VMBS OPERATIONS

Command Center Documentation Set

Layered, versioned reference · Version 1.0

2026-07-06 · Owner: Nicola Anderson, COO

Document Map

Where to go — the whole set at a glance.

VMBS Operations Documentation

EXECUTIVE

- Operations Command Center — User Guide

OPERATIONS TEAMS

- SLA Monitor
- Reconciliation
- Risk & Audit
- Branch Defects
- ICT Availability
- Cards Operations (connected system)
- ATM Operations (connected system)
- Credit Operations (connected system)

ADMINISTRATORS

- System Administration Guide

DEVELOPERS

- Technical / Architecture Guide

Contents

The set

Documentation Index — v1.0 (index)

Level 1 — Command Center

Operations Command Center — User Guide — v1.0

Level 2 — Subsystem Guides

SLA Monitor — User Guide — v1.0

Reconciliation — User Guide — v1.0

Risk & Audit — User Guide — v1.0

Branch Defects — User Guide — v1.0

ICT Availability — User Guide — v1.0

Cards Operations — User Guide — v1.0 draft (template)

ATM Operations — User Guide — v1.0 draft (template)

Credit Operations — User Guide — v1.0 draft (template)

Subsystem Guide — Template (template)

Level 3 — Administration

System Administration Guide — v1.0

Level 4 — Technical

Technical / Architecture Guide — v1.0

VMBS Operations Command Center — Documentation Set

This folder holds the complete, versioned documentation for the VMBS Operations Command Center and the operations systems it launches. It is organised as a **layered documentation set** — the same discipline an enterprise software product uses — so that each reader finds the one document written for them, and so the set can evolve alongside the software.

Document map

```
VMBS Operations Documentation
│
├── Executive
│    └── Operations Command Center — User Guide
│
├── Operations Teams
│    ├── SLA Monitor
│    ├── Reconciliation
│    ├── Risk & Audit
│    ├── Branch Defects
│    ├── ICT Availability
│    ├── Cards Operations        (connected system)
│    ├── ATM Operations          (connected system)
│    └── Credit Operations       (connected system)
│
├── Administrators
│    └── System Administration Guide
│
└── Developers
     └── Technical / Architecture Guide
```

How the set is organised

There are **four levels**, each aimed at a different audience:

#	Document	Audience	Answers
1	**Operations Command Center — User Guide**	Executives, administrators	What the portal is, how to read it, how to run it day-to-day
2	**Subsystem User Guides**	Each system's operators	How to use one specific operations system
3	**System Administration Guide**	Platform support	Installing, configuring, registering systems, troubleshooting, backup
4	**Technical / Architecture Guide**	Developers	Architecture, data model, APIs, the Pulse interface, correlation engine, security, deployment

Every subsystem guide follows the **same template** ([`subsystems/_TEMPLATE.md`](./subsystems/_TEMPLATE.md)) — Purpose · Users · Screen overview · Filters · KPIs · Analysis views · Reports · Common workflows · FAQs — so they are uniform to read and cheap to maintain.

A note on scope

This repository is the **Command Center** — the *launcher and executive-brief layer*. Each operations system (SLA Monitor, Reconciliation Tracker, etc.) is a **separate application with its own deployment and its own repository**; the Command Center links to it and reads a small summary ("pulse") from it. Because of that split:

- The **Command Center**, **Administration**, and **Technical/Architecture**

guides are authoritative and complete — that software lives here.

- Each **Subsystem guide** documents (a) what that system is for and (b) the

metrics the Command Center reads from it, which is verified against this codebase. Screen-by-screen detail internal to each subsystem app is owned by that app's team; where a section depends on the subsystem's own UI it is marked **[owned by the subsystem team]** so it is completed from the source of truth rather than guessed here.

Version register

Documentation is versioned independently of the software, starting at v1.0. Bump the **minor** version for additive edits, the **major** version for a restructure or a change that invalidates prior instructions. Record every bump in the document's own *Revision history* table and here.

