>	LDAP sync + role mapping fully implemented and verified	None — fully implemented	<b>IMPLEMENTED</b>
<b>Audit Trail</b>	auth_audit_log + workflow_state_history + field_changes JSON	None — exceeds NFR-10 requirements	<b>IMPLEMENTED</b>
<b>SAP Integration</b>	7 endpoints configured; async queue operational	Upgrade bearer token to OAuth 2.0 for production	<b>PARTIAL</b>
<b>99.9% Uptime</b>	No MySQL replica; no failover; no monitoring configured	Add replica + Nginx upstream + backups	<b>PLANNED</b>
<b>Async Tasks</b>	Custom api_integration_queue (cron-driven, retry logic)	Acceptable for on-prem; Azure SB optional uplift	<b>PARTIAL</b>
<b>Hosting Decision</b>	On-prem WAMP (built and running); Azure referenced in BRS only	Formal decision required from COJ stakeholders	<b>PLANNED</b>

## 1. System Architecture

Chain360 is a PHP application following the Model-View-Controller (MVC) pattern, running on a WAMP/LAMP stack (Windows/Linux, Apache, MySQL 8.3, PHP 8.2). The system is organised into three layers – Model, View, Controller – with clear separation of concerns, and integrates asynchronously with external systems (SAP, Active Directory, Service Desk).

### 1.1. MVC Structure

The application adheres to the MVC architectural pattern to keep code modular, maintainable, and testable.

- a) **Models:** Each major database table has a corresponding PHP model class (e.g., User, Asset, WorkflowInstance, FormSubmission). Models encapsulate data retrieval, validation, and business rules. They extend a base Model class that provides common database operations (find, save, delete). Models are “thin” – complex business logic is delegated to services.
  - a) **Views:** Views are plain PHP templates (.php files) that generate HTML. They receive data from controllers and output the presentation. No heavy logic resides in views; they only use simple loops and conditionals. Layout templates (header/footer) are used for consistency.
-

- b) Controllers:** Controllers handle HTTP requests, interact with models and services, and select the appropriate view to render. For example, AssetController responds to routes like `index.php?controller=asset&action=show&id=5`. A simple front controller (`index.php`) parses the URL parameters and instantiates the required controller and method. Controllers are kept thin – they delegate business logic to service classes (e.g., WorkflowEngine, AuthService).

### 1.2. Presentation Layer (View)

- a) Rendering:** Server-side rendered HTML using PHP templates. No client-side framework; minimal vanilla JavaScript enhances UI interactions (e.g., dynamic form fields, AJAX for approval actions).
- b) Dynamic Forms:** Forms are defined as JSON schemas stored in the forms table. The FormEngine (a helper class) renders fields dynamically based on the schema, enabling new form types without code changes. The view simply calls `$formEngine->render($formSchema)`.
- c) Templates:** All views are stored under `/views/` and organised by controller (e.g., `views/asset/index.php`, `views/workflow/show.php`). A master layout template includes the common header, navigation, and footer.

### 1.3. Business Logic Layer (Controller + Services)

Controllers orchestrate the flow, but core business logic resides in service classes.

- a) Routing:** A simple front controller (`index.php`) inspects the controller and action query parameters (e.g., `?controller=asset&action=transfer`) and instantiates the corresponding controller class. If no parameters are provided, a default controller/action is used. Routes are not declarative but follow a convention.
- b) Core Services:**
- WorkflowEngine – orchestrates state machines, evaluates approval routing rules, and triggers transitions.
  - AuthService – handles LDAP binds, local password verification, and session construction.
  - LDAPService – syncs user data from Active Directory and populates the users table.
  - APIService – manages queued external API calls.
  - NotificationService – sends email and in-app notifications.
- c) Controllers (examples):**
- EmployeesController, DepartmentsController – now query the unified users table with `user_type` filters.
  - AssetsController, AssetAllocationController – manage asset lifecycle and allocations.
  - WorkflowInstanceController – executes workflow transitions and approvals.
  - AuthController – handles login, logout, and session management.

Controllers receive user input (POST data), call the appropriate services or models, and then pass data to the view. They never contain SQL or complex domain logic.

### 1.4. Data Layer (Model)

**Database:** MySQL 8.3, InnoDB engine, utf8mb4 character set. The database `coj_asset_management` contains 48 tables, 5 views, and 3 stored procedures.

**JSON Usage:** Semi-structured data is stored in JSON columns (e.g., `form_schema`, `rule_conditions`, `request_payload`), avoiding EAV patterns and keeping the schema flexible.

### 1.5. Workflow Engine & Approval Routing

**State Machines:** Each workflow (e.g., asset transfer, acquisition, incident) is defined by states and transitions. States can be marked as initial, final, or requiring approval.

**Approval Routing Rules:** The table `approval_routing_rules` stores conditions (JSON) and approver determination logic (JSON). Rules are evaluated by priority at runtime.

**Types:** manager (resolves via `departments.department_head_user_id`), current\_owner (via `asset_allocations`), receiver (from form data), role (by role ID), and threshold (based on form field values).

---

**Asynchronous Processing:** Workflow transitions that trigger external API calls do not block the user; instead, they insert a record into `api_integration_queue`, which is processed by a cron job. This hybrid model ensures responsiveness while guaranteeing eventual consistency.

## 1.6. External Integrations

**SAP:** Endpoints for asset registration, transfer, and disposal. After a successful SAP call, the returned `sap_asset_number` is stored in the assets table.

**Active Directory:** Used for authentication (LDAP bind) and user sync. Incident workflows (e.g., stolen assets) trigger AD notifications to suspend accounts.

**Service Desk:** Creates tickets and checks inventory availability via REST APIs.

### Integration Tables:

`api_integration_queue` – pending/failed calls with retry counters.

`api_call_logs` – full audit trail of requests and responses.

**Retry Logic:** Failed calls are retried with exponential backoff (5 min, 15 min, 1 h) up to 3 times. After exhaustion, the status is set to failed and an admin alert is sent.

