

TECHNICAL WHITE PAPER

Bhagirath ERP Architecture, Design and Operation

How the plant's manufacturing ERP works: its architecture, the design decisions behind it, the mechanisms that enforce quality and material flow, its data model, security, and how to run and extend it.

VERSION

**1.0 · Handover
edition**

DATE

October 2026

AUDIENCE

**Technical owner,
developers**

CLASSIFICATION

**Internal ·
Confidential**

Document control

Title	Bhagirath ERP: Architecture, Design and Operation (Technical White Paper)
Version	1.0, handover edition
Date	October 2026
System version	Database migration head <code>g3b4c5d6e7f8</code>
Audience	Technical project manager taking ownership; developers and analysts maintaining the system
Classification	Internal and confidential. Do not distribute outside the organisation.
Companion documents	Setup guide for Windows; setup guide for macOS; database export guide; summary of changes (before and after); <code>docs/DATABASE_REFERENCE.md</code> inside the project

Scope

This paper describes the complete system as delivered: the staff application, the shop-floor station screens, the dealer, vendor and maintenance portals, the REST API and the PostgreSQL database. It covers architecture, design rationale, the core mechanisms, data, security, code organisation, business processes, change impact, extension, operations and known limitations. Installation steps are in the separate setup guides.

Contents

Executive summary

1 About this document

1.1 Purpose and audience

1.2 How to read it

1.3 Conventions

2 Business context

2.1 The plant

2.2 Problems the system solves

3 System overview

3.1 Users and portals

3.2 Capabilities

3.3 Technology

3.4 Deployment topology

4 Architecture

4.1 Architectural style

4.2 Back-end structure

4.3 Front-end structure

4.4 How a request flows

4.5 Cross-cutting concerns

5 Design decisions

D1 A modular monolith rather than microservices

D2 Asynchronous Python with FastAPI and SQLAlchemy 2

D3 One relational database with strong keys

D4 Permissions as data, with no hidden administrator bypass

D5 One table per dropdown list, indexed by a registry

D6 Station access by assignment, with permission fallback

D7 The QA gate as an explicit state machine table

D8 Routing by the chassis's own bill of material

D9 A frozen BOM snapshot per chassis

- D10** Server-issued sequential document numbers
- D11** Server state in React Query, one hook per API call
- D12** Phone-first station screens on the same host
- D13** A self-describing database
- D14** Idempotent, reversible migrations

6 Core mechanisms

- 6.1** The chassis lifecycle
- 6.2** The QA-gated station hand-over
- 6.3** Next-station routing
- 6.4** Procurement check: exploding the bill of material
- 6.5** Stock consumption on QA pass
- 6.6** QA inspection lifecycle and concurrency
- 6.7** Permission resolution
- 6.8** Station access
- 6.9** Cycle time and overdue detection
- 6.10** Sequential document numbering
- 6.11** The configuration layer: changing behaviour without code

7 Data architecture

- 7.1** Overview
- 7.2** Naming conventions
- 7.3** Business areas
- 7.4** Where to find a value
- 7.5** Schema evolution

8 Security model

- 8.1** Authentication
- 8.2** Authorisation
- 8.3** Public surface
- 8.4** Secrets and network

9 Code reference

- 9.1** Repository layout
- 9.2** Back-end modules
- 9.3** Key functions

9.4 Screens

10 Business processes end to end

10.1 Order to chassis

10.2 Chassis inward

10.3 Induction and procurement

10.4 Production with QA gates

10.5 Final inspection and dispatch

10.6 Supporting processes

11 Change impact

12 Extending the system

12.1 Add a dropdown list

12.2 Add an API endpoint

12.3 Add a screen

12.4 Add a table or column

12.5 Change a status flow

13 Operations

13.1 Running

13.2 Back-up and restore

13.3 Upgrading

13.4 Monitoring

13.5 Moving to production hosting

14 Quality assurance

15 Limitations and recommendations

Appendix A Glossary

Appendix B Configuration reference

Appendix C Status catalogue

Executive summary

Bhagirath ERP is the operating system of a bus body-building plant. It follows every customer chassis from the moment it arrives at the yard, through each production station and its quality gate, to the gate pass that lets the finished bus leave. Around that core it runs sales and dealer orders, bills of material, purchasing and stores, and three external portals for dealers, vendors and the maintenance team.

The system is a single web application backed by one PostgreSQL database. A React front end serves office screens and phone-first shop-floor screens; a FastAPI back end exposes a versioned REST API organised into 22 business modules; the database holds about 157 tables, each described in plain language inside the database itself.

Four ideas shape the design:

1. **The chassis is the unit of work.** Every process (inward, procurement, production, quality, dispatch) reads and writes against one record per chassis, and every movement is logged.
2. **Quality is a gate, not a report.** A chassis cannot be checked in at the next station until QA at the current station has passed every mandatory checksheet and released it. The gate is an explicit, auditable state machine.
3. **Configuration lives in data, not code.** Stations and their order, QA checksheets, dropdown lists, roles, permissions and station staff are all rows that administrators edit on screen. Most day-to-day changes need no developer and no redeployment.
4. **The system explains itself.** Tables carry descriptions, a generated reference lists every column, and permissions, statuses and flows are documented in this paper.

This paper explains how the system works, why it is built the way it is, and how its key mechanisms solve the plant's problems, so that a technical owner can operate, change and extend it with confidence.

1. About this document

1.1 Purpose and audience

This is the authoritative technical description of Bhagirath ERP at handover (October 2026). It is written for the technical project manager who owns the system, and for developers and analysts who will maintain it. It assumes familiarity with web applications and relational databases, but no prior knowledge of this system or of bus manufacturing.

1.2 How to read it

If you want to	Read
Understand the system in ten minutes	Executive summary, Chapter 3
Understand how it is built	Chapters 4 and 5
Understand the key algorithms and rules	Chapter 6
Find data in the database	Chapter 7
Understand security	Chapter 8
Locate a file or screen	Chapter 9
Follow a business process end to end	Chapter 10
Change something safely	Chapters 11 and 12
Run, back up and upgrade it	Chapter 13

1.3 Conventions

- `monospace` marks file names, table and column names, status values and commands.
- Status values are written in capitals, exactly as stored, for example `READY_FOR_DISPATCH`.
- Menu paths are written **Group > Item**, for example **Setup > Station Staff**.
- File paths are relative to the project folder (`backend/...` , `frontend/...`).
- Companion files inside the project: `docs/DATABASE_REFERENCE.md` (every table and column), `docs/SETUP_GUIDE_WINDOWS.md` , `docs/SETUP_GUIDE_MAC.md` , `backend/scripts/README.md` .

2. Business context

2.1 The plant

A customer (usually a dealer acting for a school, company or transport operator) orders buses of a given model, body type, customer segment and seating layout, with optional features such as air conditioning, door types or fire-protection systems. The customer's bare chassis arrives at the plant yard. The plant then builds the body on it through a fixed sequence of stations (chassis preparation, floor and side structure welding, roof, structure QA, primer and top-coat painting, electrical wiring, flooring and panelling, seat fitment, glass and doors) and finishes with a final vehicle inspection (FVI), an air-conditioning (thermal) check and a gate pass.

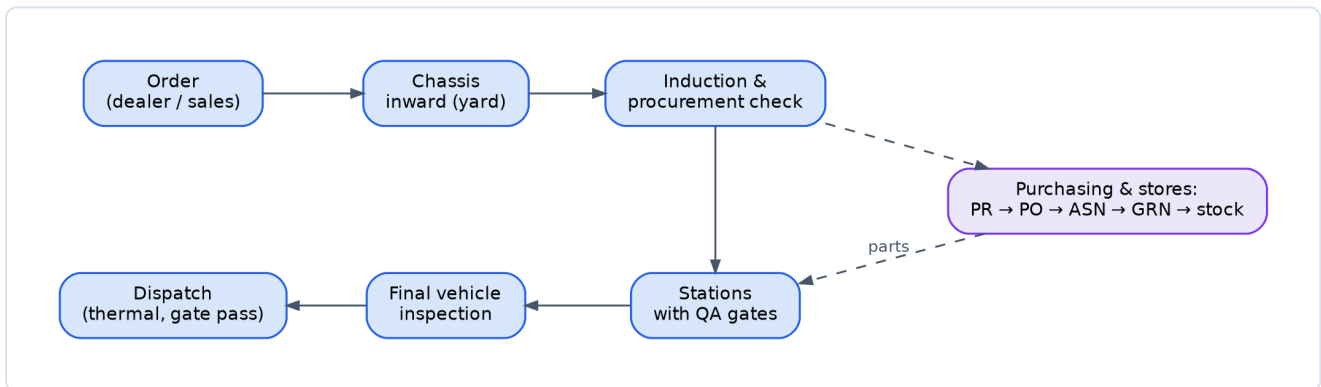


Figure 1. The value stream: from order to dispatch, with purchasing feeding parts to the stations.

