Warranty System

1. High-level overview (one-sentence)

Each *sold item* gets a single WarrantyRecord (unique warranty). A QR on the invoice/product resolves to that record; claims create append-only WarrantyEvents; supplier responses are recorded as SupplierActions; the system enforces expiry rules, inspection workflows, supplier linking, and financial reconciliation.

2. Data model (complete fields + relationships)

Primary tables

warranty_records

- id (PK)
- warranty_code (unique, human-friendly e.g. W-2025-0000123)
- warranty_token (GUID or short token used in QR)
- invoice_id (FK)
- invoice_item_id (FK, nullable)
- product_id (FK)
- serial_number (nullable)
- customer_id (FK)
- supplier_id (FK) — supplier who supplied the item to you
- purchase_date (date)
- warranty_period_months (int)
- warranty_start_date (date) — usually purchase_date
- warranty_expiry_date (date) — computed
- current_status (enum): active, expired, claimed, under_inspection, replacement_pending, replaced, refunded, declined, closed
- replaced_by_warranty_id (nullable FK) — links to new warranty record if replacement issued
- transferable (bool) — optional
- created_by, created_at, updated_by, updated_at

warranty_events

- id (PK)
- warranty_id (FK)
- event_type (enum): claim_registered, claim_updated, inspection_started, inspection_passed, inspection_failed, supplier_action_requested, supplier_action_recorded, replacement_issued, cash_refund_issued, manual_verification, escalated

- actor_type (enum): customer, staff, supplier, system
- actor_id (nullable) — user/supplier id
- note (text)
- attachments (JSON/array of file refs)
- meta (JSON) — structured details (images metadata, RMA id, estimated repair cost)
- created_at

supplier_actions

- id (PK)
- warranty_id (FK)
- supplier_id (FK)
- action_type (enum): replacement_sent, repair_sent, cash_refund_offered, declined, partial_refund
- action_details (JSON): {replacement_serial, replacement_warranty_period_months, amount, supplier_invoice_id, tracking_number}
- status (enum): pending, confirmed, rejected
- created_at, updated_at

audit_log (optional but recommended)

- logs all API calls/role actions with before/after snapshots.

financial_transactions

- When refund/cash replacement occurs, store a transaction linking to accounting module.

3. QR / token design & security

Do **not** encode PII. Use a short, opaque token printed as a QR.

Options:

- **GUID token**: QR contains https://erp.example.com/w/{token}. Backend resolves token -> warranty_record and performs signature/time checks.
- **Signed token**: QR contains a small signed string: base64(payload).sig where payload includes warranty_id and issued_at. Use HMAC-SHA256 with server secret and timetoLive huge (or unlimited) — signature prevents casual tampering.

Recommended: use GUID + short HMAC to detect forged tokens:

QR payload: <token>=<guid>.<sig>
where sig = base64url(HMAC(secret, guid|issued_at))

Validation steps on resolve:

1. Verify signature and token format.
2. Fetch warranty by token GUID.
3. Verify not soft-deleted.
4. Return only allowed fields (no unneeded PII) unless proper auth.

QR size targets: <300 bytes total. Use base62 or base64url for compactness.

4. State machine (statuses & transitions)

Statuses and allowed transitions:

- active (default after sale)
  - -> expired (on expiry_date pass)
  - -> claimed (on customer claim)
- claimed
  - -> under_inspection (when staff starts inspection)
  - -> declined (if claim invalid)
  - -> replacement_pending (if supplier will provide replacement)
  - -> refunded (if cash refund issued)
- under_inspection
  - -> inspection_passed (internal/alias: replacement_pending or refunded)
  - -> inspection_failed (alias: declined)
- replacement_pending
  - -> replaced (on replacement received)
  - -> declined (if supplier rejects)
- replaced
  - -> closed
- refunded
  - -> closed
- declined
  - -> closed

Always append warranty_events for each transition.

5. End-to-end flows (detailed)

A. Purchase flow (system-generated)

1. On invoice finalization, for each warranty-eligible item:
    - Create warranty_record with computed expiry_date = add_months(purchase_date, warranty_period_months).
    - Generate token + QR and store warranty_token.
    - Optionally send warranty activation mail/SMS to customer with QR link.
    - Print QR on invoice or product sticker.