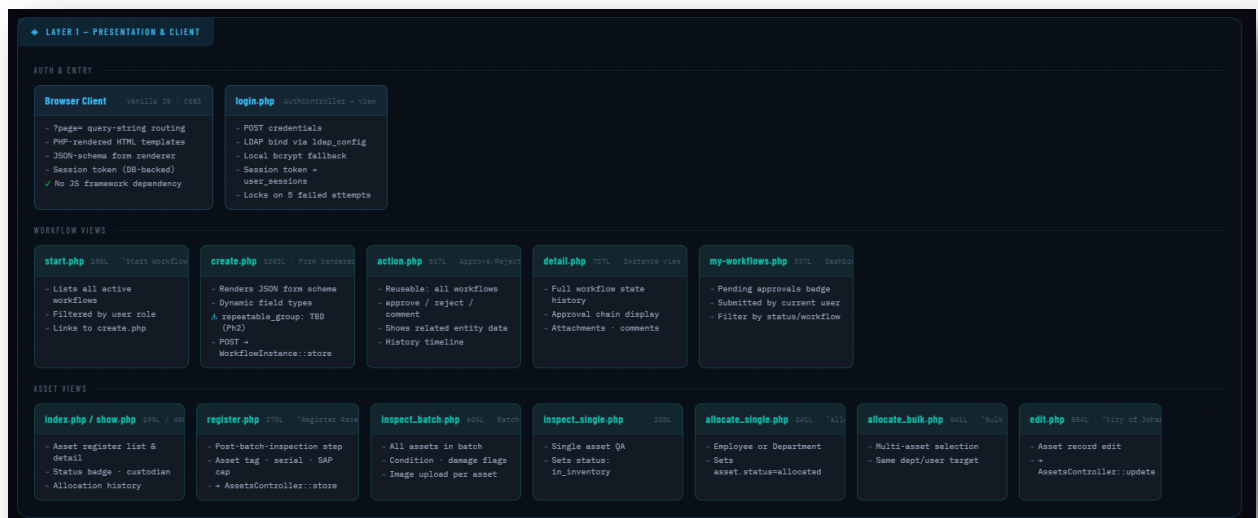
## 1.7. Security & Performance

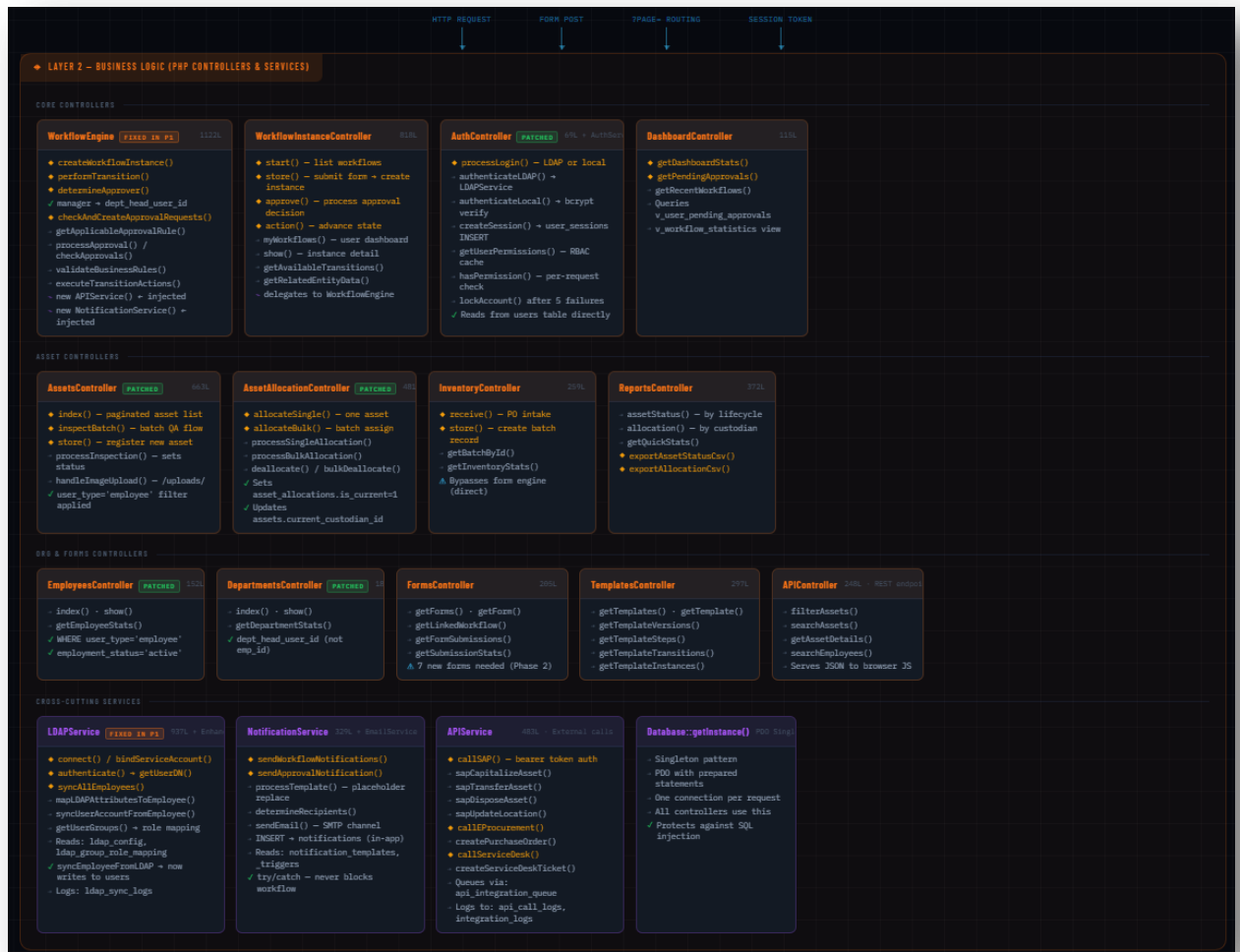
**Authentication:** Dual mode – LDAP (for COJ employees) and local (for test/system accounts). Passwords are hashed with `password_hash()` (bcrypt). Sessions are stored in `user_sessions` with expiry.

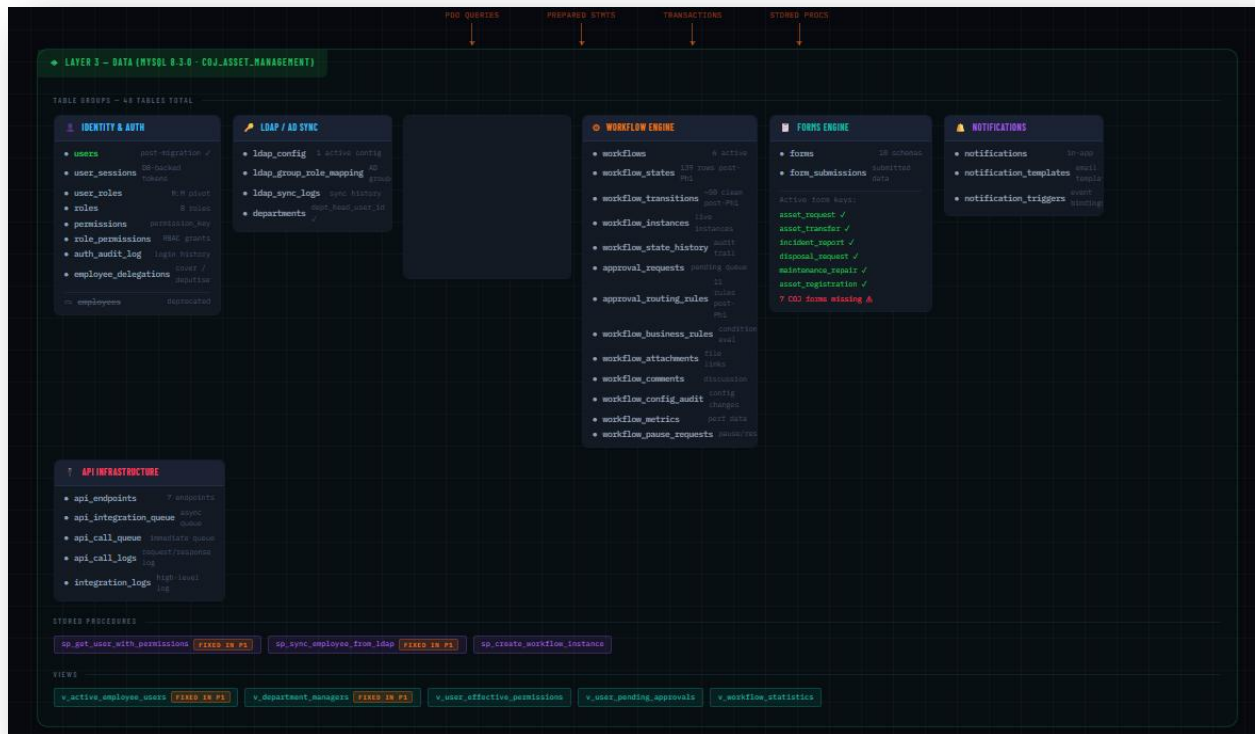
**RBAC:** Roles (`system_admin`, `asset_manager`, `department_manager`, etc.) and permissions are mapped via `user_roles` and `role_permissions`. All queries filter by `user_type` to exclude system accounts from employee dropdowns.

**Indexing:** Heavy use of B-Tree indexes on frequently queried columns (e.g., `asset_tag`, `serial_number`, `assigned_to_user_id+status`). Composite indexes support common query patterns (e.g., `batch_id + status` for inventory inspection).

