IMS Solution Design — Patch v1 → v2

Module: Raw Materials — GP Coil / GI Coil / Steel Sheets
Changes applied: Database → MongoDB · Computed fields · Embedded history · FSM rework · DEPRECATED confirmation flow
Date: 2 April 2026
Supersedes: Sections 3.1, 5, 6, 7, 8 of `IMS_RawMaterial_SolutionDesign_v1.md`

Change Summary

#	What changed	Impact
1	Database changed from PostgreSQL → MongoDB	Schema, queries, ORM, all rewritten
2	`CoilRecord` updated — computed fields, embedded history, `supplierName`	Types file updated
3	SCHEDULED status removed from FSM	OPEN loops to itself; `orderRef` field replaces status
4	DEPRECATED — manual only, weight alert, reason required	New confirmation flow defined
5	SOLD — manual only, weight warning if remainder > 0	New confirmation flow defined
6	`weight_ledger` and `movement_log` tables removed	History embedded in coil document

---

1. Updated CoilRecord Fields

New fields added

Field	Type	Formula / Notes
`averageWeightPerFoot`	`number`	`(originalWeight_kg / length_m) / 3.28` — kg per foot of coil length
`realThickness_mm`	`number`	`(width_mm / averageWeightPerFoot) / 0.305 / 0.00785` — cross-checks mill-stated thickness

Field	Type	Formula / Notes
`weightDiff_kg`	`number`	`originalWeight_kg - currentWeight_kg` — derived, stored on document
`weightDiffPercent`	`number`	`(weightDiff_kg / originalWeight_kg) × 100`
`supplierName`	`string`	Free text (e.g. "Tata Steel", "JSW") — replaces `supplierId` foreign key
`warehouseName`	`string`	Denormalised for display — no join needed on list view
`orderRef`	`string | null`	Replaces SCHEDULED status — coil stays OPEN, order number stored here
`history`	`CoilHistoryEntry[]`	Embedded array — full lifecycle record on the document

`CoilHistoryEntry` shape

Each entry written on every weight change or status event:

typescript

```typescript
interface CoilHistoryEntry {
  clientName:          string;       // client, job, or internal reference
  kg:                  number;       // weight moved (negative = consumed)
  orderNumber:         string;       // sales/job order or PO number
  newCurrentWeight:    number;       // coil weight AFTER this operation (kg)
  editorName:          string;       // full name of user who performed the action
  timestamp:           Date;         // UTC
  movementType:        MovementType; // GOODS_IN | JOB_ISSUE | SALE_DISPATCH | ...
  machine?:            string;       // for JOB_ISSUE entries
  operator?:           string;       // for JOB_ISSUE entries
  depreciationReason?: string;       // required when movementType = DEPRECIATION
}
```

Computed field calculation

Computed fields are calculated once in `CoilService` and stored on the document. They are NOT recalculated per query — they are updated only when `originalWeight_kg`, `currentWeight_kg`, `length_m`, or `width_mm` changes.

```typescript
// averageWeightPerFoot
const avgWtPerFoot = (originalWeight_kg / length_m) / 3.28;

// realThickness_mm — uses width and density of steel (0.00785 kg/cm³)
const realThickness_mm = (width_mm / avgWtPerFoot) / 0.305 / 0.00785;

// weightDiff — updated after every movement
const weightDiff_kg     = originalWeight_kg - currentWeight_kg;
const weightDiffPercent = (weightDiff_kg / originalWeight_kg) * 100;
```

---

2. Database: MongoDB (replaces PostgreSQL)

Why MongoDB for this module

Concern	PostgreSQL (v1)	MongoDB (v2)	Decision
Coil history	Separate `movement_log` table, JOIN required	Embedded array on coil document	MongoDB wins — history always loaded with coil
Computed fields	Generated columns	Stored fields, recomputed on write	Equivalent
Weight aggregation by spec	SQL GROUP BY	MongoDB `$group` aggregation pipeline	Equivalent
Schema flexibility	Rigid — all coils same columns	Document model — future attributes per grade	MongoDB wins
ACID on a single coil operation	Full transaction	Single-document operations are atomic in MongoDB	MongoDB wins
Cross-collection transactions	Native	Supported via sessions (needed for movement_log)	Acceptable

MongoDB collections

Collection	Description
`coils`	Primary document. Full coil record with embedded history. One doc per coil.
`movement_log`	Separate collection for cross-cutting queries (e.g. "all issues from Warehouse A this week"). Mirrors history entries. Append-only.
`warehouses`	Location master.
`suppliers`	Supplier master.
`users`	Auth records.