2.2 Problems the system solves

Problem in the plant	How the system addresses it
Nobody could say where a given chassis was, or how long it had been there	Every check-in and completion is logged per station with timestamps; the Production Floor shows each station's chassis, elapsed time, age in days and overdue flags
Defects found late, after the chassis had moved on	QA-gated hand-over: the next station cannot accept a chassis until QA at the current station passes and releases it
Parts missing at the station when the chassis arrived	Procurement check per chassis: the order's full bill of material is exploded, stock reserved, shortages raised as purchase requests before production starts
Commercial holds (payment, documents) discovered at dispatch	DA-lite status recorded at inward; holds are visible to sales and on dashboards throughout
Paper checksheets, lost photos, unreadable remarks	Digital checksheets on phones with measurements, photos and remarks per checkpoint, section and checksheet
Dropdown values and station lists hard-coded by developers	Administrators maintain them on screen; values live in database tables
Dealers and vendors phoning for status	Dealer and vendor portals with their own logins; public QR tracking page per chassis

3. System overview

3.1 Users and portals

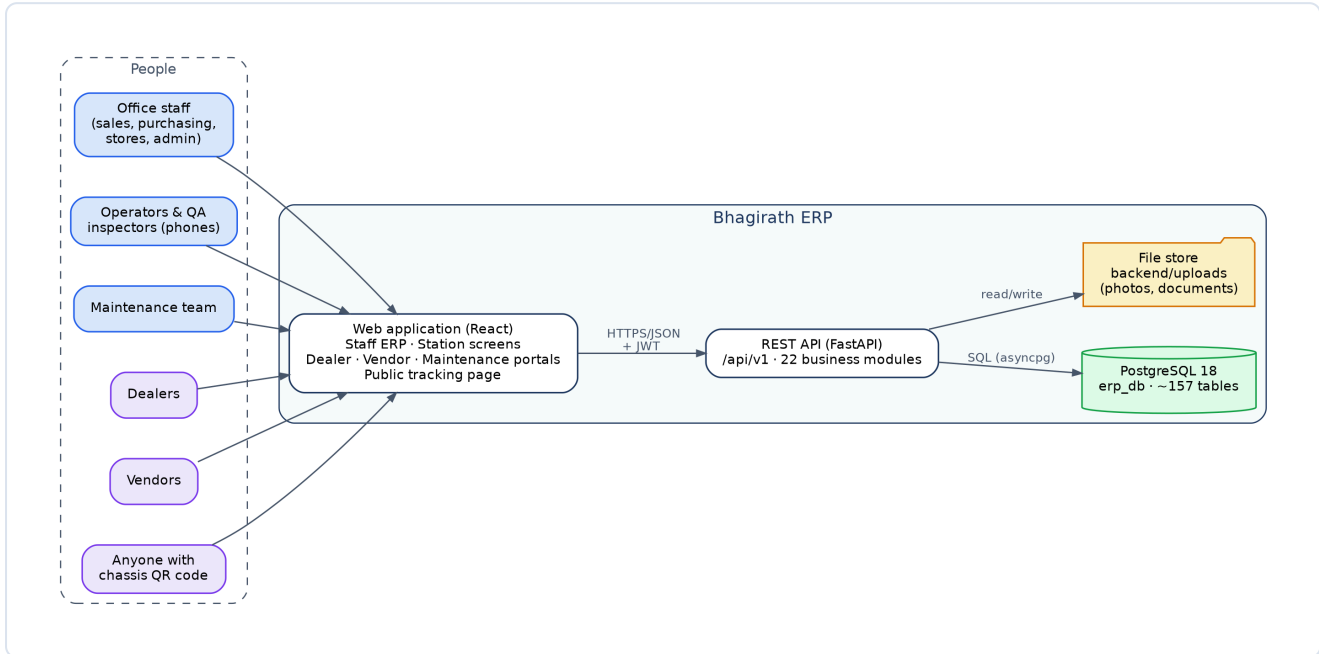


Figure 2. System context: who uses the system and its main parts.

Portal	Address	Users	Sign-in
Staff ERP	/	Office staff, supervisors, administrators	Staff account (<code>login_users</code>)
Station screens	/my-station , /qa/inspect	Operators and QA inspectors on phones	Staff account
Dealer portal	/dealer-portal	Dealers: orders, clients, chassis status	Dealer account (<code>customer_master</code>)
Vendor portal	/vendor-portal	Suppliers: purchase orders, shipping notices, rejections	Vendor account (<code>vendor_master</code>)
Maintenance portal	/maintenance-portal	Maintenance team	Staff account with MAINTENANCE permission
Public tracking	/track/<chassis no>	Anyone with the chassis QR code	None

3.2 Capabilities

Area	Capabilities
Chassis and yard	Chassis inward, arrival documents, DA-lite check with photo, yard QA checklist, movements between yard checkpoints, store hand-over sheet with approvals, hand-over to production, yard reports, QR tracking

Area	Capabilities
Production	Induction scheduling, station check-in and completion from phones, live floor view, cycle-time warnings and overdue detection, delay reasons, holds, production report
Quality	Checksheet templates (sections, checkpoints, specifications), mapping to stations, phone inspections with photos and remarks, sign-off, QA gate and release, chassis QA history
Dispatch	Dispatch queue, thermal (AC) checksheet, gate pass
Sales	Dealers and clients, dealer order review and confirmation, chassis allocation, induction schedule, payments, goodwill approvals, spare parts and service orders, sales dashboard
Engineering	Vehicle models, body types, segments, special features and options, seating layouts, seat rules, base and feature bills of material
Purchasing and stock	Procurement engine, monthly plans, purchase requests and orders, goods receipt, debit notes, stock balances, movements, consumption, store requests
Administration	Users, roles, permission matrix, per-user overrides, station staff, dropdown lists, system logs

3.3 Technology

Layer	Technology
Database	PostgreSQL 18
Back end	Python 3.12, FastAPI, SQLAlchemy 2 (asynchronous, asyncpg driver), Pydantic 2, Alembic migrations
Front end	React 18, Vite, React Router, TanStack Query (React Query), Tailwind CSS, axios
Shop-floor scanning	html5-qrcode (phone camera), with typed or picked chassis as a fallback
Authentication	JSON Web Tokens (HS256), bcrypt password hashes

3.4 Deployment topology

In the current (local) deployment, three processes run on one computer: PostgreSQL on port 5432, the API on port 8000 and the front-end server on port 5173. Uploaded photos and documents are stored as files under `backend/uploads` and served by the API; the database stores only their paths. Phones on the same network reach the application at `http://<computer address>:5173`; the front end calls the API on the same host, port 8000.

4. Architecture

4.1 Architectural style

Bhagirath ERP is a **modular monolith** with a **three-tier** structure: a single-page web application, a single API process, and a single relational database. Business areas are separated into modules inside the API, each with its own routes, schemas and services, but they share one process, one database and one transaction model.

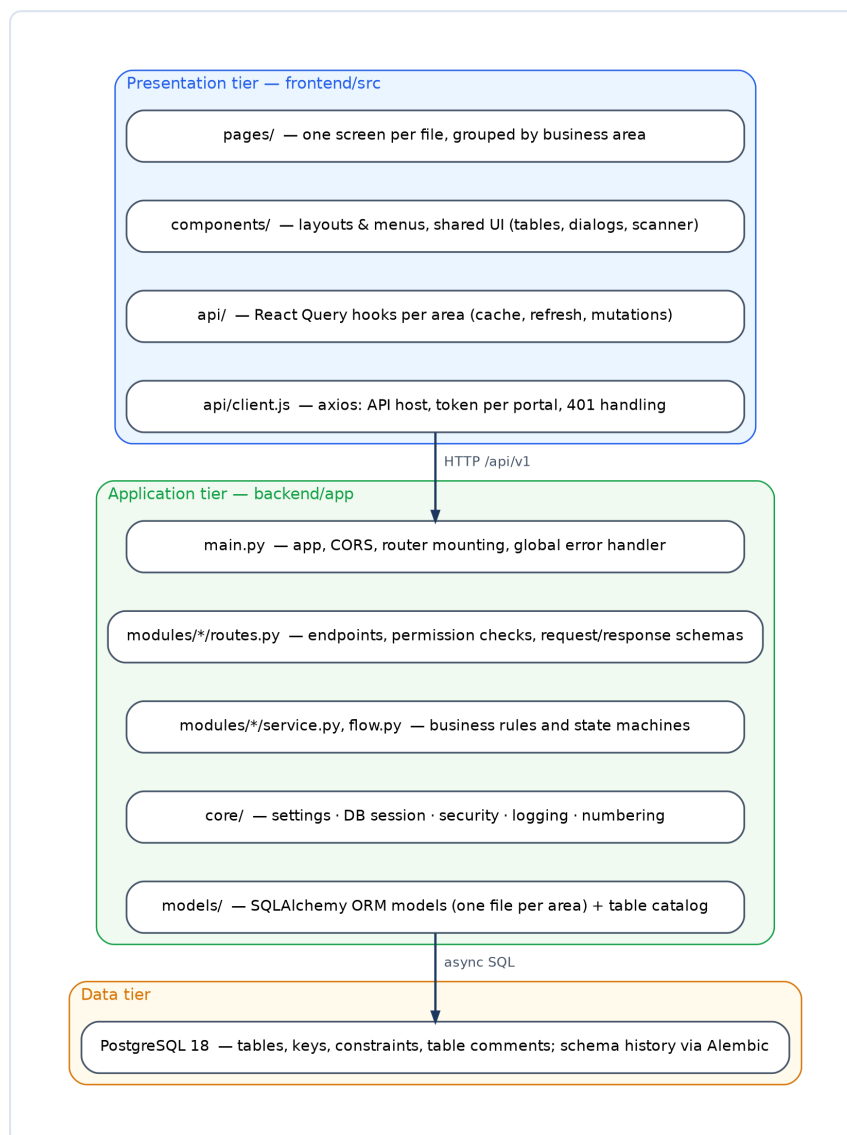


Figure 3. Layered architecture.

Tier	Responsibility	Location
Presentation	Screens, navigation, client-side caching, phone camera scanning	frontend/src
Application	Authentication, authorisation, business rules, state machines, validation	backend/app
Data	Persistent state, referential integrity, schema history	PostgreSQL, backend/migrations

4.2 Back-end structure

Each business module follows the same layering:

Layer	File	Holds
Interface	<code>routes.py</code>	Endpoint declarations, permission dependencies, request and response models
Domain	<code>service.py</code> (and <code>flow.py</code> for station work)	Business rules, state transitions, calculations
Contract	<code>schemas.py</code>	Pydantic models validating input and shaping output
Persistence	<code>app/models/*.py</code>	SQLAlchemy models mapped to tables

Cross-cutting services live in `app/core`: configuration (`config.py`), database engine and per-request session (`database.py`), authentication and authorisation (`security.py`), error logging (`logging.py`) and sequential document numbering (`numbering.py`).

