Bilko Backoffice — Support & Fix Loop Runbook

Bilko Backoffice — Support & Fix Loop Runbook

Scope: Operational runbook for ALAI staff handling live customer problems on `app.bilko.cloud`.
Environment: backend `bilko-api-demo`, frontend `bilko-web-demo`, database `bilko-demo-db` (GCP Cloud Run / Cloud SQL).
Related pages: BookStack 3100 (Backend MVP), BookStack 3104 (Prod Topology).
MC: #103327 (docs), parent #103322 (Support & Fix Loop feature).

---

1. Overview — The Support Loop

The support loop is the full chain from a customer-visible error to a closed ticket:

1. **Error occurs** — The customer hits an accounting error in the browser (e.g. an invoice save fails, a VAT calculation returns 500).
2. **Toast notification** — The frontend renders an error toast with the `errorCode` (e.g. `INFRA_001`) and a **Report this problem** CTA button. The toast carries the `requestId` and `errorCode` from the RFC 7807 ProblemDetail response body — *not* from response headers.
3. **SupportIntakeForm Dialog** — Clicking the CTA opens a focus-trapped `role=alertdialog` Dialog (never embedded in the toast itself). The form pre-fills the `errorCode`, `requestId`, and a `contextBundle` (10 allowlisted fields: IDs and codes, no PII).
4. **Ticket submission** — `POST /support/tickets` creates a row in `support_tickets` (V73 migration). The backend validates the `contextBundle` allowlist server-side before insert. On duplicate `(org_id, request_id)` the API returns HTTP 409 and the form shows "A ticket for this error has already been filed."
5. **Admin queue** — Staff open `/admin/support` (Admin Support Queue page) to see all open tickets, paginated 50 per page, filterable by status.
6. **Triage** — Staff click a ticket to open the detail view (`/admin/support/{id}`), read the `contextBundle`, `customerDescription`, and join to the audit trail by `request_id`.
7. **Impersonate** — From the detail view, staff start a time-limited impersonation session (read-only, reason pre-locked to `support:{ticketId}`) to reproduce the issue in the

customer's org context. All actions during impersonation are audited.
8. **Fix** — Staff apply a fix (DB correction, config change, user education) using safe data-correction procedures.
9. **Status transition** — Staff `PATCH /admin/support/tickets/{id}` to advance the ticket status. Every status change writes an `audit_log` row with the `request_id` threaded through.
10. **Close** — Ticket moves to `RESOLVED` (then optionally `CLOSED`). Both require a `resolutionNote`.

Note: This is a human admin triage queue. There is no automated resolution. No AI agent writes to `triage_json` at MVP — that column is reserved for V2 AI triage scope.

---

2. Component Map

2.1 Sentry Capture — `apps/api/.../plugins/Sentry.kt`

- **DSN guard:** `Sentry.init` is only called when `SENTRY_DSN` env var is non-blank. When absent (local dev, CI, Testcontainers), the SDK is a safe no-op — all `captureException` calls are silently discarded.
- **Cloud Run metadata:** `options.release = K_REVISION` (e.g. `bilko-api-00028-abc`), `options.serverName = K_SERVICE` (e.g. `bilko-api-demo`).
- **beforeSend PII scrub:** `event.request.data` is cleared (strips invoice fields, amounts, emails). `event.breadcrumbs` are cleared. Extra context is filtered to the allowlist: `errorCode, requestId, orgId, httpStatus, instancePath`.
- **OCD-1:** `bilko-sentry-dsn` and `bilko-web-sentry-dsn` secrets are provisioned as empty strings in Secret Manager — CEO action required to populate them. Until populated, Sentry is inert in all environments.