> `weight_ledger` is removed. The embedded `history` array on each coil document serves this role. `movement_log` captures the same data as a separate collection for reporting queries that span multiple coils.

MongoDB Document Schema

`coils` collection

```javascript
```

```
// Mongoose schema — coils collection
const coilHistorySchema = new Schema({
  clientName:          { type: String, required: true },
  kg:                  { type: Number, required: true },
  orderNumber:         { type: String, required: true },
  newCurrentWeight:    { type: Number, required: true },
  editorName:          { type: String, required: true },
  timestamp:           { type: Date,   default: Date.now },
  movementType:        { type: String, required: true, enum: [
    'GOODS_IN','JOB_ISSUE','OFFCUT_RETURN','SALE_DISPATCH',
    'TRANSFER_OUT','TRANSFER_IN','REJECTION',
    'ADJUSTMENT_ADD','ADJUSTMENT_REMOVE','DEPRECIATION'
  ]},
  machine:             { type: String },
  operator:            { type: String },
  depreciationReason:  { type: String },
}, { _id: false });  // no separate _id per history entry

const coilSchema = new Schema({
  // Identity
  seriesNumber:         { type: String, required: true, unique: true },

  // Physical spec
  grade:                { type: String, required: true },
  thickness_mm:         { type: Number, required: true },
  width_mm:             { type: Number, required: true },
  length_m:             { type: Number, required: true },
  coating_gsm:          { type: Number, required: true },
  colour:               { type: String, default: null },

  // Weight
  originalWeight_kg:    { type: Number, required: true },
  currentWeight_kg:     { type: Number, required: true },

  // Computed (stored — updated on write, not per query)
  averageWeightPerFoot: { type: Number, required: true },
  realThickness_mm:     { type: Number, required: true },
  weightDiff_kg:        { type: Number, required: true },
  weightDiffPercent:    { type: Number, required: true },

  // Supplier & location (denormalised)
  supplierName:         { type: String, required: true },
  warehouseId:          { type: Schema.Types.ObjectId, ref: 'Warehouse', required: t
```

```
  warehouseName:        { type: String, required: true },

  // Status & order
  status:               { type: String, required: true, default: 'NEW_PACK',
                          enum: ['NEW_PACK','OPEN','SOLD','DEPRECATED'] },
  orderRef:             { type: String, default: null },

  // Goods-in receipt
  dcNumber:             { type: String, required: true },
  vehicleNumber:        { type: String, required: true },
  weighMethod:          { type: String, required: true, enum: ['GATE_WEIGH','INVOICE

  // Embedded history — append-only
  history:              { type: [coilHistorySchema], default: [] },

  // Audit
  createdBy:            { type: Schema.Types.ObjectId, ref: 'User', required: true }
}, {
  timestamps: { createdAt: 'receivedAt', updatedAt: 'updatedAt' },
  collection: 'coils',
});

// Indexes
coilSchema.index({ seriesNumber: 1 }, { unique: true });
coilSchema.index({ grade: 1, thickness_mm: 1, width_mm: 1 });
coilSchema.index({ warehouseId: 1, status: 1 });
coilSchema.index({ status: 1 });
```

`movement_log` collection

Mirrors history entries across all coils. Used for reports that span multiple coils (e.g. "all JOB_ISSUE movements from Warehouse A this month").