4.3 Front-end structure

Layer	Location	Role
Routing	<code>src/App.jsx</code>	Maps every address to a screen, per portal; old addresses redirect
Shell	<code>src/components/layout/*</code>	Menus, top bar, phone tab bar; menu entries declare the permission module that shows them
Screens	<code>src/pages/<area>/*</code>	One file per screen
Data access	<code>src/api/<area>.js</code>	React Query hooks wrapping each API call; mutations refresh related data
HTTP	<code>src/api/client.js</code>	API host resolution, token per portal, sign-out on expired session
Shared UI	<code>src/components/ui/*</code>	Tables, dialogs, drawers, scanners, dropdowns bound to database lists

4.4 How a request flows

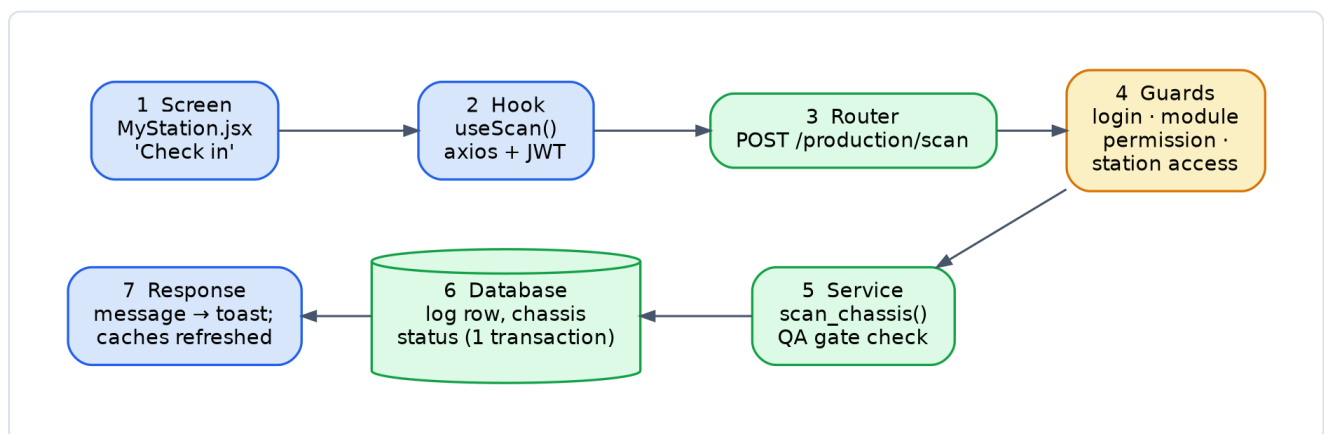


Figure 4. Life of a request: an operator checks a chassis in at a station.

1. The operator presses **Check in** on **My Station**. The screen calls the `useScan` hook.
2. The hook sends `POST /api/v1/production/scan` with the stored token.
3. FastAPI routes the request to the production module.
4. Dependencies run before the handler: the token is decoded and the user loaded; the PRODUCTION permission is checked; the station-access rule confirms the operator may work at that station.
5. `scan_chassis` applies the business rules, including the QA gate (Chapter 6.2), and writes a station log row and the chassis status.
6. The request's database session commits once, at the end. Any error rolls everything back.
7. The response carries a human-readable message; the front end shows it and React Query refreshes the station board.

4.5 Cross-cutting concerns

Concern	Mechanism
Transactions	One database session per request (<code>get_db</code>): commit on success, rollback on any exception, so a request is all-or-nothing
Validation	Pydantic schemas reject malformed input with HTTP 422 before business code runs
Errors	Business errors raise <code>HTTPException</code> with a readable <code>detail</code> shown to the user; unexpected errors are logged with full traceback to <code>backend/logs/error.log</code> and answered with a generic 500
Audit	Append-only log tables (<code>chassis_status_log</code> , <code>chassis_station_log</code> , portal action logs, permission audit)
Concurrency	Database advisory locks where two users can race (inspection start, document numbering)
Configuration	Environment file <code>backend/.env</code> ; no secrets in code
API documentation	Generated live at <code>/docs</code> from the code

5. Design decisions

Each decision below records the context, the choice made, why, and its consequences. Together they explain why the system looks the way it does and what to preserve when changing it.

D1. A modular monolith rather than microservices

Context. One plant, one IT owner, tightly linked processes: a QA sign-off changes stock and production state in the same moment.

Decision. One API process and one database, split internally into business modules.

Rationale. Cross-area operations stay in a single database transaction, so they are consistent by construction. Operating one process is within reach of a small team.

Consequences. Simple to run and reason about. Modules share tables, so a schema change can affect several modules; Chapter 11 lists the main dependencies.

D2. Asynchronous Python with FastAPI and SQLAlchemy 2

Context. Many short requests from phones and office screens, largely waiting on the database.

Decision. FastAPI with async SQLAlchemy and the asyncpg driver.

Rationale. High concurrency per process, automatic request validation and live API documentation.

Consequences. Database access must use `await`; scripts that reuse the models run inside `asyncio.run`. The test suite shares one event loop across tests (`pytest.ini`).

D3. One relational database with strong keys

Context. Chassis, orders, parts and inspections are deeply interrelated and must stay consistent.

Decision. PostgreSQL with UUID primary keys, foreign keys, unique and check constraints; schema changes only through Alembic migrations.

Rationale. Integrity is enforced by the database, not only by the code; UUIDs can be generated anywhere without collisions.

Consequences. Deletes of referenced rows fail by design; masters are deactivated (`is_active`) instead of deleted.

D4. Permissions as data, with no hidden administrator bypass

Context. Roles and responsibilities change; the plant should not need a developer to change who can do what.

Decision. Every endpoint declares a module and action (`require_permission('QA', 'approve')`). Whether a user may do it is resolved from `role_module_permission` , with optional per-user overrides. The administrator role has no special case in code.

Rationale. One rule for everyone is easy to audit; the screen shows exactly what the server enforces, because the front end reads the same resolved permissions.

Consequences. Granting access is an on-screen change. Removing a row can lock out a role, including administrators, so changes to the ADMIN role need care.

D5. One table per dropdown list, indexed by a registry

Context. Seventeen dropdown lists were originally typed into the program.

Decision. Each list is its own `*_master` table with the same core columns (`code` , `label` , `sort_order` , `is_active`), and `master_list_registry` names every list and its table. A list can add typed columns of its own (for example the gate pass's default quantity and print row).

Rationale. Each list is visible and editable as a normal table, can carry list-specific columns with real types, and is discoverable through the registry, so one admin screen and one API serve all lists.

Consequences. Adding a list is a small migration plus one line of configuration (Chapter 12).

D6. Station access by assignment, with permission fallback

Context. Operators and inspectors should see only their stations; supervisors need to act anywhere.

Decision. `station_user_assignment` links users to stations as OPERATOR or QA. A user with assignments in a role is restricted to them; a user with none falls back to the module permission.

Rationale. Restriction is opt-in per person, so introducing assignments never locks supervisors out.

Consequences. Several assignments give one tab per station on the phone screens.

D7. The QA gate as an explicit state machine table

Context. Quality must be enforced between stations, and every decision must be traceable.

Decision. A dedicated table, `chassis_station_handoff` , holds one row per hand-over with an explicit status (`AWAITING_QA` , `QA_PASSED` , `RELEASED` , `RECEIVED` , `SUPERSEDED`), timestamps and the people involved. All transitions go through one module (`station_work/flow.py`).

Rationale. The gate's state is queryable and auditable at any moment, and the rules live in one place.

Consequences. Chassis already on the floor before the gate existed have no hand-over history and are deliberately not blocked.

D8. Routing by the chassis's own bill of material

Context. Not every chassis needs every station; a bus without, say, a particular fit-out has no parts for that station.

Decision. The next station is the next active station, by sequence, for which this chassis's frozen BOM has parts; if there is none, simply the next station in sequence.

Rationale. The route adapts to each order without a separate routing table to maintain.

Consequences. Part-to-station assignment (`rm_master.station_id`) directly affects routing.

D9. A frozen BOM snapshot per chassis

Context. Base BOMs and feature BOMs change over time; a chassis in production must keep the parts list it was planned with.

Decision. At the procurement check, the exploded BOM is written to `chassis_bom_snapshot` , with reserved and consumed quantities.

Rationale. Stock reservation, station routing and consumption all work from one stable list.

Consequences. Changing a BOM affects chassis checked afterwards; re-running the check for a chassis rebuilds its snapshot.

D10. Server-issued sequential document numbers

Context. Purchase order, goods receipt and debit note numbers were random numbers made in the browser, so duplicates were possible and numbers meant nothing.

Decision. The server issues `PREFIX-YEAR-NNNNN` numbers, serialised with a database advisory lock.

Rationale. Unique, ordered, human-readable numbers, safe under concurrent use.

Consequences. A user may still type a number; a blank field gets the next one.

D11. Server state in React Query, one hook per API call

Context. Many screens show the same data (chassis, stations, lists) and must stay current.

Decision. Screens never call the API directly; they use hooks from `src/api`, which cache data and refresh related queries after every change.

Rationale. Consistent loading and error handling, fewer requests, and screens that update themselves.

Consequences. When adding an API call, add a hook and invalidate the right query keys.

D12. Phone-first station screens on the same host

Context. Operators and inspectors work on the floor with phones.

Decision. Dedicated large-touch screens (**My Station, QA Inspection**); the front end calls the API on whatever host served the page, so the same build works on a desktop and on a phone on the plant network.

Rationale. No per-device configuration; the floor uses the same system as the office.

Consequences. Browsers allow the camera only on HTTPS pages, so camera scanning needs an HTTPS deployment; typed or picked chassis numbers always work.

D13. A self-describing database

Context. A new owner must be able to find data without reading code.

Decision. Every table carries a `COMMENT` written from `app/models/table_catalog.py`, and `docs/DATABASE_REFERENCE.md` is generated from the live schema.

Rationale. Documentation that lives with the data does not drift.