Document	Version	Date	Status
Operations Command Center — User Guide	v1.0	2026-07-06	Current
SLA Monitor — User Guide	v1.0	2026-07-06	Current
Reconciliation — User Guide	v1.0	2026-07-06	Current
Risk & Audit — User Guide	v1.0	2026-07-06	Current
Branch Defects — User Guide	v1.0	2026-07-06	Current
ICT Availability — User Guide	v1.0	2026-07-06	Current
Cards Operations — User Guide	v1.0 (draft)	2026-07-06	Connected-system template
ATM Operations — User Guide	v1.0 (draft)	2026-07-06	Connected-system template
Credit Operations — User Guide	v1.0 (draft)	2026-07-06	Connected-system template
System Administration Guide	v1.0	2026-07-06	Current
Technical / Architecture Guide	v1.0	2026-07-06	Current

Conventions used across the set

- **Command Center** — this portal (the product; internal name *VMBS Command Center*).
- **System / subsystem** — one operations application the Command Center launches (SLA Monitor, Reconciliation, etc.).
- **Built-in system** — one of the five systems the Command Center reads with a bespoke adapter (SLA, Reconciliation, Risk & Audit, Branch Defects, ICT).
- **Connected system** — a system added at runtime from Settings (e.g. Cards, ATM, Credit) that follows the standard **Pulse contract**.
- **Pulse** — the small JSON summary a system exposes so the Command Center can place it in the Executive Brief.
- **Executive Brief** — the synthesised, cross-system situational read shown

above the launcher grid.

Owner: Nicola Anderson, COO.

Operations Command Center — User Guide

Product: VMBS Command Center **Document version:** v1.0 **Audience:** Executives and administrators **Last updated:** 2026-07-06

---

1. Purpose

The Operations Command Center is the **single front door** to the VMBS operations suite. From one page you can:

- **Open every operations system** — one tile per system, each linking straight

to the live application.

- **Read a cross-system executive brief** — a short, plain-English situational

summary synthesised across all systems, so leadership sees the state of operations without opening each app.

Two design principles shape everything else in this guide:

1. **The Command Center holds no data of its own.** Each system owns and

displays its own live data. The portal links to systems and reads a small summary from them; it never becomes a second copy of the numbers.

1. **It requires no ongoing maintenance to keep the links working.** Adding or

re-pointing a system is done from Settings in the browser — no code, no deployment.

Think of it as the lobby of the operations building: it tells you which floor to go to, and it posts a one-page situational board by the door.

2. Navigation

The page has three regions, top to bottom:

1. **App bar** (top) — the VMBS Command Center brand, and a **gear icon**

(top-right) that opens **Settings**. Settings is the only place that requires the admin password; everything else is open to any viewer.

1. **Executive Brief** — the synthesised situational read (Section 4).
2. **Operations systems grid** — the launcher tiles (Section 3), with a small

header showing *"X of Y live"* (how many systems currently have a link configured).

To open a system, **click its tile**. A tile with no link configured shows a grey **"Coming soon"** badge and is not clickable.

3. Opening a system (the launcher grid)

Each tile shows the system's name, a one-line description, a colour accent, and an **Open ↗** hint. The five **built-in** systems are:

Tile	Opens	One-line description
Operations SLA Monitor	the SLA app	Service levels across operations
Reconciliation Tracker	the reconciliation app	GL, suspense & settlement recons
Risk & Audit Register	the risk & audit app	KRIs and audit remediation
Branch Defect & Standards Tracker	the branch-defects app	Defects & operational standards by branch
ICT System Availability	the ICT app	Uptime & EOL / EOS lifecycle

Below the built-ins, any **Connected Systems** you have added (e.g. Cards, ATM, Credit) appear as additional tiles in the same style.

Each system opens in the same browser tab by default. Use your browser's back button to return to the Command Center.

4. The Executive Brief (cross-system intelligence)

Above the grid, the Command Center shows a short **Executive Brief** — a single situational read synthesised across every system. It turns the portal from a link menu into a leadership dashboard.

What you see

- **A status line** — the bottom line in one sentence, e.g. *"Mostly fine — 4 of

5 areas healthy, one needs attention. Most urgent: ageing IT systems."*

- **A ranked list of items**, most urgent first. Each item is a colour-coded

one-liner you can click to expand:

- 🔴 red / 🟠 orange / 🟡 amber — needs attention, in order of severity.
- 🟢 green — improving or steady, shown after the attention items.
- Expanding an item reveals the reasoning ("*Why it matters*" / "*What to

do*") and an **Open** → link to the underlying system.

- **A cross-system item** — when the *same* entity (e.g. a branch) is flagged by

more than one system, the brief surfaces it as one line: *"Two branches flagged in more than one system."* Expanding it lists each branch and where it was flagged. This is the Command Center's most valuable output: it connects dots no single app can see on its own.

- **Blind spots** — if a system cannot be read, it is named as a *blind spot*

("No data from: …") rather than being silently dropped. An unreadable system is itself a finding.

How to read it well

- The **status line first** tells you whether to relax or dig in.
- Then scan the **coloured items top to bottom** — they are already ranked by

severity, so the first red item is the most important thing in operations right now.