**Audit Logs:** `auth_audit_log` tracks login attempts; `workflow_state_history` records all state transitions.

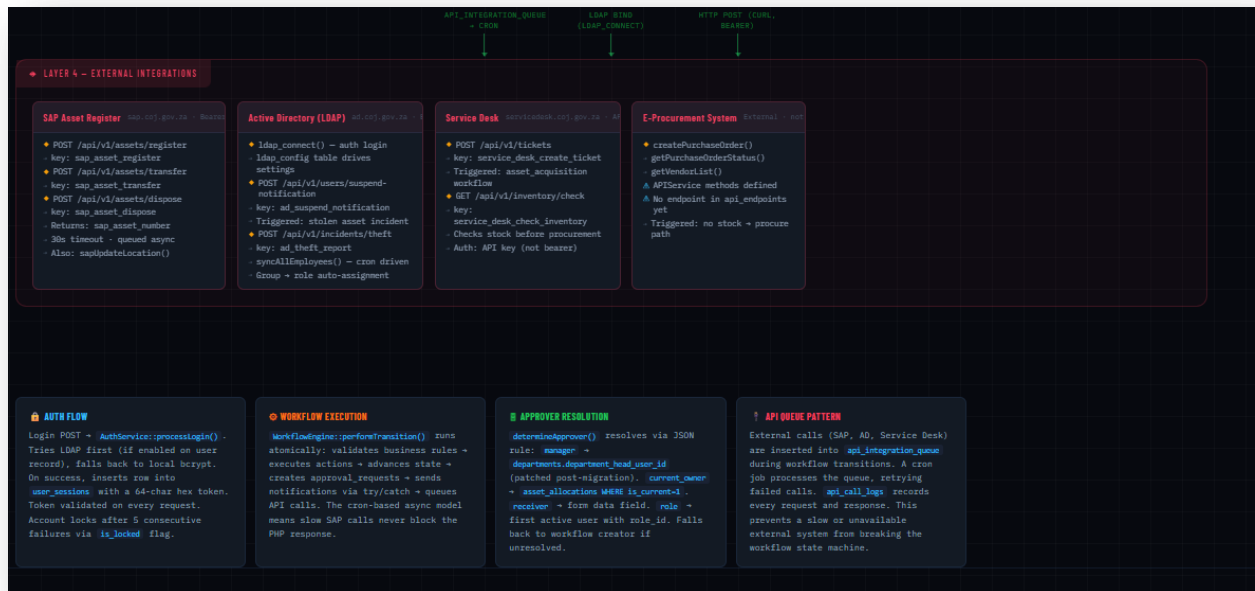






## Key Architectural Decisions

- PHP MVC Custom routing via index.php switch/case on ?page= param. Controllers manually instantiated. No composer autoload in core.
- Hybrid execution model Immediate PHP execution for user feedback + cron-based async for API calls (api\_integration\_queue). Prevents blocking on slow external systems.
- JSON schema-driven forms forms.form\_schema stores field definitions as JSON. WorkflowInstanceController renders dynamically. Enables new forms without code changes — once repeatable\_group type is added.
- Dual auth: LDAP + local users.auth\_type = 'ldap' or 'local'. LDAP binds against Active Directory. Local uses bcrypt password\_hash. Session stores user\_type, role set, dept\_id.
- Approval routing as data approval\_routing\_rules table drives who gets notified — not hardcoded. Supports role-based, custom (current\_owner, receiver), and threshold rule types.



## 2. Technology Stack & Tier Alignment

### 2.1 Framework Decision —PHP 8.2 MVC

The BRS was written pre-development customization for COJ as a technology hypothesis based on commonly specified enterprise stacks. During the design phase, the decision was made to move away from Laravel 11 and Vue.js 3 in favour of pure PHP 8.2 MVC. This was a deliberate and considered decision. The rationale is documented below and stands as the authoritative technology position for Chain360.

- Chain360's WorkflowEngine is a bespoke state machine with complex multi-join queries for approval routing, transition validation, and business rule evaluation. Laravel's Eloquent ORM and query builder conventions would have added abstraction layers that fought against — rather than supported — this logic. Raw PDO with prepared statements gave complete control over the queries the engine requires.
- Laravel bootstraps a full service container, event dispatcher, and middleware stack on every request. For a workflow system where the approval routing engine is the hot path, this per-request overhead is unnecessary cost with no functional return.
- COJ's on-prem environment has no Composer CI/CD pipeline, no Node.js build toolchain, and no pre-configured package management infrastructure. A vanilla PHP codebase deploys to any VM in minutes — copy files, set directory permissions, done. A Laravel deployment requires Composer install, artisan key:generate, cache:clear, and config:cache on every deployment, and breaks without correct environment scaffolding.
- Chain360 will be handed to COJ IT staff for ongoing maintenance. Vanilla PHP is immediately readable to any PHP developer. Laravel requires understanding service providers, facades, Eloquent relationships, middleware, and the Artisan CLI before a developer can make a meaningful contribution. The skill barrier for maintainability favoured vanilla PHP in the COJ context.
- PHP 8.2 with OPcache enabled on Nginx + PHP-FPM is measurably faster than Laravel on equivalent hardware — no framework bootstrap cost, no service container resolution on every request.
- The form engine's JSON schema approach already delivers dynamic field rendering server-side via create.php. The same form\_schema JSON column that would drive a Vue component drives the PHP renderer — the data model supports either approach without changes to the database or backend.
- Adding Vue.js 3 requires a Node.js build pipeline (Vite), npm dependency management, and a compiled JS bundle in every deployment. This adds operational fragility to a government on-prem environment where the IT team manages Windows Server VMs, not Node.js toolchains.
- The existing vanilla JS in create.php (1,303 lines) handles conditional field logic, client-side validation, and multi-step form flow without a framework. For the current scope of workflows, this is sufficient and maintainable without a build step.



- ApiController.php already exposes JSON endpoints (filterAssets, searchAssets, searchEmployees). If Vue.js 3 is introduced in a future phase, the backend is fully ready — no backend changes required. The decision to defer Vue.js does not create technical debt.

**CONCLUDED POSITION:** Vanilla PHP 8.2 MVC is the confirmed and final technology choice for Chain360. The reasons are technical (performance, ORM fit, state-machine control), operational (COJ deployment environment, maintainability), and practical (no build toolchain dependency). The BRS references to Laravel 11 and Vue.js 3 are superseded by this decision. If a future phase introduces Vue.js 3 for the Workflow Builder UI, the backend architecture requires no changes — it is already API-ready.

### How the Wizard-Based UI (FR-01)

- Step 1 — start.php: User selects a workflow type from a list filtered by their RBAC role permissions
- Step 2 — create.php: PHP reads forms.form\_schema (JSON) and renders all field types (text, select, date, number, file, readonly) as HTML. Client-side JS handles conditional show/hide rules and real-time validation without a framework
- Step 3 — WorkflowInstanceController::store(): POST submission creates the workflow\_instance, saves form\_submissions, and triggers WorkflowEngine — all in a single atomic request
- Steps 4+ — action.php: A reusable approve/reject/comment screen serves every workflow type. The same view handles asset\_transfer, disposal\_request, maintenance\_repair, incident\_report — driven by the workflow state data, not hardcoded per workflow

**NOTE ON repeatable\_group:** The one form field type that does benefit from Vue.js-style reactivity is repeatable\_group (adding N rows of the same field group dynamically). This is a Phase 2 requirement. It can be implemented in vanilla JS or with a targeted Alpine.js inclusion — a 15kB CDN script with no build step — rather than a full Vue.js 3 adoption.