B. Customer scan & resolve (public)

1. Customer scans QR → hits GET /api/warranty/resolve?q=token.
2. Backend validates token and returns summary:
    - product name, purchase date, expiry date, warranty status, allowed actions (e.g., can_claim: true).
    - If user unauthenticated: limit PII; show Register Claim button with T&C.
3. Customer clicks Register Claim:
    - Submit POST /api/warranty/{id}/claim with reported_issue, images[], contact.
    - Backend creates warranty_event claim_registered, sets current_status = claimed (if active/eligible), responds with claim id.

Rules: If warranty expired, show ineligible but allow manual_verification submission (staff can accept after review).

C. Staff inspection flow

1. Staff gets notification / sees new claims dashboard.
2. Staff opens claim → POST /api/warranty/{id}/inspection/start (event: inspection_started).
3. Staff inspects physically (photos, serial checks, invoice ownership checks).
4. Staff records result with POST /api/warranty/{id}/inspect:
    - result: valid or invalid
    - notes, attachments, estimated_repair_cost
    - If valid: update current_status to under_inspection → then create supplier_action_request (if supplier-supplied)
    - If invalid: set current_status = declined and append event inspection_failed.
5. Staff triggers supplier communication if applicable (email/RMA creation).

D. Supplier action & reconciliation

1. System calls supplier portal/email with supplier_action_request (include claim id, photos, RMA id).
2. Supplier responds: replacement, repair, cash_refund, or declined.
3. Record supplier_action row with action_type and action_details.
4. Update warranty_events with supplier_action_recorded.
5. If replacement:
    - If supplier sends replacement item with serial R-SN-1, then create a new warranty_record for replacement or link replacement warranty.
    - Set original warranty_record.current_status = replaced, link replaced_by_warranty_id to new warranty.
    - Optionally set warranty period for replacement (e.g. remaining time or new period).
6. If cash refund:
    - Create financial_transaction and mark warranty_record.current_status = refunded.
    - Reconcile with accounting module.
7. If supplier declines:
    - Set current_status = declined and create an escalation workflow (SLA + internal replacement policy).

E. Customer notification & closure

- Notify customer at each key event: claim received, inspection result, supplier action, replacement sent, refund processed, closed.
- After final action (replaced, refunded, declined), mark closed after optional cooldown and append final warranty_event.

6. Business rules / validations (explicit list)

- **Eligibility**: today <= warranty_expiry_date OR claim_created_at within grace_window (configurable).
- **Proof of purchase verification**: verify invoice id or serial_number vs invoice. If not found: route to manual_verification.
- **Ownership check**: if the requesting customer ID/email != invoice.owner -> require extra verification or reject.
- **Duplicate claim prevention**: same customer + same product + same issue within X days → block or flag.
- **Inspection mandatory**: claims automatically go to under_inspection if product_value > threshold or serial_number present.
- **SLA**:
    - Staff must start inspection within 24 hours of claim (configurable).
    - Supplier must respond to action request within 72 hours.

    - If SLA breach, escalate to manager and optionally auto-issue internal replacement/refund.
- **Financial authorization**: only finance role can approve cash refund; staff may recommend.

7. API contracts (concise JSON schemas)

GET /api/warranty/resolve?q=token
Response:

```
{
  "warranty_id": 123,
  "warranty_code": "W-2025-0000123",
  "product": {"id": 321,"name":"Model X"},
  "purchase_date":"2025-05-15",
  "expiry_date":"2025-11-15",
  "status":"active",
  "can_claim": true,
  "allowed_actions":["register_claim"],
  "links": {"invoice":"/invoices/1234"}
}
```

POST /api/warranty/{id}/claim
Request:

```
{
  "customer_id": 555,
  "reported_issue": "screen cracked after fall",
  "images": ["s3://...","s3://..."],
  "contact_phone": "+8801....",
  "preferred_action": "replacement" // optional
}
```

Response: 201 with claim_event_id and next_steps.