2.2 StatusPages — `apps/api/.../plugins/StatusPages.kt`

- **RFC 7807 ProblemDetail responses:** All error responses emit `Content-Type: application/problem+json` with fields: `type, title, status, detail, instance, errorCode, requestId`.
- **Sentry fires in Throwable catch-all only** (line 237). Named handlers (`BadRequestException`, `ConflictException`, `UnauthorizedException`, etc.) do NOT call `captureException` — this prevents flooding Sentry with 4xx user-error noise. Ktor dispatches named handlers first (exact type or nearest supertype), so Throwable only fires

for genuine INFRA/unexpected exceptions.
- **`requestId` in the response body** comes from `call.callId` (CallId plugin — reads `X-Request-ID` header, generates a UUID when absent). This is the *single canonical source* used consistently across StatusPages, AdminPortalRoutes, ImpersonationService, and SupportTicketRoutes. A mixed source (raw header vs `call.callId`) would produce two different `requestId` values for the same headerless request, breaking the join-by-requestId diagnostic chain.
- **requestId is a BODY field** — the backend does not emit it as a response header. Frontend must read it from the parsed JSON body, not from response headers.

2.3 V72 — `audit_log.request_id`

Migration: `apps/api/src/main/resources/db/migration/V72__audit_log_request_id.sql`

- Adds a nullable `TEXT` column `request_id` to `audit_log`. No NOT NULL constraint — background/internal audit actions have no HTTP request context.
- Partial index `idx_audit_log_request_id` on `(request_id) WHERE request_id IS NOT NULL` for correlation queries. Plain `CREATE INDEX` (not `CONCURRENTLY`) — Flyway wraps migrations in a transaction; `CONCURRENTLY` is prohibited inside a transaction block.
- **Correlation only:** `request_id` is a debuggability handle, NOT a tamper-evidence mechanism. The append-only guarantee comes from the `block_audit_mutation()` trigger (V51).
- **No unique constraint on `(org_id, request_id)`:** one HTTP request legitimately produces multiple audit rows (e.g. impersonation start + org update). The idempotency constraint lives on `support_tickets`, not `audit_log`.

2.4 V73 — `support_tickets` table

Migration: `apps/api/src/main/resources/db/migration/V73__support_tickets.sql`

Column	Type	Notes
`id`	UUID PK	`DEFAULT gen_random_uuid()`
`org_id`	UUID NOT NULL	FK to `organizations.id`, CASCADE DELETE
`user_id`	UUID NOT NULL	FK to `users.id`
`error_code`	TEXT nullable	e.g. `INFRA_001`
`request_id`	TEXT nullable	Correlation ID from originating failed request. NOT a FK to `audit_log.request_id` — join via equality.

Column	Type	Notes
`context_bundle`	JSONB NOT NULL	10-key allowlist: `requestId, errorCode, httpStatus, instancePath, orgId, userId, appRoute, planTier, country, auditRef`. CHECK `jsonb_typeof = 'object'`. Server-side validated.
`customer_description`	TEXT nullable	Free text from customer, min 10 chars (enforced frontend)
`status`	TEXT NOT NULL	CHECK IN (`OPEN, TRIAGED, IN_PROGRESS, RESOLVED, CLOSED`). Default `OPEN`.
`triage_json`	JSONB nullable	V2 AI triage output. NULL = not yet triaged at MVP.
`created_at`	TIMESTAMPTZ NOT NULL	`DEFAULT now()`
`updated_at`	TIMESTAMPTZ NOT NULL	Auto-updated by trigger on BEFORE UPDATE.
`resolution_note`	TEXT nullable	Required for RESOLVED/CLOSED (enforced in route handler)
`external_ref`	TEXT nullable	V2 Zendesk/Linear sync reference. Blank at MVP.

Idempotency: `UNIQUE INDEX idx_support_tickets_org_request_id_unique ON support_tickets(org_id, request_id) WHERE request_id IS NOT NULL` — one customer request produces at most one ticket. Returns HTTP 409 on violation.