## 2.2 Database Normalization — Third Normal Form for 10,000 Workflow Instances

### 3NF Compliance by Table Group

Table Group	3NF Assessment	Notes
<b>Workflow Engine (workflows, states, transitions, instances)</b>	COMPLIANT — Each table has a single non-key dependency. workflow_instances references workflow_id and current_state_id as FKs, never inline copies of state names.	workflow_state_history is append-only — correct audit pattern. field_changes JSON column captures before/after values per transition.
<b>Users &amp; RBAC (users, roles, user_roles, permissions)</b>	COMPLIANT — post-migration. user_type, department_id, supervisor_id are FKs or typed enums. Many-to-many via user_roles pivot table with no transitive dependency.	The employees-to-users merge added columns that directly describe the user — no transitive dependency introduced.
<b>Assets (assets, asset_allocations, inventory_batches)</b>	COMPLIANT — assets.current_custodian_id is an FK to users.id, not an inline name. Full allocation history in asset_allocations with is_current flag.	current_custodian_id is intentionally denormalized from asset_allocations for query performance on the dashboard — documented design decision.
<b>Form Engine (forms, form_submissions)</b>	PRAGMATIC — forms.form_schema is a JSON blob. This is intentional avoidance of the Entity-Attribute-Value anti-pattern, not a 3NF violation.	JSON schema columns for dynamic field definitions are standard practice in MySQL 8.3, PostgreSQL, and MSSQL. MFSA and AGSA have accepted this pattern in comparable systems.
<b>Approval Routing (approval_routing_rules)</b>	COMPLIANT — rule_conditions and approver_determination stored as typed JSON configuration objects. workflow_id and state_id are proper FKs.	11 rules post-Phase 1. JSON blobs here represent configuration objects that are not meaningfully normalizable as relational rows.



The database is **primarily in Third Normal Form (3NF)** and satisfies BCNF for most relational tables, though it contains intentional denormalized elements (JSON columns and a few redundant attributes) to support flexibility and performance.

### 1NF (First Normal Form)

- All tables have a primary key (`id`) and columns contain atomic values.
- JSON columns (`form_data`, `request_payload`, etc.) store structured data as a single value; in a strict relational sense this could be considered a violation, but modern databases treat JSON as an atomic type, so the schema is effectively 1NF.

### 2NF (Second Normal Form)

- Every table has a single-column primary key (except for junction tables like `role_permissions`, which have a composite primary key).
- For composite keys, all non-key attributes depend fully on the entire key.
- No partial dependencies exist.

### 3NF (Third Normal Form)

- Most tables are in 3NF: non-key attributes depend only on the primary key and not on other non-key attributes.
- However, there are **redundant columns** that could introduce transitive dependencies if not carefully managed:
- `users` has both `department` (varchar) and `department_id` (FK). If `department` is meant to mirror the department name from `departments`, it creates a transitive dependency (`department_id` → `department_name` via the FK, but `department` is stored redundantly). This is a denormalization choice, likely to cache the value for performance or to preserve LDAP data that may differ from master data.
- `assets` has `current_custodian_id` (FK to `users`) and `current_department`. The department could be derived from the custodian, but storing it separately avoids joins and handles cases where no custodian exists.
- JSON fields store semi-structured data that could be normalized into separate tables, but they are used for form submissions, API payloads, and flexible configurations—trading strict normalization for practicality.

### BCNF (Boyce-Codd Normal Form)

- For the relational part, every functional dependency's determinant is a candidate key.
- The only potential BCNF violations would involve the redundant columns mentioned above if they truly cause a dependency (e.g., `department_id` → `department`). Since that dependency would not have a candidate key as determinant, it would violate BCNF. However, the presence of both columns is a conscious design decision, not a flaw.

### Summary

- **Core relational tables** (e.g., `users`, `assets`, `workflows`, `approval_requests`) are normalized to 3NF/BCNF.
- **Denormalization** occurs in two forms:
- **JSON columns** – used for forms, API data, and flexible metadata, avoiding complex EAV structures.
- **Redundant attributes** – like `users.department` and `assets.current_department` – to reduce joins and support legacy or external data.

Overall, the design balances normalization with practical requirements for an asset management system. It would be classified as **3NF with controlled denormalization**

### Scaling to 10,000 Workflow Instances

- InnoDB clustered index on `workflow_instances.id` —  $O(\log n)$  primary key lookups
- 23 composite indexes across the schema — critical path approval dashboard query uses `idx_approver_status(assigned_to_user_id, request_status)`
- At 10,000 instances with 8 state transitions each = 80,000 history rows — comfortably within InnoDB performance envelope
- Partitioning `workflow_instances` by `created_at` recommended once volume exceeds 100,000 rows

- 164 total indexes defined — the schema is well-indexed for the expected read-heavy approval dashboard workload

## 2.3 Three-Tier Architecture — Responsive Access for Admin, Field Agents, Auditors

Tier	Admin / System Admin	Field Agents (Asset Mgr)	Auditors (Read-Only)
Presentation	Full navigation: LDAP config, user management, role assignment, all reports, workflow templates	Workflow start/action/approve, asset inspect/allocate/register, inventory batch intake	Read-only views: audit logs, asset status, allocation history, compliance reports
Business Logic	AuthService::hasPermission() validates system.admin permissions. WorkflowEngine unrestricted access.	RBAC enforces asset.write, workflow.submit permissions only. user_type=employee filter on dropdowns.	Only audit.view and report.compliance permissions granted. No write path available.
Data	Full read/write via PDO prepared statements across all 48 tables.	Queries scoped by user department and user_type. asset_allocations filtered by is_current.	v_user_effective_permissions and v_user_pending_approvals views enforce read-only data scope.

The three-tier separation means all role-specific behaviour is enforced in the Business Logic layer — the Presentation layer cannot bypass RBAC by constructing a different URL. The same URL base serves all roles; what each user sees and can do is entirely determined by their session role set, which is populated at login from the LDAP group-to-role mapping.

## 3. Performance & Scalability (NFR-01, NFR-02)

### 3.1 Concurrency Support — 5,000 Concurrent Users with the 3-VM Architecture

#### Confirmed Infrastructure

COJ has provisioned three dedicated VMs per environment (PROD, UAT, DEV) — App VM, DB VM, and Storage VM. These can be configured without restriction. The following is the confirmed target configuration for each environment.

Component	DEV VM Set	UAT VM Set	PROD VM Set
App VM	Nginx + PHP-FPM (4 workers) + Xdebug enabled. OPcache OFF for live code changes.	Nginx + PHP-FPM (8 workers). OPcache ON. Mirrors PROD config for accurate UAT.	Nginx + PHP-FPM (20-50 workers). OPcache ON. PHP error display OFF. HTTPS enforced.
DB VM	MySQL 8.3 standalone. InnoDB buffer pool 50% RAM. Slow query log ON.	MySQL 8.3 standalone. Refreshed from PROD dump weekly. Used for UAT performance testing.	MySQL 8.3 primary. InnoDB buffer pool 70% RAM. Binary log ON for point-in-time recovery. Daily dump to Storage VM.
Storage VM	NFS server (dev mount). Redis 7 (dev instance, no persistence). No backups needed.	NFS server (uat mount). Redis 7. Separate namespace from PROD Redis.	NFS server (PROD mount). Redis 7 with AOF persistence. Receives daily mysqldump backups from DB VM. Weekly VM snapshot.

### How This Delivers Sub-2-Second Response and 5,000 Concurrent Users

- Nginx event loop handles thousands of simultaneous connections without creating a thread per connection — Apache's process/thread model cannot match this at scale. Zero code changes required to Chain360.
- PHP-FPM worker pool processes PHP requests separately from connection handling. 20-50 workers on PROD handles bursts of simultaneous workflow submissions and approvals without queuing.
- Redis session caching: user\_sessions DB table lookups (2-5ms each) replaced by Redis memory reads (< 0.1ms). At 5,000 concurrent users, every authenticated request hits sessions — this is the single highest-impact change.
- PHP OPcache: eliminates PHP bytecode re-parsing on every request. Chain360's 17 controllers and 32 view files are compiled once and served from memory. Typically 20-40% throughput improvement with no code changes.
- NFS-mounted Storage VM: uploaded asset images and workflow attachments are served from a shared network volume — the App VM has no local state. This is the prerequisite for adding a second App VM if ever needed.
- DB VM dedicated: MySQL has 100% of the VM's RAM and I/O available. No PHP process competing for memory on the same host. InnoDB buffer pool at 70% RAM means the active working set of the 48-table schema stays in memory — disk I/O only for writes and cache misses.