Consequences. When adding a table, add one line to the catalog and re-run two scripts.

D14. Idempotent, reversible migrations

Context. The database was originally built from a dump, then evolved by migrations.

Decision. Migrations check before creating (`IF NOT EXISTS`, existence queries) and provide a downgrade. `alembic check` verifies that models and database agree.

Rationale. Upgrades are safe to re-run and safe on databases restored from older dumps.

Consequences. New environments are built from a dump and then upgraded (Chapter 13).

6. Core mechanisms

This chapter explains the algorithms and rules that solve the plant's central problems.

6.1 The chassis lifecycle

A chassis carries several independent status fields, each owned by one part of the process. Reading them together tells you exactly where it stands.

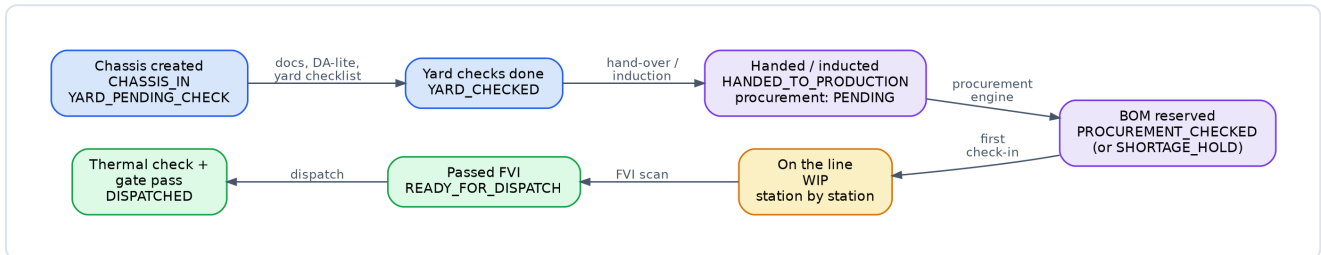


Figure 5. Chassis lifecycle across the status fields.

Field	Values	Set by
<code>tracking_status</code>	<code>CHASSIS_IN</code> → <code>WIP</code> → <code>READY_FOR_DISPATCH</code> → <code>DISPATCHED</code>	Yard (creation), first station check-in, FVI scan, dispatch
<code>current_status</code>	<code>YARD_PENDING_CHECK</code> → <code>YARD_CHECKED</code> → <code>HANDED_TO_PRODUCTION</code>	Yard checklist, hand-over or induction
<code>procurement_status</code>	<code>PENDING</code> → <code>PROCUREMENT_PENDING</code> → <code>PROCUREMENT_CHECKED</code> or <code>SHORTAGE_HOLD</code>	Induction, procurement engine
<code>da_lite_status</code>	<code>PENDING</code> , <code>OK</code> , <code>HOLD</code>	Yard DA-lite check
<code>production_hold_status</code>	<code>NONE</code> , <code>ON_HOLD</code> , <code>RELEASED</code>	Production holds

Every change of `tracking_status` is recorded in `chassis_status_log` with the user, time and location, giving a complete history.

6.2 The QA-gated station hand-over

Problem. Defects must be caught at the station that caused them, before the chassis moves on.

Mechanism. When an operator marks work complete at a station, a hand-over record is created to the next station. Its status decides whether the next station may accept the chassis.

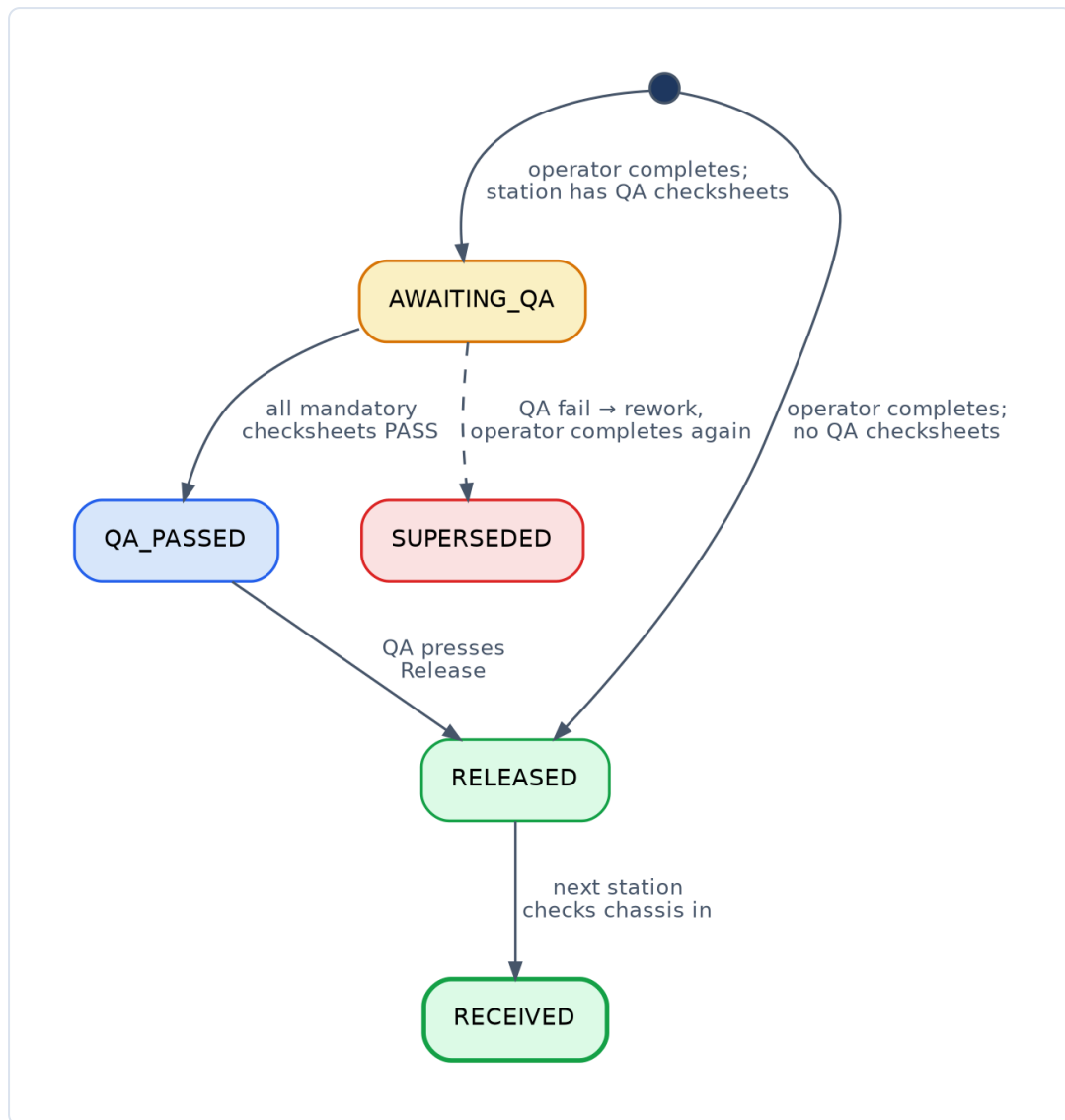


Figure 6. Hand-over states.

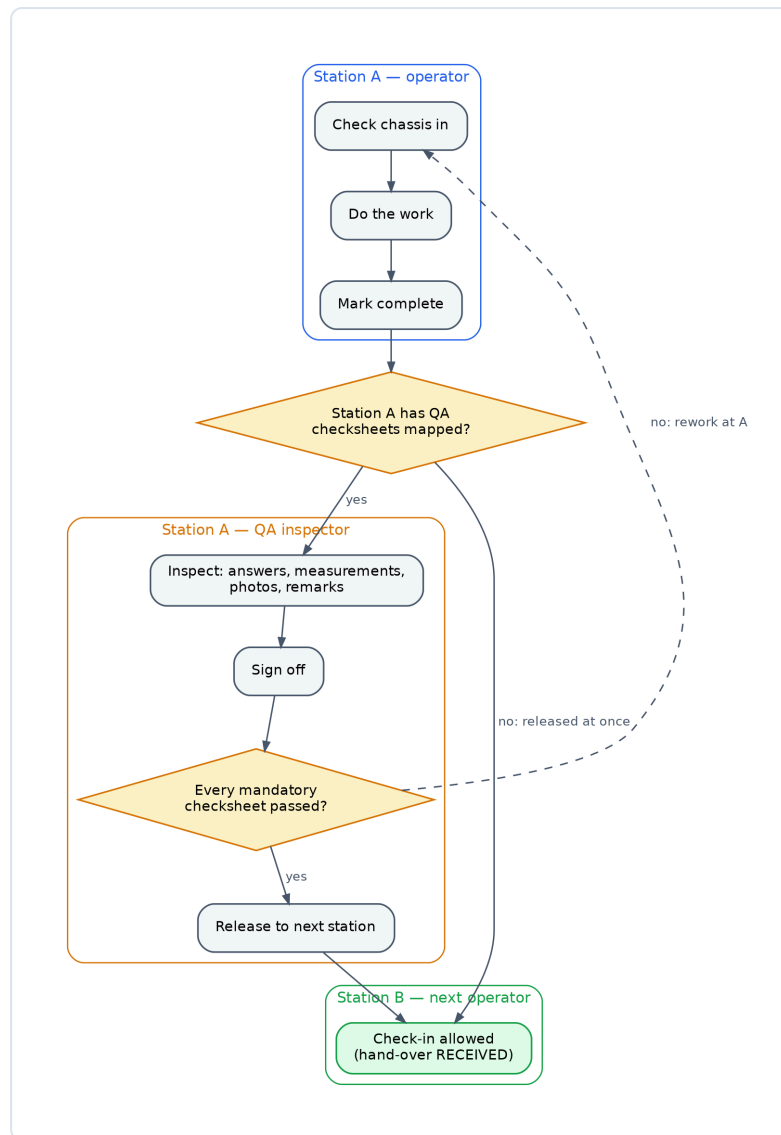


Figure 7. The QA gate as seen by operator, inspector and next station.