POST /api/warranty/{id}/inspect
Request:

```
{
  "staff_id": 10,
  "inspection_result": "valid",
  "notes":"physical damage consistent with customer description",
  "estimated_cost": 1500,
  "attachments": ["s3://..."]
```

}

POST /api/warranty/{id}/supplier-action
Request:

```
{
  "supplier_id": 77,
  "action_type":"replacement_sent",
  "action_details": {"replacement_serial":"R-SN-0001","tracking":"TR123"}
}
```

All responses include event_id and new status.

8. Concurrency, idempotency & transactions

- Wrap multi-step operations in DB transactions (e.g., inspection result + supplier_action creation + status update).
- Use idempotency keys for external calls (customer claim submissions, webhook responses) to avoid duplicates.
- When supplier notifies via webhook, validate signature and check if event already processed by external_id stored in supplier_actions.action_details.external_id.
- Use optimistic locking on warranty_records updates (e.g., version field or compare updated_at) to prevent race conditions.

9. Security & privacy

- Enforce RBAC:
  - customer: view own warranty, file claim (with evidence)
  - staff: view/inspect claims, create supplier actions, escalate
  - supplier: view only linked claims and act on them (via supplier portal)
  - finance: approve refunds
- Sanitize attachments and scan for malware.
- Throttle public resolve endpoint to avoid scraping.
- Store attachments in private S3 with signed URLs for short durations.
- Encrypt sensitive fields at rest if required.
- Log all access; maintain audit trail for compliance.

10. Fraud mitigation & heuristics

- **Invoice/serial mismatch** → flag as suspected_fraud.
- **Multiple claims** from same customer within short timeframe → flag and require manager approval.

- **High refund amounts** require dual-approval (staff + finance).
- **Geo check**: claim location far from purchase location → require extra verification.
- **Device fingerprinting** for reuse of QR across many claims to detect bulk fraud.
- Provide a machine learning or rule-based scoring field fraud_score and automatic gating over threshold.

11. UI / UX detailed behavior

QR Resolve page (public)

- Show: Product image, brief specs, invoice id (masked), purchase date, expiry date, remaining days, current status, history timeline (condensed).
- If eligible: show Register Claim CTA. Ask for short issue category and 3 photos minimum for damage.
- If not eligible: show reason and Request Manual Review option.

Staff Dashboard

- Filters: new claims, under_inspection, awaiting_supplier, SLA-breached.
- Quick actions: Start inspection, Request supplier action, Escalate, Approve refund (if role).
- Timeline per claim with inline attachments.

Supplier Portal

- Supplier sees claims assigned to them with photos and RMA id, can create supplier_action (replacement/tracking/refund) and upload invoice.

Notifications

- Email/SMS/WhatsApp templates for:
  - claim received
  - inspection started/completed
  - supplier action taken
  - replacement shipped
  - refund processed
  - claim closed

12. Reporting & KPIs

Track and report:

- claims_per_month, claims_by_product, claims_by_supplier
- average_time_to_inspect, average_supplier_response_time, SLA_breach_rate
- refund_amounts_total, replacement_rate (% of claims resulting in replacement)
- fraud_rate (flagged / total)
- Dashboards and scheduled reports with CSV export.

13. Edge cases & how to handle them

- **Lost QR**: allow claim by invoice number + customer verification (OTP to customer phone/email).
- **Third-party reseller**: if product sold by reseller, link original supplier via supplier_id and treat warranty per your policy.
- **Warranty transfer**: allow owner change flow with owner_history and require proof.
- **Partial damage**: staff records partial_repair and system calculates partial refund or repair workflow.
- **Supplier unresponsive**: after SLA, auto-escalate and optionally apply internal replacement/refund policy; log all steps.
- **Stolen serials/cloned tokens**: tokens are signed; require serial verification and purchase matching to accept claim.