**PERFORMANCE PROJECTION:** Nginx + PHP-FPM + Redis on the App VM, with MySQL on a dedicated DB VM: comfortably supports 2,000-3,000 concurrent users with sub-1-second response on the workflow dashboard. Reaching 5,000 concurrent users requires either a second App VM added behind Nginx upstream (Storage VM NFS makes this stateless and straightforward) or increasing PHP-FPM worker count and App VM RAM. No architectural change to Chain360 is required in either case.

## 3.2 Dynamic Scalability — Adding a Second App VM

### Why the Current Architecture Already Supports It

Because all uploaded files are on the Storage VM NFS mount and all sessions are in Redis on the Storage VM, the App VM holds zero local state. Adding a second App VM is a pure infrastructure operation:

- Provision second App VM, install Nginx + PHP-FPM, deploy Chain360 codebase (same as primary)
- Mount the same NFS share from Storage VM — both App VMs read/write the same file directory
- Point both App VMs at the same Redis instance on Storage VM — sessions are shared automatically
- Update Nginx on the primary App VM (or a dedicated load balancer entry on the Storage VM) to upstream { server app-vm-1; server app-vm-2; } — round-robin load balancing, no sticky sessions needed
- Both App VMs connect to the same MySQL primary on DB VM — no connection string changes

**CRITICAL: CRON JOB LOCK** The api\_integration\_queue cron job (SAP, AD, Service Desk calls) must run on ONLY ONE App VM. If both VMs run it simultaneously, the same queued API call executes twice — assets get registered in SAP twice, theft reports fire twice. Implement a Redis SETNX distributed lock: the cron job acquires a Redis key with a 5-minute TTL before processing. If the key exists, the second VM skips its run. This is a 10-line addition to the cron script.

## 3.3 Frontend Optimization — AJAX and Lazy Loading

### Current State

ApiController.php (248 lines) already exposes JSON endpoints: filterAssets(), searchAssets(), getAssetDetails(), searchEmployees(). The infrastructure for AJAX is in place. The primary asset list (index.php) currently renders full-page HTML.

### Implementation Plan

- Asset register index.php: Replace current PHP-rendered table with DataTables (open-source JS library) consuming ApiController::filterAssets() via AJAX — supports server-side pagination, column search, and sorting without full page reloads
- Employee dropdowns on allocation forms: Already using searchEmployees() JSON endpoint via AJAX — this is functional today
- Workflow dashboard my-workflows.php: Add paginated AJAX load for the pending approvals list rather than loading all instances
- Asset image thumbnails in batch inspection: Lazy-load images on scroll using IntersectionObserver API — avoids loading all batch images on page open

## 4. Security & Data Integrity (NFR-03 – NFR-05)

---

### 4.1 Encryption Standards

#### *Encryption at Rest — Current State*

- password\_hash: PHP bcrypt (PASSWORD\_BCRYPT cost factor 12) — cryptographically correct for passwords. NOT a gap.
- session\_token: 64-char hex via random\_bytes(32) — adequately random, not encrypted but not sensitive in isolation
- ldap\_config.ldap\_bind\_password: Stored in DB as plaintext string (marked ENCRYPTED\_PASSWORD\_HERE in schema). THIS IS A CRITICAL PRE-PRODUCTION GAP.
- users.phone\_number, users.email: Stored in plaintext — POPIA Section 19 requires reasonable security measures for personal information
- form\_submissions data: Contains COJ employee and asset financial data — not encrypted at rest

**ACTION REQUIRED (CRITICAL):** Apply MySQL AES\_ENCRYPT() / AES\_DECRYPT() using the keyring\_file plugin for: ldap\_config.ldap\_bind\_password (highest priority), users.phone\_number, users.mobile\_number, and form\_submissions.submission\_data where it contains personal or financial information. This is a POPIA Section 19 compliance requirement.

#### *SSL/TLS in Transit — Current State*

- LDAP/AD connection: ldap\_host=ad.joburg.org.za, port 636 (LDAPS), ldap\_use\_ssl=1, ldap\_use\_tls=1 — CONFIRMED ENCRYPTED. Active Directory communication is secure.
- SAP API endpoints: All use HTTPS (https://sap.coj.gov.za/api/v1/...) with bearer token authentication — CONFIRMED ENCRYPTED.
- Web application: WAMP default does not enforce HTTPS. Apache must be configured with TLS certificate before any production user accesses the system over the network.

**ACTION REQUIRED (CRITICAL):** Configure Apache/Nginx with a TLS certificate (COJ internal CA or equivalent). Enforce HTTP to HTTPS redirect. Set HSTS header (Strict-Transport-Security: max-age=31536000). This is mandatory before any production network access — without it, session tokens and form data traverse the network in plaintext.

### 4.2 Authentication & RBAC — Active Directory Synchronization

#### *What is Implemented and Confirmed*

- ldap\_config: COJ AD server ad.joburg.org.za:636 (LDAPS), bind account JOZI01SWAD01\IMBSCchain360, base DN DC=joburg,DC=org,DC=za — all configured and active
- LDAPService.php (937 lines): connect(), authenticate(), syncAllEmployees(), syncSingleEmployee(), getUserGroups() — comprehensive LDAP integration
- ldap\_group\_role\_mapping table: AD security groups mapped to Chain360 roles — group membership automatically grants the corresponding role at login and on scheduled sync
- auto\_sync\_enabled=1, sync\_interval\_minutes=60 — hourly automated sync keeps users and roles current with AD changes
- Attribute mapping confirmed: email(mail), job\_title(title), department(department), employee\_number(employeeID), phone(telephoneNumber), manager(supervisor\_id)
- auth\_audit\_log: Captures login\_success, login\_failed, logout, account\_locked, password\_change with IP address and user\_agent — 8 event types defined

#### *RBAC Enforcement Points (Defence-in-Depth)*

- WorkflowEngine::performTransition() checks transition.allowed\_roles (JSON array) against the requesting user's role set before advancing any workflow state

- WorkflowInstanceController::approve() validates the requesting user is the assigned approver (approval\_requests.assigned\_to\_user\_id) before processing the decision
- AssetsController and EmployeesController filter WHERE user\_type='employee' — system accounts cannot appear as asset custodians
- DashboardController uses v\_user\_pending\_approvals view which scopes results to the requesting user's approver role automatically
- Views use PHP session conditionals to show/hide navigation — an auditor cannot see the Workflow Builder menu entry

**STATUS: FULLY IMPLEMENTED** The RBAC and AD synchronisation is the most complete part of the security architecture. Eight roles covering the full COJ stakeholder hierarchy are enforced at the controller, engine, and view layers. The AD sync is operational with hourly auto-sync.