The rules, implemented in `backend/app/modules/station_work/flow.py` :

Event	Rule
Operator completes station A	If station A has QA checksheets mapped, create hand-over AWAITING_QA ; otherwise create it RELEASED . Any earlier open hand-over from A for this chassis becomes SUPERSEDED .
QA signs off at station A	If every mandatory checksheet for this chassis at A has passed, the hand-over becomes QA_PASSED ; if not, it stays AWAITING_QA and the operator reworks.
QA releases	QA_PASSED → RELEASED , recording who released it and when.
Operator at station B checks in	Allowed only if the chassis's latest hand-over is RELEASED to B (it then becomes RECEIVED), or if the chassis has no hand-over history at all. Otherwise refused with a message naming the reason.

Why it works. The gate is enforced on the server at check-in time, so no screen can bypass it; each transition is a timestamped row, so the history of every decision is preserved.

6.3 Next-station routing

Problem. Different orders need different stations.

Algorithm.

```
next_station(chassis, current):
    candidates = active stations with sequence_no > current.sequence_no, in order
    for station in candidates:
        if chassis_bom_snapshot has a part for this chassis whose station is `station`:
            return station
    return first of candidates, or "end of line" if none
```

The station of a part comes from `rm_master.station_id`, copied into the plan when the procurement check runs.

6.4 Procurement check: exploding the bill of material

Problem. Know before production starts which parts each chassis needs and whether they are in stock.

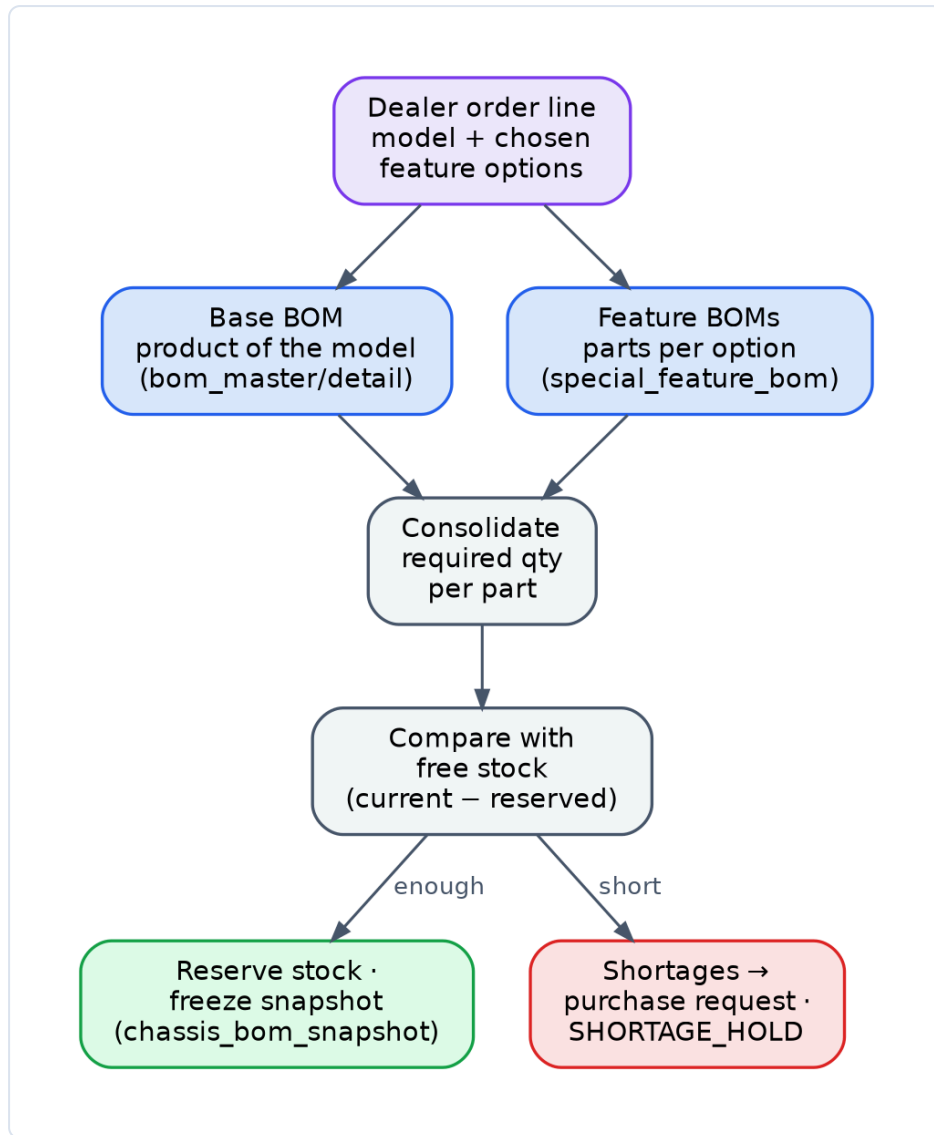


Figure 8. Procurement check for one chassis.

Algorithm (`procurement/engine_service.py`):

1. Take the order line behind the chassis: vehicle model and chosen feature options.
2. Base BOM: the product matching the model, and its `bom_detail` lines.
3. Feature BOMs: for each chosen option, its `special_feature_bom` lines.
4. Consolidate quantities per part.
5. For each part, free stock = current – reserved (`rm_inventory`). Report required, available and shortage per part (the *simulation* shown to the planner).
6. On confirmation: replace the chassis's `chassis_bom_snapshot` , reserve what stock allows, and turn shortages into purchase requests. The chassis becomes `PROCUREMENT_CHECKED` , or `SHORTAGE_HOLD` if anything is short.

A bulk variant (`bulk_engine_service.py`) runs the same logic for many chassis at once.

6.5 Stock consumption on QA pass

When QA passes a chassis at a station, the parts of its BOM snapshot assigned to that station are consumed from stock and the snapshot's consumed quantity is updated (`qa/service.py` , `_deduct_inventory_on_qa_pass`). Stock therefore moves when work is verified, not when it is planned.

6.6 QA inspection lifecycle and concurrency

1. **Start.** For a chassis at a station, the system finds the mapped checksheet templates (optionally restricted to some vehicle models or body types) and creates one inspection instance per template, or reopens existing ones. Two inspectors opening the same chassis at the same moment are serialised with a database advisory lock, so duplicate instances are not created.
2. **Answer.** Each checkpoint takes OK / NOT OK / NA, an optional measured value, a remark and photos. Answers are saved in bulk; photos are uploaded individually and do not discard unsaved answers.
3. **Remarks.** Remarks can be added per checkpoint, per section (`qa_section_remark`) and for the whole checksheet.
4. **Sign-off.** Records the outcome and reports whether all mandatory checksheets for this chassis and station have passed, which drives the hand-over (6.2).

6.7 Permission resolution

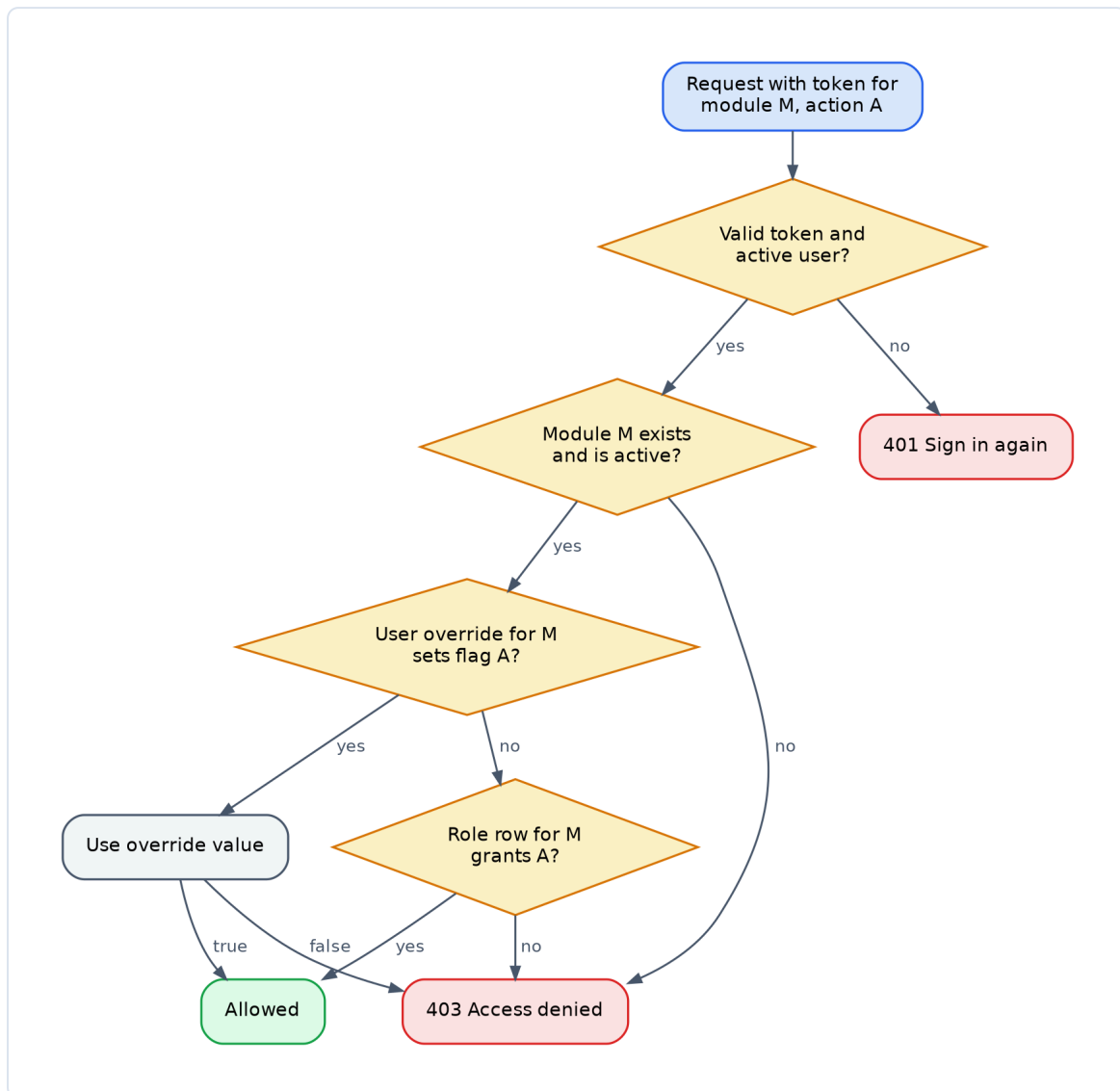


Figure 9. How the server decides whether a request is allowed.