Status transitions (enforced in route handler, not DB):

- OPEN → TRIAGED | CLOSED
- TRIAGED → IN_PROGRESS | CLOSED
- IN_PROGRESS → RESOLVED | CLOSED
- RESOLVED → CLOSED
- CLOSED → (terminal, no further transitions)

RLS policies:

- `support_tickets_customer_insert`: `FOR INSERT WITH CHECK (org_id = current_setting('app.current_org_id', true)::uuid)` — prevents any authenticated DB connection from inserting a ticket for another org.
- `support_tickets_customer_select`: customers see only their own org's tickets.
- `support_tickets_admin_all`: platform-admin bypass via `current_setting('app.is_platform_admin', true)::boolean = true`. Must be `SET LOCAL` per transaction (pgBouncer transaction-mode pooling safe — session-level GUC leaks across connections).

- No UPDATE/DELETE policy for customers — deny-by-default after submit. Tickets are never deleted.

2.5 SupportTicketRoutes — `apps/api/.../routes/SupportTicketRoutes.kt`

Endpoint	Auth	Notes
`POST /support/tickets`	JWT (customer)	Creates ticket. `org_id`/`user_id` from `BilkoPrincipal` — NOT from request body. `contextBundle` server-side allowlist validated. Returns `{ id, status: "OPEN" }`.
`GET /admin/support/tickets`	Platform admin	Paginated list. Query params: `limit` (1–100, default 50), `offset`, `status` (optional filter), `orgId` (optional filter). Returns `{ data, meta: { total, limit, offset } }`.
`GET /admin/support/tickets/{id}`	Platform admin	Single ticket detail.
`PATCH /admin/support/tickets/{id}`	Platform admin	Status transition + optional `resolutionNote`, `triageJson`, `externalRef`. Invalid transitions → HTTP 422. Every status change writes an `audit_log` row.

2.6 Frontend — SupportIntakeForm

Source: `apps/web/components/support/SupportIntakeForm.tsx`

- Radix Dialog with `role="alertdialog"` and `aria-modal="true"`. Not embedded in a toast.
- Props: `{ open, onClose, prefill?: Partial<ContextBundle & { customerDescription? }> }`.
- Pre-fills `errorCode` and `requestId` as read-only display. Customer fills `customerDescription` (min 10 chars).
- Assembles `contextBundle` using `buildContextBundle()` from `lib/api-support.ts` — typed explicit field picks from `BilkoApiError` and auth store; never spreads raw error object.
- On HTTP 409: shows inline "A ticket for this error has already been filed." Does not silently retry.

2.7 Frontend — Admin Support Queue

Source: `apps/web/app/(admin)/admin/support/page.tsx` and `[id]/page.tsx`

- Queue page: DataTable with columns Ticket ID, Org, Error Code, Status (colored badge), Created, Actions. Status filter dropdown. Server-side pagination (limit/offset), 50 per page.

Client-side page navigation.
- Detail page: full ticket fields, `contextBundle` parsed and rendered as text-only (no `dangerouslySetInnerHTML`). Status transition controls with `resolutionNote` required for RESOLVED/CLOSED. Impersonation shortcut.
- **Security boundary note:** The admin layout's `platformAdmin` guard in `app/(admin)/admin/layout.tsx` is a client-side UX redirect only. The actual security boundary is the backend `AdminAuthPlugin.kt` which enforces the `isPlatformAdmin` JWT claim on every request.

---

3. Operator Workflow

Step 1 — Find the ticket

1. Navigate to `app.bilko.cloud/admin/support`.
2. Default view shows all tickets newest-first. Use the status filter to narrow to `OPEN` or `TRIAGED`.
3. Note the `errorCode` column. At MVP, most tickets will carry `INFRA_001` (the generic catch-all code) until domain-specific error codes (V2 scope, MC #103333) are deployed.