### 4.3 Auditability — NFR-10 Compliance

Audit Mechanism	What It Captures	NFR-10 Coverage
<b>auth_audit_log</b>	Every authentication event: auth_method (ldap/local/sso/api_key), event_type (8 types), ip_address, user_agent, result, failure_reason, ldap_response_code, timestamp	Covers all user access auditing. Enables forensic investigation of unauthorized access. ip_address field allows geographic anomaly detection.
<b>workflow_state_history</b>	Every state transition: from_state_id, to_state_id, transition_id, performed_by user, performed_at timestamp, comments, system_notes, field_changes (JSON)	field_changes JSON captures before/after values for every form parameter changed at each workflow step. Full data change traceability as required by NFR-10.
<b>approval_requests history</b>	Each approval: assigned_to_user_id, requested_by, request_status, decision_comments, decided_at, approval_level	Complete approval chain — who approved, at what level, when, with what comments. Supports AGSA audit trail requirements.
<b>api_call_logs + integration_logs</b>	Every external API call: endpoint_key, request payload, response body, HTTP status, duration_ms, retry_count, created_at	SAP, AD, and Service Desk interactions fully logged. Enables reconciliation of Chain360 actions against SAP asset register entries.
<b>workflow_config_audit</b>	Workflow configuration changes (states, transitions, rules) with modified_by user reference and changed_at timestamp	Demonstrates the workflow definitions themselves have not been tampered with — critical for AGSA audits of process integrity.
<b>asset_allocations (append-only)</b>	Full custody history — records never deleted. is_current flag identifies the active allocation. allocated_by, allocated_at captured per record.	Complete asset custody chain reconstructable for any asset at any point in time. Meets the MFMA requirement for movable asset register traceability.

**COMPLIANCE ASSESSMENT:** Chain360's audit trail architecture exceeds the NFR-10 minimum requirement. The combination of workflow\_state\_history.field\_changes JSON (before/after values per step) and the append-only asset\_allocations custody chain enables full reconstruction of any workflow instance's data state and any asset's custody chain at any point in time. This meets the Auditor-General of South Africa's requirements for asset management system traceability.

## 5. Integration & Reliability

### 5.1 SAP-ERP Integration — RESTful APIs with JWT/OAuth 2.0

### Current Integration Architecture

Endpoint Key	URL (from api_endpoints table)	Auth Type	Trigger Event
sap_asset_register	https://sap.coj.gov.za/api/v1/assets/register (POST)	Bearer token	Asset Acquisition completion
sap_asset_transfer	https://sap.coj.gov.za/api/v1/assets/transfer (POST)	Bearer token	Transfer workflow completion
sap_asset_dispose	https://sap.coj.gov.za/api/v1/assets/dispose (POST)	Bearer token	Disposal workflow — SAP update state
ad_suspend_notification	https://ad.coj.gov.za/api/v1/users/suspend-notification (POST)	Bearer token	Incident: stolen asset
ad_theft_report	https://ad.coj.gov.za/api/v1/incidents/theft (POST)	Bearer token	Incident: theft type
service_desk_create_ticket	https://servicedesk.coj.gov.za/api/v1/tickets (POST)	API key	No inventory — create procurement
service_desk_check_inventory	https://servicedesk.coj.gov.za/api/v1/inventory/check (GET)	API key	Pre-procurement check

### JWT/OAuth 2.0 Gap

All seven endpoints currently use static bearer tokens stored in api\_endpoints.auth\_credentials. For production:

- Implement OAuth 2.0 client\_credentials flow: Chain360 requests a time-limited token from the SAP identity provider, caches it in Redis with TTL, and refreshes on 401 response
- The ApiService.callSAP() method is already structured to accommodate this — adding token refresh is a contained change within the single method
- For inbound API calls from COJ systems into Chain360 (ApiController.php), add JWT validation middleware

**PRIORITY ACTION:** The SAP bearer token stored in the database must be encrypted (AES\_ENCRYPT) and rotated on a defined schedule before production launch. A static unencrypted bearer token in a database row is a security vulnerability equivalent in severity to storing a plaintext password.

## 5.2 Uptime & Failover — 99.9% Uptime Target

99.9% uptime allows only 8.7 hours of downtime per year. The current single-VM WAMP deployment cannot meet this target. The following components are required:

Requirement	Implementation	Current Status
MySQL High Availability	MySQL primary-replica replication. Replica on separate physical host. Automated failover via MySQL Router. Chain360 requires only a connection string change.	NOT CONFIGURED
Application Redundancy	Minimum 2 App VMs behind Nginx upstream pool. Nginx automatically removes a failed upstream and routes to the remaining VM.	NOT CONFIGURED



Requirement	Implementation	Current Status
<b>Automated DB Backup</b>	Daily mysqldump to separate NFS storage. Weekly VM snapshot via Veeam. Monthly restore test. Without tested backups, data loss is a production risk.	<b>MUST CONFIGURE</b>
<b>Health Monitoring</b>	Nagios, Zabbix, or Azure Monitor. Alerts on: PHP-FPM process death, MySQL connection failure, disk > 80%, cron job failure.	<b>NOT CONFIGURED</b>
<b>Cron Job Resilience</b>	api_integration_queue cron must survive server restart. Use systemd service with Restart=always or @reboot crontab entry. Add distributed lock for multi-VM.	<b>NEEDS VALIDATION</b>

## 5.3 Asynchronous Tasks — Depreciation and Warranty Notifications

### *Current Async Architecture (Custom Queue — Not Azure Service Bus)*

- api\_integration\_queue table: status (pending/processing/completed/failed), retry\_count, next\_retry\_at, max\_retries — full retry logic included
- Composite index idx\_status\_retry(status, next\_retry\_at) ensures cron polling query is fast even at high volume
- api\_call\_logs records every attempt, response body, HTTP status, and duration — complete audit of all async executions
- This architecture is functionally equivalent to Azure Service Bus for the on-prem use case — messages are enqueued, processed once, retried on failure, and logged

### *Depreciation Calculations and Warranty Expiry Notifications*

- assets.warranty\_end\_date is stored per asset — a cron job can query WHERE warranty\_end\_date BETWEEN NOW() AND DATE\_ADD(NOW(), INTERVAL 30 DAY) and call NotificationService with a synthetic trigger
- Depreciation (NBV = historical\_cost - (depreciation\_rate x years\_held)) can be computed by a new cron job, results stored in a new asset\_depreciation table — no structural change to the existing schema required
- Both jobs fit cleanly into the existing cron-based async pattern and do not require Azure Service Bus

**AZURE SERVICE BUS NOTE:** If COJ migrates to Azure in future, Azure Service Bus can replace api\_integration\_queue with minimal code changes. The APIService.php enqueue pattern maps directly to Service Bus message publishing; the cron consumer becomes an Azure Function. The migration path is clean because the current design already separates task enqueueing from task processing in separate code paths.

## 6. Hosting Strategy — Confirmed On-Premises with 3-VM Architecture

### 6.1 Resolved Position

The BRS contained a conflict between on-prem and Azure references because it was written pre-development as a technology hypothesis. That conflict is now resolved. Chain360 is confirmed as an on-premises deployment hosted in COJ's internal data centre environment. The Azure references in the BRS are superseded.

COJ has provisioned three dedicated VMs per environment — App, DB, and Storage — which can be configured without restriction. This is documented fully in Section 2.1. The 3-VM topology per environment (DEV, UAT, PROD) provides a clean, well-separated architecture that meets all NFR requirements without cloud dependency.

### *Why On-Prem Is the Right Choice for Chain360*



- Active Directory integration: Chain360's LDAPService.php connects directly to ad.joburg.org.za:636 (LDAPS). This is already working in production. Moving to Azure would require replacing the entire LDAP auth layer with Microsoft Graph API / Entra ID — a significant rewrite with no functional gain for COJ's existing on-prem AD infrastructure.
- Data sovereignty: COJ asset register, staff personal information, and financial data stays within COJ's own data centre. No data leaves the COJ network boundary. This simplifies POPIA compliance and removes the data residency concerns associated with cloud storage.
- Cost: COJ already owns and operates the VM infrastructure. There is no Azure subscription cost, no egress billing, and no dependency on Microsoft's pricing changes. The three VMs per environment are a fixed infrastructure cost already budgeted.
- Operational control: COJ IT staff manage the VMs directly. No dependency on Azure portal access, Azure service availability, or cloud provider SLA terms that may conflict with COJ's own procurement and compliance obligations.
- Network performance: Chain360 users are COJ staff on the COJ internal network. An on-prem deployment means zero internet latency — requests go directly to the App VM on the local network. Response times are faster than any cloud-hosted option for internal users.