javascript

```
const movementLogSchema = new Schema({
  coilId:           { type: Schema.Types.ObjectId, ref: 'Coil', required: true },
  seriesNumber:     { type: String, required: true },   // denormalised for fast loo
  warehouseId:      { type: Schema.Types.ObjectId, ref: 'Warehouse', required: true
  movementType:     { type: String, required: true },
  weightBefore_kg:  { type: Number, required: true },
  kg:               { type: Number, required: true },
  weightAfter_kg:   { type: Number, required: true },
  clientName:       { type: String },
  orderNumber:      { type: String },
  machine:          { type: String },
  operator:         { type: String },
  depreciationReason: { type: String },
  performedBy:      { type: Schema.Types.ObjectId, ref: 'User', required: true },
  editorName:       { type: String, required: true },
  performedAt:      { type: Date, default: Date.now },
}, {
  collection: 'movement_log',
  // Capped or TTL index can be added later if log grows very large
});

movementLogSchema.index({ coilId: 1, performedAt: -1 });
movementLogSchema.index({ warehouseId: 1, movementType: 1, performedAt: -1 });
movementLogSchema.index({ performedAt: -1 });
```

Key Aggregation Queries (MongoDB)

Coil-wise balance report:

```
javascript
```

```
db.coils.find(
  { status: { $nin: ['SOLD', 'DEPRECATED'] } },
  {
    seriesNumber: 1, grade: 1, thickness_mm: 1, width_mm: 1,
    length_m: 1, coating_gsm: 1, colour: 1,
    originalWeight_kg: 1, currentWeight_kg: 1,
    weightDiff_kg: 1, weightDiffPercent: 1,
    averageWeightPerFoot: 1, realThickness_mm: 1,
    supplierName: 1, warehouseName: 1, status: 1, orderRef: 1,
  }
).sort({ grade: 1, thickness_mm: 1, width_mm: 1 });
```

Pooled balance by spec per warehouse:

javascript

```
db.coils.aggregate([
  { $match: { status: { $nin: ['SOLD', 'DEPRECATED'] } } },
  {
    $group: {
      _id: { grade: '$grade', thickness_mm: '$thickness_mm', width_mm: '$width_mm',
      coilCount:         { $sum: 1 },
      totalWeight_kg:    { $sum: '$currentWeight_kg' },
      totalDiff_kg:      { $sum: '$weightDiff_kg' },
    }
  },
  { $sort: { '_id.grade': 1, '_id.thickness_mm': 1 } }
]);
```

Movement log for a coil (from embedded history):

javascript

```
// Preferred — single document read, no join
db.coils.findOne(
  { _id: ObjectId(coilId) },
  { history: 1, seriesNumber: 1 }
);

// For cross-coil reporting, use the movement_log collection:
db.movement_log.find(
  { warehouseId: ObjectId(wid), movementType: 'JOB_ISSUE',
    performedAt: { $gte: startDate, $lte: endDate } }
).sort({ performedAt: -1 });
```