14. Testing checklist (unit, integration, manual QA)

- Unit tests:
  - expiry calculation across month boundaries and leap years
  - token signing/verification
  - state transitions allowed & forbidden
  - duplicate claim prevention checks
- Integration tests:
  - end-to-end claim → inspection → supplier → replacement
  - supplier webhook idempotency
- Manual QA:
  - QR scan with damaged network (offline capture then sync)
  - UI for expired warranties
  - Fraud scenarios (mismatched invoice/customer)
- Load tests:
  - resolve endpoint under scanning campaigns (throttle)
  - attachments upload throughput
- Security tests:
  - RBAC checks, ACL bypass attempts, injection testing on attachments metadata

15. Deployment & integration notes

- DB migrations to add tables + indices on warranty_token, warranty_code, customer_id, supplier_id, and current_status.
- Background workers:
  - SLA monitor job (cron) — sends escalation notifications for overdue supplier responses.
  - Notification job queue for emails/SMS.
- Integrate accounting module for refunds (atomic transaction linking supplier action -> financial transaction -> ledger entry).
- Backups and retention: attachments retention policy, archival of old warranty_events after X years (comply with local laws).

16. Metrics/limits/configuration (tunable settings)

- default_warranty_months (global)
- grace_period_days (after expiry)
- inspection_start_deadline_hours (default 24)
- supplier_response_deadline_hours (default 72)
- fraud_threshold_score
- max_attachments_per_claim
- max_image_size_mb

Store these in a system_settings table or application config.

17. Sample SQL migration (Postgres)

```
CREATE TABLE warranty_records (
  id BIGSERIAL PRIMARY KEY,
  warranty_code VARCHAR(64) UNIQUE NOT NULL,
  warranty_token UUID UNIQUE NOT NULL,
  invoice_id BIGINT NOT NULL,
  invoice_item_id BIGINT,
  product_id BIGINT NOT NULL,
  serial_number VARCHAR(128),
  customer_id BIGINT,
  supplier_id BIGINT,
  purchase_date DATE NOT NULL,
  warranty_period_months INT NOT NULL,
  warranty_start_date DATE NOT NULL,
  warranty_expiry_date DATE NOT NULL,
  current_status VARCHAR(32) NOT NULL DEFAULT 'active',
  replaced_by_warranty_id BIGINT,
  transferable BOOLEAN DEFAULT FALSE,
  created_by BIGINT,
  created_at TIMESTAMPTZ DEFAULT now(),
  updated_at TIMESTAMPTZ DEFAULT now()
);
CREATE INDEX idx_warranty_token ON warranty_records (warranty_token);
```

And create warranty_events and supplier_actions per earlier schema.

18. Example functions/pseudocode

Check eligibility (pseudo-Python):

```
def is_eligible_for_claim(warranty, claim_date):
    if warranty.current_status in ['refunded','replaced','closed']:
        return False, "Warranty closed"
    if claim_date <= warranty.warranty_expiry_date:
        return True, "Within warranty"
    if config.grace_period_days and (claim_date - warranty.warranty_expiry_date).days <= config.grace_period_days:
        return True, "Within grace period"
    return False, "Expired"
```

Process supplier webhook (idempotent):

```
def handle_supplier_webhook(payload):
    external_id = payload['external_id']
    existing = db.find_supplier_action_by_external_id(external_id)
    if existing:
        return existing  # idempotent
    # validate signature
    if not verify_signature(payload):
        raise Unauthorized
    with db.transaction():
        sa = db.insert_supplier_action(...)
        db.update_warranty_status(...)
        db.insert_warranty_event(...)
    return sa
```

19. Rollout & migration strategy (if adding to live ERP)

1. Add migrations and API endpoints behind feature flag.
2. Seed warranty_records for new invoices only; optionally backfill for recent invoices using a batch job.
3. Expose QR generation to POS printing module.

4. Soft-launch with internal staff to test process (use test supplier responses).
5. Open to customers after first 2 weeks of internal tests.

20. Bonus: ERPNext implementation tips (if you use ERPNext)

- Create a custom DocType Warranty Record with events child table.
- Hook into on_submit of Sales Invoice to create records automatically.
- Use REST API to serve QR resolution pages; customize Desk for staff workflows.
- Use ERPNext's Notification and Auto Email Report for SLA and event notifications.

21. Recommended next concrete deliverables I can produce immediately

(choose any; I'll generate it right away)

- Full SQL migration + seed script for Postgres.
- Node.js (Express + Knex/TypeORM) controller set for all API endpoints with authentication middleware.
- React components: QR-resolve page + claim wizard + staff inspection panel (tailwind).
- Supplier webhook handling module with idempotency helpers.
- ERPNext custom app code skeleton (Python + DocType definitions).