The same resolution is exposed to the front end (`GET /auth/me/permissions`), which uses it to show or hide menu entries and buttons, so the screen always matches what the server enforces.

6.8 Station access

```
may_act(user, station, role):          # role is OPERATOR or QA
    assigned = stations assigned to user in this role
    if assigned is not empty:
        return station in assigned
    return user has view permission on PRODUCTION (operator) or QA (inspector)
```

6.9 Cycle time and overdue detection

Each station defines a standard cycle time and an alert threshold in minutes

(`station_master.standard_cycle_time_min` , `alert_threshold_min`). For every chassis currently at a station, elapsed time since check-in is compared with these values: above the threshold it shows a warning, above the cycle time it is overdue and can be given a delay reason (`delay_reason_master`). Durations are shown in days and hours once they pass a day.

6.10 Sequential document numbering

```
next_document_number(prefix):          # e.g. "P0"
    stem = prefix + "-" + current_year + "-"
    take a transaction-level advisory lock on stem    # serialises concurrent requests
    highest = largest numeric suffix already issued with this stem
    return stem + zero_padded(highest + 1, 5)        # P0-2026-00042
```

6.11 The configuration layer: changing behaviour without code

Much of the system's behaviour is driven by data that administrators maintain on screen. This is the system's low-code layer.

What you configure	Where	Effect
Stations, their order, cycle times	Setup > Stations	The production line, routing, floor view, overdue detection
Who works at which station	Setup > Station Staff	Station tabs on phones; station access
QA checksheets and checkpoints	Quality > QA Checksheets	What inspectors check, which checkpoints are mandatory
Which checksheets run at which station	Quality > QA Checksheets > Station assignments	Whether a station has a QA gate, for which vehicles
Dropdown lists (17)	Setup > Production Masters > Dropdown Lists	Choices offered across the system
Vehicle models, body types, segments, features, seating	Setup > Production Masters	What can be ordered and configured
Base and feature BOMs	Setup > Bill of Materials	Parts planned per chassis, routing
Roles and permissions, per-user exceptions	Admin > Users & Access	Menus, buttons and API access
Delay reasons, hold types	Production Floor, Dropdown Lists	Reporting categories

7. Data architecture

7.1 Overview

The database `erp_db` (schema `public`) holds about 157 tables in 13 business areas. Every table has a description stored in the database (pgAdmin: select the table, **Properties > Comment**). The complete column-level reference is [docs/DATABASE_REFERENCE.md](#).

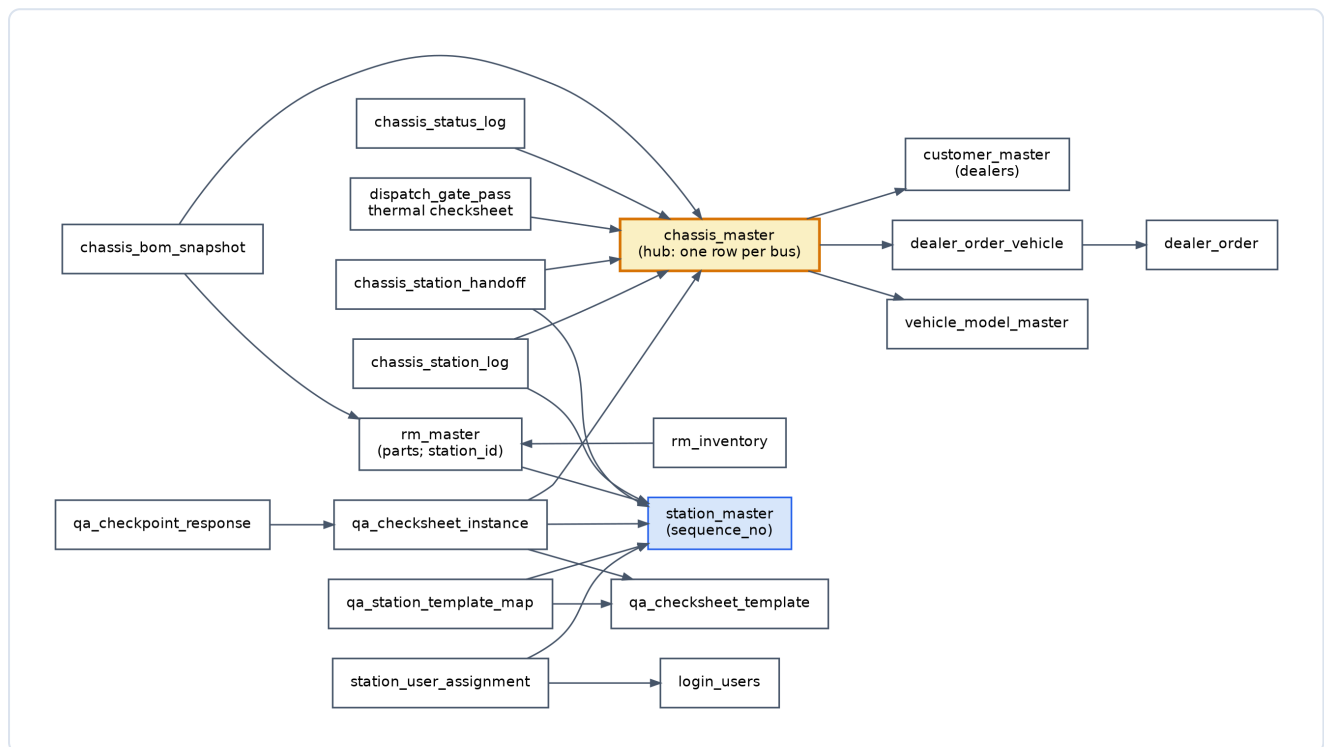


Figure 10. Core entities around the chassis.

7.2 Naming conventions

Pattern	Meaning	Example
<code>*_master</code>	Reference data maintained by administrators	<code>station_master</code> , <code>ac_make_master</code>
<code>*_log</code>	Append-only history, never edited	<code>chassis_status_log</code>
<code>*_map</code> , <code>*_mapping</code>	Link between two entities	<code>qa_station_template_map</code>
<code>*_detail</code> , <code>*_item</code> , <code>*_line</code>	Lines of a header document	<code>rm_purchase_order_detail</code>
<code><entity>_id</code>	UUID primary key, reused as the foreign key name	<code>chassis_id</code>
<code>is_active</code>	Soft delete: hidden from pickers, kept for history	
<code>created_at</code> , <code>updated_at</code>	Timestamps in UTC	

7.3 Business areas

Area	Main tables	Hub
Users and access	<code>login_users</code> , <code>role_master</code> , <code>module_master</code> , <code>role_module_permission</code> , <code>user_permission_override</code> , <code>station_user_assignment</code>	<code>login_users</code>
Dropdown lists	<code>master_list_registry</code> and 17 <code>*_master</code> tables	<code>master_list_registry</code>
Plant layout	<code>station_master</code> , <code>chassis_location_master</code> , <code>store_master</code> , <code>yard_checkpoint_master</code> , <code>yard_slot_master</code>	<code>station_master</code>
Chassis and yard	<code>chassis_master</code> and its status, movement, document, checklist and hand-over tables	<code>chassis_master</code>
Production	<code>chassis_station_log</code> , <code>chassis_station_handoff</code> , <code>production_hold</code> , <code>delay_reason_master</code> , <code>chassis_bom_snapshot</code>	<code>chassis_station_log</code>
Quality	<code>qa_checksheet_template</code> → <code>qa_checksheet_section</code> → <code>qa_checkpoint_item</code> ; <code>qa_station_template_map</code> ; <code>qa_checksheet_instance</code> → <code>qa_checkpoint_response</code> → <code>qa_response_photo</code> ; <code>qa_section_remark</code> ; <code>qa_checksheet_signoff</code>	<code>qa_checksheet_instance</code>
Dispatch	<code>dispatch_thermal_checksheet</code> , <code>dispatch_gate_pass</code> and their items	
Products and BOM	<code>product_master</code> , <code>bom_master</code> , <code>bom_detail</code> , <code>special_feature_*</code> , <code>vehicle_model_master</code> , <code>body_type_master</code> , <code>customer_segment_master</code> , <code>seating_layout_master</code>	<code>bom_master</code>
Sales and dealers	<code>customer_master</code> , <code>dealer_client_master</code> , <code>dealer_order</code> → <code>dealer_order_vehicle</code> → feature and chassis lines	<code>dealer_order</code>
Purchasing and stock	<code>rm_master</code> , <code>rm_inventory</code> , <code>rm_inventory_log</code> , purchase requests and orders, <code>rm_receiving_log</code> (GRN), <code>vendor_master</code>	<code>rm_master</code>
Vendor portal	<code>asn_log</code> , <code>rejection_log</code> , <code>ncr_log</code> , <code>debit_note</code> , <code>vendor_scorecard</code>	
Maintenance	<code>maintenance_*</code> (14 tables)	<code>maintenance_asset</code>

7.4 Where to find a value

You want	Look in
A chassis and its state	<code>chassis_master</code> (search <code>chassis_no</code>); the five status fields in 6.1
Where a chassis is now	<code>chassis_station_log</code> with <code>event_type = 'ARRIVAL'</code> and <code>status = 'AT_STATION'</code>
Its station history and times	<code>chassis_station_log</code> ordered by <code>scanned_at</code>