Step 2 — Read the context bundle and audit trail

1. Click a ticket to open the detail view at `/admin/support/{id}`.
2. The `contextBundle` section shows all 10 allowlisted fields. Key fields for diagnosis:
   - `requestId` — the correlation handle. Use this to join to the audit trail and Cloud Logging.
   - `errorCode` — the error category.
   - `httpStatus` — HTTP status code of the failed request.
   - `appRoute` — the frontend route where the error occurred (e.g. `/invoices/new`).
   - `orgId` — the customer's organization UUID.
3. Join to the audit trail using the `requestId`:

```
-- Diagnostic query: find audit rows for the failing request
SELECT
    id,
    created_at,
    portal_action,
    acting_user_id,
    payload,
```

```
    request_id
FROM audit_log
WHERE request_id = '<requestId from ticket>'
ORDER BY created_at ASC;
```

This query shows every audit event associated with the customer's specific failing HTTP request — impersonation starts, invoice mutations, status changes — in chronological order.

Step 3 — Triage and impersonate

1. On the detail page, scroll to the **Update Status** section. Transition the ticket from `OPEN` to `TRIAGED` to signal the ticket is being investigated.
2. To reproduce the issue in the customer's org context, click **Impersonate Org** in the yellow panel. The impersonation dialog:
   - Pre-fills `reason = support:{ticketId}` — this field is READ-ONLY and cannot be edited.
   - Resolves `orgId` from `ticket.orgId` — NOT the ticketId (backend endpoint `POST /admin/orgs/{orgId}/impersonate` takes the org UUID).
   - Choose a duration (15/30/60 minutes).
3. **Tab isolation warning:** The impersonation token replaces the module-level access token (`_accessToken` in `lib/api.ts`). All tabs in this browser origin are affected for the duration of the session. A banner confirms "Impersonation active — all tabs in this origin are affected."
4. All actions during impersonation are audited in `audit_log` with `reason = support:{ticketId}`.

Step 4 — Apply a fix

For data corrections, see Section 4 (Data-Correction Safety). For configuration or code fixes, follow the standard ALAI deployment pipeline (DEPLOY-MAP.md). After the fix is applied, end the impersonation session using the **End Impersonation** button. This atomically:

- Calls backend `POST /admin/impersonate/end`
- Clears `setAccessToken(null)` (module-level token)
- Removes all three sessionStorage keys (`bilko_impersonation_token`, `bilko_impersonation_org`, `bilko_impersonation_expires`) in a single synchronous block

Step 5 — Transition and close

1. Return to the detail view. Transition status to `IN_PROGRESS` (if work is ongoing) or `RESOLVED` (if fixed).
2. **RESOLVED and CLOSED both require a `resolutionNote`** (enforced by the API — HTTP 422 if blank). Write a brief note describing what was fixed.

3. The PATCH call audits the status change in `audit_log` with the admin's `request_id` threaded through for correlation.
4. Ticket moves to `CLOSED` as the final terminal state. No further transitions are possible.

---

4. Data-Correction Safety

Impersonation scope

- Impersonation is RLS-scoped: the impersonation token sets `app.current_org_id` GUC to the target org's UUID via `SET LOCAL` (pgBouncer transaction-mode safe).
- All DB operations during impersonation run under the target org's RLS policies — the admin cannot access other orgs' data even if they craft a direct API call.
- All impersonation actions are audited. The reason field (`support:{ticketId}`) is locked and stored verbatim in `audit_log`.

Direct data corrections (raw SQL)

If a data correction requires direct SQL on the database (e.g. correcting a journal entry double-entry imbalance, fixing a corrupted exchange rate):

1. **Run the preflight script before any SQL on prod/demo:**

   ```
   bash scripts/ops/bilko-support-fix-preflight.sh <orgId> <ticketId>
   ```

   This script: creates a point-in-time backup annotation, logs the operator identity and timestamp, and prints a confirmation token required for the correction script.

2. **Never run raw SQL on `bilko-demo-db` without the preflight backup pattern.** Cloud SQL supports point-in-time recovery to 7-day window.
3. Double-entry corrections must preserve the ledger balance — every credit must have a matching debit. The platform enforces `NUMERIC(19,4)` for all monetary amounts.
4. After any direct SQL correction, run a reconciliation query to verify the affected org's trial balance still balances.