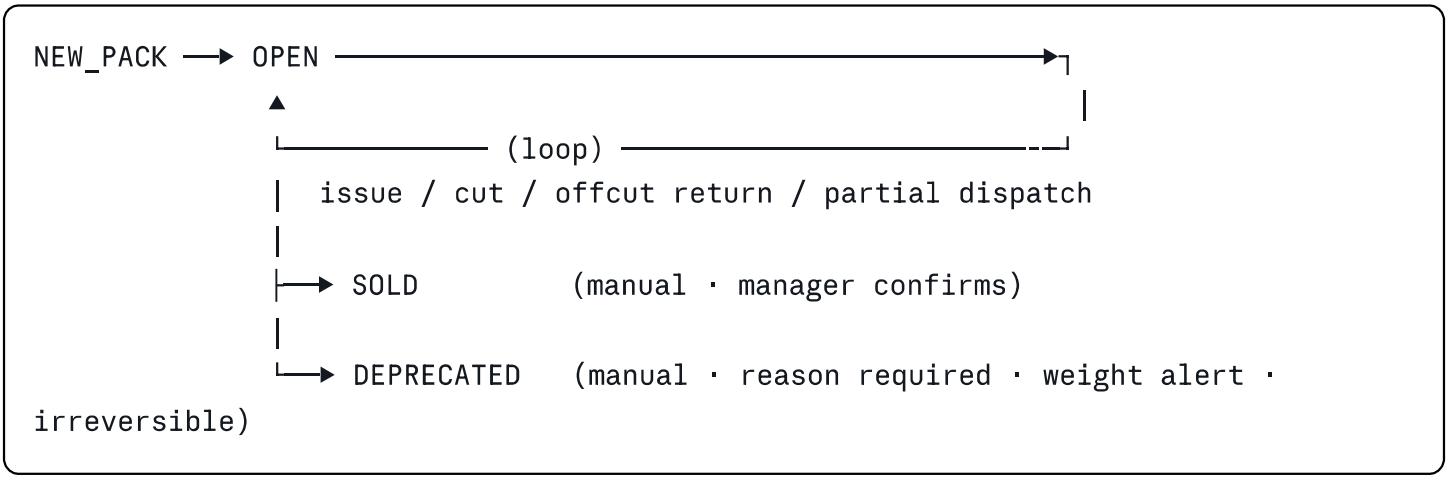
---

3. Revised Status State Machine (FSM v2)

What changed

v1	v2	Reason
5 states: NEW_PACK, OPEN, SCHEDULED, SOLD, DEPRECATED	4 states: NEW_PACK, OPEN, SOLD, DEPRECATED	SCHEDULED was a status — now `orderRef` field on OPEN coil
OPEN → DEPRECATED auto on weight threshold	DEPRECATED is manual only, always	Prevents accidental write-off of physically usable material
OPEN → SOLD auto on zero weight	SOLD is manual only, always	User must confirm final dispatch

FSM definition



typescript

```
// Enforced in CoilService.transitionStatus()
const VALID_TRANSITIONS: Record<CoilStatus, CoilStatus[]> = {
  NEW_PACK:   ['OPEN'],
  OPEN:       ['OPEN', 'SOLD', 'DEPRECATED'],
  SOLD:       [],           // terminal
  DEPRECATED: [],           // terminal
};
```

OPEN self-loop

Every operation that changes the coil weight but does not fully close the coil keeps status as `OPEN`. This includes:

- `JOB_ISSUE` — weight reduced, coil still available
- `OFFCUT_RETURN` — weight partially restored, coil still active
- `SALE_DISPATCH` — partial dispatch, coil still has remaining weight
- `ADJUSTMENT_ADD / ADJUSTMENT_REMOVE` — manual correction
- `TRANSFER_OUT / TRANSFER_IN` — location changes, coil identity unchanged

---

4. DEPRECATED — Confirmation Flow

Why DEPRECATED exists

A coil is physically finished when it runs out on the machine. But due to:

- Measurement rounding
- Scale calibration drift
- Offcut tracking gaps

...the system may still show a small positive `currentWeight_kg` (typically 5–20 kg) when the physical coil is gone. The user needs to reconcile this by **manually deprecating the coil** and writing off the digital remnant.

Confirmation flow (UI → API)

```
1. Manager opens coil detail page for coil with small remaining weight

2. Manager clicks "Deprecate coil" button
   → Button only shown to MANAGER role on OPEN coils
```

3. System opens confirmation modal:

```
┌──────────────────────────────────────────────────────┐
│  Deprecate coil AC0047?                              │
│                                                      │
│  ▲ This coil still shows 12.5 kg remaining.          │
│  Deprecating will write off this weight permanently. │
│  This action cannot be undone.                       │
│                                                      │
│  Reason (required):  _________________________       │
│                       "Coil finished on machine, scale error"  │
│                                                      │
│  [Cancel]                          [Deprecate — irreversible]  │
│                    (button locked until reason.length >= 10)   │
└──────────────────────────────────────────────────────┘
```

4. Manager types reason (minimum 10 characters)
   → "Deprecate" button unlocks

5. Manager clicks "Deprecate — irreversible"
   → Frontend sends: PATCH /coils/:id/status
     { status: 'DEPRECATED', reason: '...', acknowledgedWeight: 12.5 }