You want	Look in
Whether it may go to the next station	Its latest <code>chassis_station_handoff</code> row
Every status change	<code>chassis_status_log</code>
Its planned parts	<code>chassis_bom_snapshot</code>
Its QA results	<code>qa_checksheet_instance</code> → <code>qa_checkpoint_response</code> → <code>qa_response_photo</code> ; <code>qa_section_remark</code> ; <code>qa_checksheet_signoff</code>
Its order and dealer	<code>dealer_order_vehicle_id</code> → <code>dealer_order_vehicle</code> → <code>dealer_order</code> ; <code>dealer_id</code> → <code>customer_master</code>
Values of a dropdown	<code>master_list_registry</code> names the table; the table holds <code>code</code> , <code>label</code> , <code>sort_order</code> , <code>is_active</code>
Station order	<code>station_master.sequence_no</code> (active rows)
Who works at a station	<code>station_user_assignment</code>
Which checksheets run at a station	<code>qa_station_template_map</code>
What a role may do	<code>role_module_permission</code> ; exceptions in <code>user_permission_override</code>
Stock of a part	<code>rm_inventory</code> ; movements in <code>rm_inventory_log</code>
Uploaded files	The row stores a path such as <code>/uploads/qa/...</code> ; the file is under <code>backend/uploads</code>

7.5 Schema evolution

The schema is versioned with Alembic (`backend/migrations/versions`). The current head is `g3b4c5d6e7f8` . The handover-era revisions, in order:

Revision	Change
<code>b2c3d4e5f6a8</code>	State of the original database dump
<code>c7d8e9f0a1b2</code>	First version of administrator-managed dropdown values
<code>e1f2a3b4c5d6</code>	17 dropdown master tables and registry; station staff; QA photos
<code>f2a3b4c5d6e7</code>	Station hand-over (QA gate); QA section remarks
<code>g3b4c5d6e7f8</code>	Descriptions on every table

`alembic upgrade head` applies whatever a database is missing and is safe to repeat; `alembic check` confirms that code and database agree.

8. Security model

8.1 Authentication

Population	Mechanism
Staff	<code>POST /auth/login</code> with employee code or email and password, verified against a bcrypt hash; returns a signed JWT (HS256, key <code>SECRET_KEY</code>); valid for the configured period (30 days by default)
Dealers, vendors	Separate logins issue tokens bound to a session row (<code>dealer_portal_log</code> , <code>vendor_portal_log</code>); logout or expiry invalidates the session on the server
Maintenance team	Staff login, stored separately by the maintenance portal

The browser keeps the token in local storage and sends it as a bearer token on every call. An expired or invalid token returns 401 and the front end returns the user to the sign-in page.

8.2 Authorisation

Every protected endpoint declares its module and action; resolution is described in 6.7. Station actions add the station-access rule (6.8). There is no code path that grants access outside these rules.

8.3 Public surface

Only the sign-in endpoints, `/health` and the QR tracking endpoint (`GET /api/v1/yard/track/{chassis_no}`) answer without a token. The tracking endpoint returns status information only.

8.4 Secrets and network

Item	Handling
Database credentials, signing key	Only in <code>backend/.env</code> , never in code or in the repository; <code>.env.example</code> is the template
Seed passwords for demo data	Supplied as environment variables when running seed scripts
Cross-origin access	Allowed origins listed in <code>ALLOWED_ORIGINS</code> ; <code>ALLOWED_ORIGIN_REGEX</code> admits private network addresses so phones on the plant Wi-Fi can use the app
Transport	The local deployment uses HTTP. A production deployment should use HTTPS (Chapter 13), which also enables camera scanning on phones

9. Code reference

9.1 Repository layout

backend/	
app/main.py	application, CORS, routers under /api/v1, uploads, error handler
app/core/	settings, database, security, logging, numbering
app/models/	ORM models per area; table_catalog.py (table descriptions)
app/modules/	one folder per business module
migrations/	Alembic schema history
scripts/	seed and Excel import scripts, table comments, DB reference generator
tests/	automated tests
uploads/	uploaded files (back up with the database)
frontend/	
src/App.jsx	all routes
src/api/	API hooks per area; client.js
src/components/	layouts, shared UI, station and QA components
src/pages/	screens by area
docs/	this paper's companions: setup guides, database reference
scripts/	macOS setup and start scripts

9.2 Back-end modules

All routes are under `/api/v1`. The live list with request and response shapes is at `/docs` on the running API.

Module	Prefix	Responsibility
auth	/auth	Sign-in, current user and permissions, administrator "view as vendor or dealer"
access_control	/access-control	Users, roles, permission matrix, audit logs
lookups	/lookups	Dropdown lists (registry and master tables)
station	/stations	Station master
station_work	/station-work	Station staff, operator and QA station boards, QA release; <code>flow.py</code> holds the hand-over state machine
yard	/yard	Chassis inward to hand-over, documents, DA-lite, checklists, movements, reports, tracking, production report
production	/production	Station scan, floor dashboard, timeline, overdue, delays, holds, analytics
qa	/qa	Checksheet templates, station mapping, inspections, photos, remarks, sign-off, history
dispatch	/dispatch	Dispatch queue, thermal checksheet, gate pass

Module	Prefix	Responsibility
sales	/sales	Dashboard, customers, clients, transporters, induction, payments, goodwill, spare parts, service orders
dealer_portal	/dealer-portal	Dealer self-service
dealer_portal_admin	/admin/dealer-portal	Staff review, confirmation, chassis allocation
production_masters	/production-masters , /bom-analysis	Vehicle masters, features, seating; BOM coverage
bom	/bom	Products, BOMs, feature BOMs
rm_master , vendor , store , inventory	/rm-master , /vendors , /stores , /inventory	Parts, suppliers, stores, stock
procurement	/procurement	Procurement engine, purchase requests and orders, goods receipt, debit notes, monthly plans
vendor_portal , vendor_portal_admin	/portal , /admin/vendor- portal	Vendor self-service and access
maintenance	/maintenance	Maintenance portal

9.3 Key functions

Function	File	Purpose
scan_chassis	production/service.py	Station check-in and completion; QA gate enforcement; cycle time
next_station , check_arrival , on_completion , set_qa_outcome , release	station_work/flow.py	Hand-over state machine
start_inspection , submit_responses_bulk , signoff_instance	qa/service.py	Inspection lifecycle
_deduct_inventory_on_qa_pass	qa/service.py	Stock consumption at QA pass
simulate_check , confirm_reservation	procurement/engine_service.py	BOM explosion, reservation, shortages
create_chassis , update_da_lite , handover_to_production , station_scan	yard/service.py	Chassis inward to dispatch statuses
induct_chassis_to_production	sales/routes.py	Induction
build_dashboard	sales/dashboard.py	Sales dashboard figures
user_has_permission , require_permission	core/security.py	Authorisation
next_document_number	core/numbering.py	Document numbers

9.4 Screens

Menu group	Screens (file under <code>frontend/src/pages</code>)
Chassis & Yard	Chassis Inward (<code>yard/YardList</code> , <code>YardChassisDetail</code>), Station Scanner, Parts Approvals, Dispatch Queue (<code>dispatch/*</code>), Yard Reports
Production	My Station (<code>station/MyStation</code>), Production Floor (<code>production/ProductionDashboard</code>), Production Schedule (<code>sales/InductionReady</code>), Production Report
Quality	QA Inspection (<code>qa/QaInspect</code>), Quality Dashboard, QA Checksheets (<code>qa/TemplateManager</code> , <code>StationChecklistConfig</code>), Chassis QA History
Sales & Orders	Sales Dashboard, Dealer Orders, Dealer Requests, Service Requests (<code>sales/*</code>)
Purchasing	Monthly Planning, Material Requirements (<code>procurement/ProcurementEngine</code>), Purchase Orders, Goods Received, Debit Notes
Stock	Stock Balance, Stock Movements, Record Material Use, Store Requests (<code>inventory/*</code>)
Setup	Dealers, Vendors, Transporters, Raw Materials, Stores, Stations, Station Staff, Vehicle Templates, Bill of Materials, Production Masters
Admin	Users & Access (<code>admin/AccessControl</code>), System Logs
Portals	<code>dealer-portal/*</code> , <code>vendor-portal/*</code> , <code>maintenance-portal/*</code> ; tracking page <code>yard/ChassisTrackingPage</code>

10. Business processes end to end

10.1 Order to chassis

1. A dealer places an order in the dealer portal, or sales records it: order, vehicle lines, chosen features.
2. Sales reviews and confirms it (**Sales & Orders > Dealer Orders**); the dealer acknowledges any changes; sales gives final confirmation (**CONFIRMED**).
3. Chassis are allocated to the order lines.

10.2 Chassis inward

Chassis & Yard > Chassis Inward.

1. Create the chassis: number, model, dealer, fuel, arrival details (**CHASSIS_IN** , **YARD_PENDING_CHECK**).
2. Upload and verify arrival documents.
3. DA-lite check with receiving photo; a **HOLD** blocks it commercially.
4. Complete the yard QA checklist (**YARD_CHECKED**).
5. Record movements between yard checkpoints; complete the store hand-over sheet and its approvals.
6. Hand over to production (needs the checklist and DA-lite photo): **HANDED_TO_PRODUCTION** .

10.3 Induction and procurement

1. **Production > Production Schedule**: schedule and induct chassis (**PROCUREMENT_PENDING**).
2. **Purchasing > Material Requirements**: run the procurement check (6.4).
3. Shortages become purchase requests and orders; vendors send shipping notices through their portal; stores record goods receipts, which add stock; rejected quantities can raise a debit note valued at the PO unit price.

10.4 Production with QA gates

Operators check chassis in and complete them station by station on **My Station**; QA inspects and releases on **QA Inspection**; supervisors watch **Production > Production Floor**. The rules are in 6.2, 6.3, 6.5 and 6.9.