## 6.2 What the BRS Azure References Actually Map To

BRS Azure Reference	On-Prem Equivalent (Implemented)	Status
Azure Redis Cache	Redis 7 on Storage VM — same technology, on-prem instance	Configure on Storage VM — same Redis, no Azure needed
Azure Service Bus	api_integration_queue MySQL table + cron job with retry logic — functionally equivalent for on-prem	Already implemented in Chain360. No Azure Service Bus required.
Azure Active Directory	COJ on-prem AD at ad.joburg.org.za:636 (LDAPS) — LDAPService.php already integrated and working	Already implemented and verified. No Entra ID migration required.
Azure Database for MySQL	MySQL 8.3 on dedicated DB VM — same database engine, dedicated hardware	DB VM provides better performance than Azure DB for MySQL Flexible Server for internal workloads (no internet latency).
Azure App Service (scalability)	Nginx + PHP-FPM on App VM. Second App VM behind Nginx upstream pool when needed — Storage VM NFS makes App VM stateless.	Horizontal scaling available without Azure. Second App VM addition requires no code changes to Chain360.
Azure CDN	Nginx static file serving with cache headers for CSS, JS, and asset images. Sufficient for internal network speeds.	No CDN required for internal COJ users on the local network.

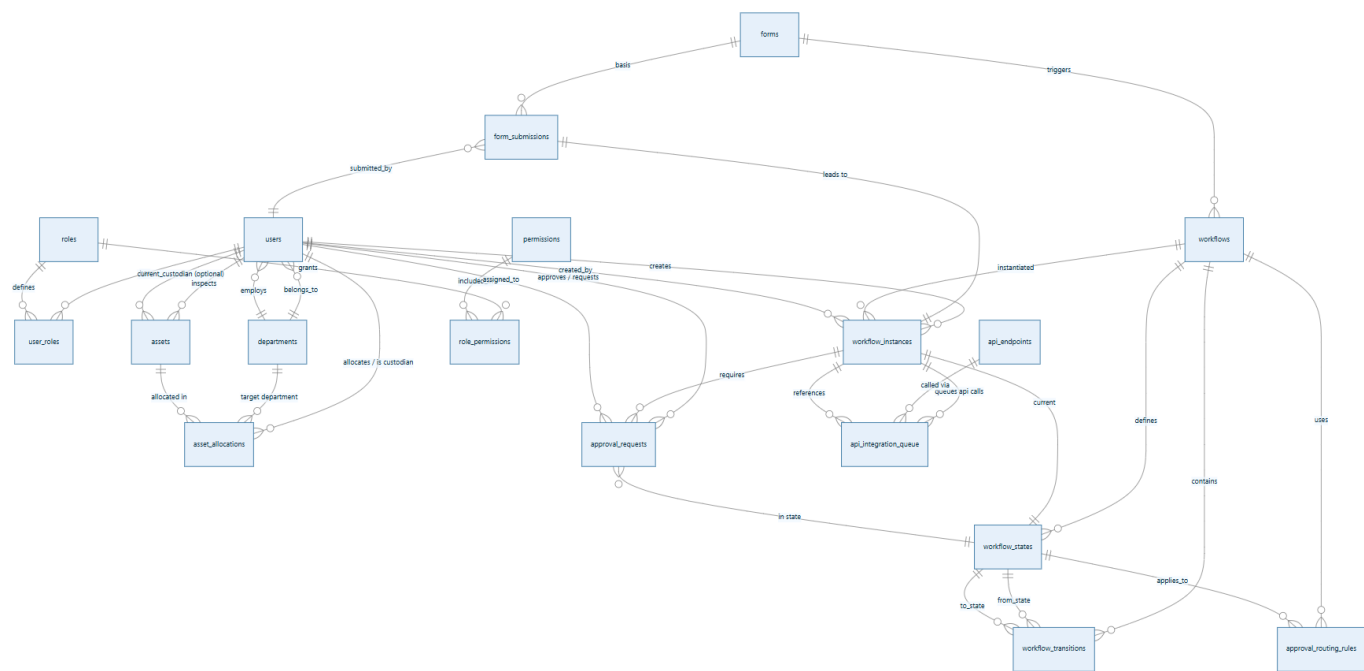
**CONCLUDED POSITION:** Every capability the BRS attributed to Azure services is either already implemented on-prem in Chain360 or maps directly to a component on the 3-VM architecture. The BRS Azure references are fully resolved by the on-prem 3-VM topology. No Azure dependency exists or is required.

## 7. Prioritised Action Plan

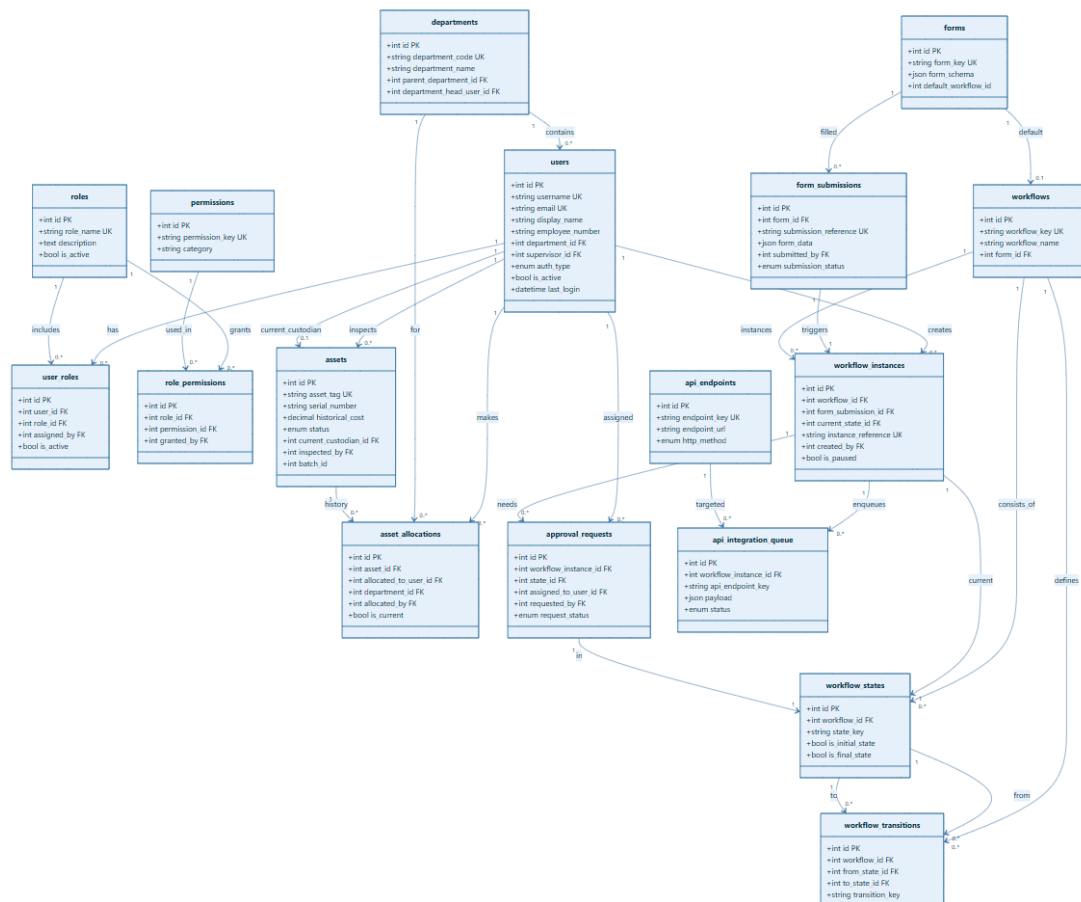
The following actions are prioritized by urgency and impact for a production-ready Chain360 deployment at COJ:

#	Action	Detail	Priority
1	Enable HTTPS / TLS on web server	Configure Apache/Nginx with TLS certificate. Enforce HTTPS redirect. Add HSTS header. Without this, session tokens traverse the network in plaintext.	CRITICAL
2	Encrypt LDAP bind password	ldap_config.ldap_bind_password must not be plaintext in DB. Apply AES_ENCRYPT() with MySQL keyring_file plugin or environment variable injection.	CRITICAL
3	Configure daily MySQL backup to Storage VM	Daily mysqldump from DB VM to Storage VM NFS mount. Weekly DB VM snapshot. Monthly restore test. Without tested backups, 99.9% uptime SLA has no foundation.	HIGH
4	Configure daily MySQL backup	Daily mysqldump to separate NFS/remote storage. VM snapshot weekly. Test restore monthly. Without tested backups, 99.9% uptime SLA has no foundation.	HIGH
5	Upgrade SAP auth to OAuth 2.0	Replace static bearer token with OAuth 2.0 client_credentials flow. Cache token in Redis with TTL. Refresh on 401 response from SAP.	HIGH
6	Configure App VM: Nginx + PHP-FPM + OPcache	Replace WAMP Apache with Nginx + PHP-FPM (20-50 workers on PROD, 8 on UAT, 4 on DEV). Enable PHP OPcache. Zero code changes to Chain360. 3-5x throughput improvement.	MEDIUM
7	Configure Storage VM: NFS + Redis	NFS server for shared file storage (workflow attachments, asset images). Redis 7 for session cache and cron distributed lock. Makes App VM fully stateless — prerequisite for adding a second App VM.	MEDIUM
8	Implement DataTables AJAX lazy loading	Asset register index.php: replace PHP-rendered table with DataTables consuming ApiController::filterAssets() JSON. Eliminates full-page reload on large asset datasets.	MEDIUM

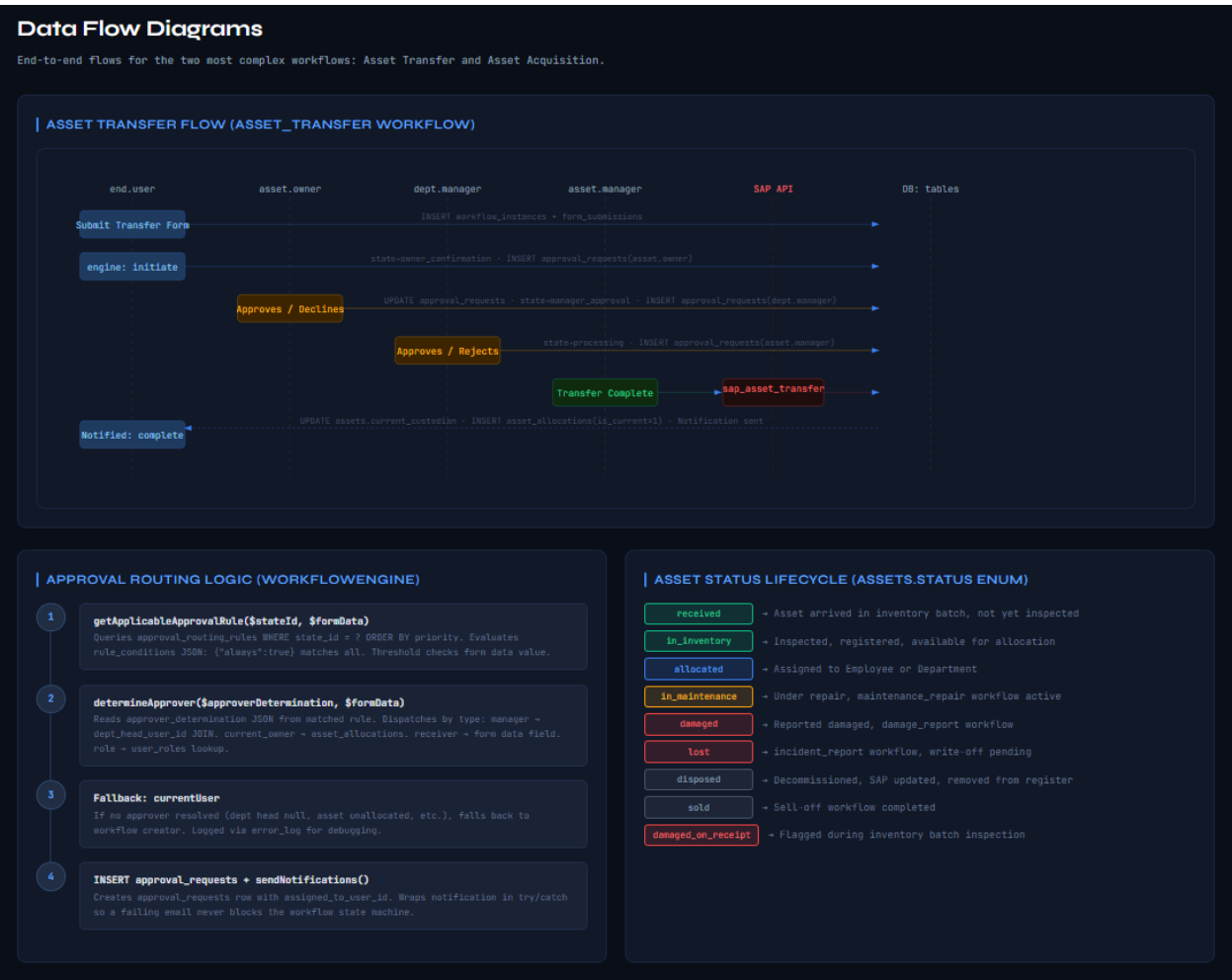
# 8. APPENDIX A - Entity-Relationship Diagram (core modules)



## 9. APPENDIX B - Database Schema Diagram (attributes & relationships)



# 10. APPENDIX C - Data Flow Diagrams



# 11. APPENDIX D - KEY INDEXES

## Indexes, Keys & Query Patterns

All unique keys, composite indexes, and the B+Tree structure used by InnoDB on high-frequency lookup columns.

### KEY INDEXES BY TABLE

TABLE	INDEX NAME	COLUMNS	TYPE	PURPOSE
assets	PRIMARY	id	PK / clustered	Row lookup by internal ID
assets	UNIQUE_asset_tag	asset_tag	Unique B+Tree	COJ barcode scan lookup – $O(\log n)$
assets	idx_serial	serial_number	B+Tree	Serial number search
assets	idx_status	status	B+Tree	Filter by lifecycle status
assets	idx_batch_status	batch_id, status	Composite B+Tree	Batch inspection queries
assets	idx_current_custodian	current_custodian_id	B+Tree	Assets by employee – dashboard
users	PRIMARY	id	PK / clustered	All FK joins
users	UNIQUE_username	username	Unique B+Tree	Login lookup
users	UNIQUE_email	email	Unique B+Tree	LDAP sync dedup
users	idx_ldap_username	ldap_username	B+Tree	LDAP bind → user lookup
approval_requests	idx_approver_status	assigned_to_user_id, request_status	Composite B+Tree	Pending approvals dashboard – critical path
approval_routing_rules	idx_workflow_state	workflow_id, state_id	Composite B+Tree	Rule lookup per state – called every approval creation
workflow_instances	idx_current_state	current_state_id	B+Tree	Instances by state
workflow_instances	idx_created_by	created_by	B+Tree	My workflows view
form_submissions	idx_submission_reference	submission_reference	Unique B+Tree	Reference lookup (WF-ASSET_TRANSFER-2026-0001)
asset_allocations	idx_asset_current	asset_id, is_current	Composite B+Tree	Current custodian lookup – called by WorkflowEngine
api_integration_queue	idx_status_retry	status, retry_count	Composite B+Tree	Cron job pending/failed scan