---

5. Cloud Logging — Saved Views

Three saved log views are available in GCP Cloud Logging for the `bilko-api-demo` service:

View Name	Purpose	Key Filter

`bilko-error-by-org`	All ERROR/CRITICAL log entries for a specific org, newest-first. Use when the customer provides their org UUID and you need to see all errors in context.	`resource.type="cloud_run_revision" severity>=ERROR jsonPayload.orgId="<orgId>"`
`bilko-request-trace`	All log entries for a specific `requestId` across all severity levels. Gives the complete request lifecycle: auth, route entry, DB queries, response. Use after reading the `requestId` from the support ticket's `contextBundle`.	`resource.type="cloud_run_revision" jsonPayload.requestId="<requestId>"`
`bilko-5xx-demo`	All 5xx responses from `bilko-api-demo` in the last 24 hours. Use for proactive triage — catch errors before customers report them.	`resource.type="cloud_run_revision" resource.labels.service_name="bilko-api-demo" httpRequest.status>=500`

To use these views: GCP Console → Cloud Logging → Log Explorer → Saved Queries. Select the relevant view, substitute the variable in angle brackets with the value from the support ticket.

Correlating a support ticket to logs:

1. Take the `requestId` from the ticket's `contextBundle` (or from the `request_id` field on the ticket row).
2. Open `bilko-request-trace` view, paste the `requestId` into the filter.
3. You will see the full request lifecycle including the Kotlin `[SupportTickets]` structured log line that confirms when the ticket was created.

---

6. Known Gaps and Follow-ups

Gap	Description	Tracking
Sentry DSN not provisioned	`bilko-sentry-dsn` and `bilko-web-sentry-dsn` secrets exist in Secret Manager but are empty. Sentry is inert (no-op) in all environments until the CEO provisions the Sentry project and populates the secrets. Until then, rely on Cloud Logging for error observability.	OCD-1 (open CEO decision)
Error-code taxonomy V2	Most support tickets at MVP will carry generic codes (`INFRA_001`, `VAL_001`) because domain-specific codes (e.g. `BILKO-VAT-001`, `BILKO-FISK-001`) are not yet defined. Triage is partly blind until V2 error codes land. Staff should use the `appRoute` and `httpStatus` from the `contextBundle` alongside the generic code to narrow the domain.	MC #103333 (V2 error code taxonomy)

Gap	Description	Tracking
Backend hardening follow-on	Server-side `context_bundle` value length constraints (max ~256 chars per key, charset ASCII printable) are a follow-on hardening item. Current implementation enforces key allowlist but not value length.	MC #103338 (backend hardening)
Sentry `setTag('httpStatus')` minor follow-on	The Throwable catch-all in `StatusPages.kt` sets scope tags `requestId`, `orgId`, and `errorCode`. The `httpStatus` tag is not yet set — the Sentry `beforeSend` filter uses it to suppress 4xx events, but for the INFRA_001 catch-all (always 500) this is a minor gap only. A one-line follow-on to add `scope.setTag("httpStatus", "500")` in the catch-all handler is tracked as a minor improvement.	Follow-on to MC #103323
Admin portal flash (middleware.ts)	`middleware.ts` checks only for cookie presence (`bilko_refresh_token` or `bilko_auth` marker), not for the `platformAdmin` JWT claim. A non-admin authenticated user who navigates directly to `/admin/support` will see admin HTML briefly before the client-side `useEffect` redirect fires. The real security boundary is the backend — `AdminAuthPlugin.kt` enforces the claim on every API call. UI flash is a UX gap open for CEO decision.	OCD (open CEO decision)
Customer-facing ticket status	Customers have no way to view the status of their submitted ticket or receive a notification when it is resolved. The backend `POST /support/tickets` returns the ticket ID; a customer-facing `GET /support/tickets` endpoint and notification flow are deferred to V2.	V2 scope

---

Published by Skillforge (MC #103327). Complements BookStack 3100 (Backend MVP) and BookStack 3104 (Prod Topology). Last updated: 2026-06-11.

Revision #2
Created 2026-06-11 20:06:33 UTC by John
Updated 2026-06-11 20:06:40 UTC by John