10.5 Final inspection and dispatch

1. FVI is the last station; its scan sets **READY_FOR_DISPATCH** .
2. **Chassis & Yard > Dispatch Queue**: thermal (AC) checksheet, then gate pass: **DISPATCHED** .
3. The dealer, and anyone with the QR code, can follow the chassis throughout.

10.6 Supporting processes

Process	Summary
Maintenance	Assets, breakdowns with actions and photos, preventive schedule, consumables, monthly stock, machine cost, diesel, LPG and solar registers, equipment purchase and rental
Vendor portal	Purchase orders and acknowledgement, shipping notices, rejections, non-conformance reports, debit notes, monthly plans
Dealer portal	Profile and contacts, clients, orders and edits, templates, requests, chassis status (when enabled)

11. Change impact

If you change	It affects
Station order (<code>sequence_no</code>) or <code>is_active</code>	The route of every chassis, next-station choice, floor view, sales dashboard pipeline
Checksheet-to-station mapping	Whether completing a station waits for QA, and what QA sees there
A checkpoint's mandatory flag	Whether a checksheet can pass
Station staff assignments	Which stations people may act at, and their tabs
A part's station (<code>rm_master.station_id</code>)	Which station consumes it and therefore routing (from the next procurement check)
Feature BOMs	Parts added to chassis that choose that option (from the next procurement check)
Role permissions	Menus, buttons and API access for every user with that role, immediately
A dropdown list	Choices offered on screens; saved records keep their values
<code>da_lite_status</code>	Sales hold queues and dashboard counts
<code>SECRET_KEY</code>	Signs every user out
<code>ALLOWED_ORIGINS</code> / <code>ALLOWED_ORIGIN_REGEX</code>	Which addresses can use the API from a browser

12. Extending the system

12.1 Add a dropdown list

1. Add `'LIST_CODE': 'new_thing_master'` to `MASTER_LIST_TABLES` in `app/models/master_list_models.py`.
2. Add a description to `app/models/table_catalog.py`.
3. Create a migration that creates the table with the standard columns and inserts its registry row (follow revision `e1f2a3b4c5d6`); run `alembic upgrade head`.
4. On screens use `useLookupOptions('LIST_CODE')` or the `LookupOptions` component.

12.2 Add an API endpoint

1. Declare it in the module's `routes.py` with `Depends(require_permission('MODULE', 'action'))`.
2. Put the logic in `service.py` and the request and response models in `schemas.py`.
3. Add a hook in the matching `frontend/src/api/*.js` and invalidate the affected query keys.

12.3 Add a screen

Create the page under `src/pages/<area>/`, add a route in `App.jsx`, and add a menu entry in `components/layout/Layout.jsx` naming the permission module that should show it.

12.4 Add a table or column

1. Change or add the model in `app/models/<area>_models.py`; for a new table add a line to `table_catalog.py`.
2. `alembic revision --autogenerate -m "describe the change"`; review the file, keep only your change.
3. `alembic upgrade head`; `alembic check` should report no differences.
4. `python scripts/apply_table_comments.py` and `python scripts/generate_db_reference.py`.

12.5 Change a status flow

Tracking statuses are set in `yard/service.py` and `production/service.py`, the hand-over in `station_work/flow.py`, QA outcomes in `qa/service.py`. Search the code for the status string to find every reader and writer before changing it.

13. Operations

13.1 Running

Component	Command (from the project folder)
API	<code>cd backend</code> , activate the environment, <code>uvicorn app.main:app --port 8000</code> (add <code>--host 0.0.0.0</code> for phones)
Screens	<code>cd frontend</code> , <code>npm run dev</code>
macOS shortcut	<code>./scripts/run_mac.sh</code> (<code>--lan</code> for phones)

13.2 Back-up and restore

Task	Command
Back up the database	<code>pg_dump -h localhost -U erp_user -d erp_db -Fc -f erp_db_YYYYMMDD.dump</code>
Back up files	Copy <code>backend/uploads</code>
Restore	Create the role and an empty database, then <code>pg_restore -h localhost -U erp_user -d erp_db --no-owner --no-privileges <file></code> , then <code>alembic upgrade head</code>

The restoring `pg_restore` must be version 18 or newer.

13.3 Upgrading

Replace the code, keeping `backend/.env` , `backend/venv` and `backend/uploads` , then: `pip install -r requirements.txt` , `alembic upgrade head` , and `npm install` in `frontend` .

13.4 Monitoring

Source	Shows
<code>backend/logs/error.log</code>	Unexpected server errors with full traceback
Admin > System Logs	User and portal actions, permission changes
<code>/health</code>	API liveness
<code>/docs</code>	Live API contract

13.5 Moving to production hosting

For a server deployment: build the front end (`npx vite build` , output `frontend/dist`) and serve it from a web server; run the API without `--reload` behind the same web server; enable HTTPS (required for phone cameras); restrict `ALLOWED_ORIGINS` to the real address; schedule daily database and uploads back-ups.

14. Quality assurance

Check	How
Automated tests	<code>cd backend</code> , <code>python -m pytest</code> ; tests create and remove their own data and run as an administrator
Code and database agree	<code>alembic check</code> reports "No new upgrade operations detected"
Front end builds	<code>cd frontend</code> , <code>npm vite build</code>
API contract	<code>/docs</code> on the running API

At handover all checks pass, every GET endpoint of the API answers without server errors, and every screen of the staff application loads without errors.

15. Limitations and recommendations

Item	Detail	Recommendation
Phone camera needs HTTPS	Local HTTP setup: phones pick or type chassis numbers	Deploy with HTTPS
New environments from a dump	A database built only from migrations completes but differs slightly in detail	Keep restoring from dumps; consider a consolidated baseline migration
Files on disk	<code>backend/uploads</code> is outside the database	Back up with the database; consider object storage on a server
QA shortages	API and table exist; no screen	Add a screen if shortage logging is required
Two production report columns	<code>dispatch_transporter</code> and <code>pdi_inspector_name</code> are not captured	Capture at dispatch and PDI if needed
Legacy QA tables	<code>station_qa_checklist_item_master</code> , <code>chassis_station_qa_result</code> no longer written	Archive after confirming no reports depend on them
Development servers	Run commands use development servers	Use the production hosting steps in 13.5

Appendix A. Glossary

Term	Meaning
ASN	Advance shipping notice sent by a vendor
BOM	Bill of materials: parts and quantities to build a product
Chassis	The vehicle base supplied by the customer, on which the body is built
CSP	Customer-supplied part
DA-lite	Authorisation check at receipt; HOLD blocks the chassis for payment or documents
Debit note	Amount claimed from a vendor for rejected or short material
FAPS	Fire alarm and protection system fitted to buses
Feature BOM	Parts an ordered option adds to the BOM
FVI	Final vehicle inspection, the last station
Gate pass	Document that lets the bus leave the plant
GRN	Goods receipt note
Induction	Releasing a scheduled chassis into production
JWT	JSON Web Token, the signed sign-in token
NCR	Non-conformance report
PDI	Pre-delivery inspection
PR / PO	Purchase request / purchase order
QA gate	The rule that QA must pass and release a chassis before the next station accepts it
RFD	Ready for dispatch
RTO	Regional transport office
Thermal checksheet	Air-conditioning performance check before dispatch
WIP	Work in progress: the chassis is on the line

Appendix B. Configuration reference

Variable (<code>backend/.env</code>)	Purpose
<code>DATABASE_URL</code>	Application database connection (<code>postgresql+asyncpg://...</code>)
<code>DATABASE_URL_SYNC</code>	Migration connection (<code>postgresql://...</code>)
<code>ALLOWED_ORIGINS</code>	Browser addresses allowed to call the API
<code>ALLOWED_ORIGIN_REGEX</code>	Pattern for further allowed addresses (private networks)
<code>SECRET_KEY</code>	Token signing key; changing it signs everyone out
<code>ALGORITHM</code>	<code>HS256</code>
<code>ACCESS_TOKEN_EXPIRE_MINUTES</code>	Staff token lifetime
<code>REFRESH_TOKEN_EXPIRE_DAYS</code>	Refresh lifetime
<code>DEBUG</code>	Logs every SQL statement when true
<code>VITE_API_URL</code> (<code>frontend/.env</code> , optional)	API address; leave unset to use the page's host on port 8000

Appendix C. Status catalogue

Field	Values
<code>chassis_master.tracking_status</code>	CHASSIS_IN , WIP , READY_FOR_DISPATCH , DISPATCHED
<code>chassis_master.current_status</code>	YARD_PENDING_CHECK , YARD_CHECKED , HANDED_TO_PRODUCTION
<code>chassis_master.procurement_status</code>	PENDING , PROCUREMENT_PENDING , PROCUREMENT_CHECKED , SHORTAGE_HOLD
<code>chassis_master.da_lite_status</code>	PENDING , OK , HOLD
<code>chassis_master.production_hold_status</code>	NONE , ON_HOLD , RELEASED
<code>chassis_station_log.event_type</code>	ARRIVAL , COMPLETION
<code>chassis_station_handoff.status</code>	AWAITING_QA , QA_PASSED , RELEASED , RECEIVED , SUPERSEDED
<code>qa_checksheet_instance.station_qa_status</code>	IN_PROGRESS , then the sign-off result
<code>dealer_order.status</code>	DRAFT , SUBMITTED , PENDING_DEALER_ACK , DEALER_EDITED , PENDING_FINAL_CONFIRMATION , CONFIRMED
<code>station_user_assignment.role</code>	OPERATOR , QA
Permission modules	BOM, DEALER_PORTAL_ADMIN, INVENTORY, LOCATIONS, MAINTENANCE, PROCUREMENT, PRODUCTION, PRODUCTION_MASTERS, QA, RM_MASTER, ROLE_MANAGEMENT, SALES, STORE, SYSTEM_LOGS, USER_MANAGEMENT, VENDOR, VENDOR_PORTAL_ADMIN, YARD