6. API (CoilService):
   a. Validates transition: OPEN → DEPRECATED ✓
   b. Validates reason.length >= 10 ✓
   c. Validates acknowledgedWeight matches currentWeight_kg ✓ (prevents stale-UI issue)
   d. Appends history entry:
      { movementType: 'DEPRECIATION', kg: -12.5, newCurrentWeight: 0,
        depreciationReason: reason, editorName, timestamp }
   e. Sets currentWeight_kg = 0, weightDiff_kg = originalWeight_kg,
      weightDiffPercent = 100, status = 'DEPRECATED'
   f. Appends to movement_log collection
   g. Returns updated coil document

7. UI: Coil card updates to show DEPRECATED badge.
   History tab shows DEPRECIATION entry with reason and manager name.

SOLD — confirmation flow

1. Manager clicks "Mark as sold"
   → Button shown to MANAGER role on OPEN coils

```
2a. If currentWeight_kg > 0:
    Modal warns: "AC0047 still shows 8.3 kg remaining. Confirm it is fully
dispatched?"
    Manager confirms → PATCH /coils/:id/status { status: 'SOLD' }

2b. If currentWeight_kg === 0:
    Direct confirm — no extra warning needed.

3. CoilService:
   a. Validates OPEN → SOLD ✓
   b. Appends SALE_DISPATCH history entry if remaining weight > 0
   c. Sets status = 'SOLD', updatedAt = now()
   d. No further transitions allowed
```

---

5. Updated API Endpoints (delta only)

Changed endpoints

Endpoint	Change
`PATCH /coils/:id/status`	Now accepts `{ status, reason?, acknowledgedWeight? }` — validates FSM; DEPRECATED requires reason
`GET /coils/:id`	Returns full document including `history[]`, computed fields, `orderRef`
`GET /coils/:id/history`	Returns `history[]` from coil document (no DB join)

Removed endpoints

Endpoint	Reason
`GET /coils/:id/ledger`	Replaced by `history[]` on coil document
`POST /movements/offcut-return`	Still exists but now also updates `averageWeightPerFoot`, `realThickness_mm` if `length_m` changes

---

6. CoilService — Key Method Signatures (updated)

typescript

```typescript
class CoilService {

  /** Register a new coil. Computes derived fields. Writes GOODS_IN history entry. *
  async register(dto: RegisterCoilDto, userId: string): Promise<CoilRecord>

  /** Update coil weight. Appends to history[]. Updates computed fields. Writes to m
  async recordWeightChange(params: {
    coilId:        string;
    movementType:  MovementType;
    kg:            number;          // negative = consumption, positive = return
    clientName:    string;
    orderNumber:   string;
    machine?:      string;
    operator?:     string;
    performedBy:   string;
    editorName:    string;
  }): Promise<CoilRecord>

  /**
   * Transition coil status.
   * DEPRECATED requires reason + acknowledgedWeight.
   * SOLD shows warning if currentWeight_kg > 0.
   * All other invalid transitions throw 422.
   */
  async transitionStatus(params: {
    coilId:             string;
    to:                 CoilStatus;
    reason?:            string;
    acknowledgedWeight?: number;
    performedBy:        string;
    editorName:         string;
  }): Promise<CoilRecord>

  /** Assign or update order reference on an OPEN coil (replaces SCHEDULED status).
  async setOrderRef(coilId: string, orderRef: string | null, performedBy: string): P

  /** Recompute and persist all derived fields. Called after any weight or dimension
  private async recomputeDerivedFields(coilId: string): Promise<void>
}
```

7. OTD Updates (Open Technical Decisions — revised)

#	Decision	v1	v2
OTD-01	Database hosting	Supabase / GCP Cloud SQL	MongoDB Atlas (free tier → M10 as needed)
OTD-02	ORM / driver	Drizzle ORM	Mongoose 8 + native MongoDB driver
OTD-04	Depreciation threshold	Configurable % auto-trigger	Removed — DEPRECATED is always manual
OTD-08	Pooled vs individual	Weight threshold	`seriesNumber` presence = individual; no series = pooled batch entry

Patch v2 — supersedes sections 3.1, 5, 6, 7, 8 of IMS_RawMaterial_SolutionDesign_v1.md
Full updated spec = v1 doc + this patch applied in order