- Treat a **cross-system item as a priority** — a shared cause across two

systems usually matters more than two isolated issues.

- A **blind spot is an action**, not noise: it means a system needs its link or

its pulse restored (see the Administration Guide).

What it is — and is not

- It is **deterministic and auditable**: the same data in always produces the

same brief. There is **no AI/LLM** and **nothing is sent to any external service** — the brief is computed by fixed rules from numbers the systems already publish.

- It is a **summary, not the source of truth**. For any figure, open the system

behind it. The brief points you to where to look; the system holds the detail.

- It **never blocks the launcher**. If the brief cannot be built, the tiles

still work.

The technical detail of how the brief is produced (pull → store → synthesise → narrate) is in the **Technical / Architecture Guide**.

5. Settings

Open Settings with the **gear icon** (top-right) and enter the **admin password** (default `pa55w0rd` — change it, see below). Settings is where the whole portal is configured, and — on a properly hosted deployment — a save applies **for every user at once**.

Settings has three areas:

1. **System paths** — the URL or intranet path each built-in system opens. Leave

a field blank to show that system as *"Coming soon"*. **Reset built-in paths** returns them to the shipped defaults.

1. **Connected Systems** — add systems that are not built in (see Section 6).
2. **Admin password** — set a new password, or reset it to the default.

A banner at the top of Settings tells you the **scope** of your saves:

- *"Changes here update the Command Center for everyone who uses it."* — a

shared server store is available; saves are **global**.

- *"No server store detected — changes save on this device only."* — the host

is read-only and has no shared store; saves fall back to **this browser**. The Administration Guide explains how to make saves global.

> **Change the default password before go-live.** The default (`pa55w0rd`) is public in this documentation; anyone who knows it can re-point systems.

6. Adding connected systems

To add a system that is not one of the five built-ins (for example **Cards**, **ATM**, or **Credit** operations):

1. Open **Settings** → **Connected Systems** → **Add a system**.
2. Fill in:

- **Name** — e.g. *ATM Network*.
- **Short description** — e.g. *Uptime & cash availability*.
- **Link (URL or /path)** — where the tile opens.
- **Pulse URL** *(optional)* — a summary endpoint so the system also appears

in the Executive Brief (see below).

1. **Save changes.** The tile appears immediately, for every user (on a hosted

deployment).

You can add up to **20** connected systems.

Making a connected system appear in the Brief

A connected system with only a link is a **launcher tile**. To also give it a row in the Executive Brief, point its **Pulse URL** at an endpoint that returns this JSON:

```
{
  "status": "healthy | watch | critical",
  "headline": "42 ATMs offline",
  "detail": "optional longer text",
  "series": [10, 20, 30],
  "tags": ["Portmore", "May Pen"],
  "goodDirection": "up | down"
}
```

- `status` and `headline` are **required** — they colour and label the row.
- `detail` is the expandable body.
- `series` enables a **trend** ("rose from 10 to 30").
- `tags` let the system **correlate** with others — a branch flagged in both

ATM *and* Branch Defects surfaces as one cross-system item.

- `goodDirection` says whether rising is good (default: down is good).

Connected systems are **first-class**: they go through the same synthesis engine as the built-ins. The exact contract and examples are in the **Technical / Architecture Guide** and the Connected-System subsystem guides (**Cards** · **ATM** · **Credit**).

7. Frequently asked questions

Does the Command Center store our operational data? No. It holds links, connected-system definitions, and the admin password. The Executive Brief keeps only a small rolling snapshot of each system's *headline metric* so it can show trends; the underlying systems are never modified.

A tile says "Coming soon" — is the system down? No — it means no link is configured for that system yet. Set its path in Settings.

The Brief says "No data from …" — what does that mean? The Command Center could not read that system's pulse (wrong link, the system was unreachable, or it returned something unexpected). The launcher tile still works; see the Administration Guide's troubleshooting section.

I saved a change but a colleague doesn't see it. Your deployment saved on-device (the Settings banner will have said so). To make saves global, the host needs a shared store — see the Administration Guide.

Is anything sent to the cloud / an AI service? No. The brief is computed by fixed rules on your own servers. There is no LLM and no external call other than the Command Center reading each system you have configured.

Who owns this? Nicola Anderson, COO.

Revision history

Version	Date	Change
v1.0	2026-07-06	Initial guide.

SLA Monitor — User Guide

Document version: v1.0 **Audience:** Operations service-level owners, team leads, executives **Last updated:** 2026-07-06 **System URL:** set in Command Center → Settings (default `https://vm-ecru.vercel.app/`)

> Sections marked **[owned by the subsystem team]** describe the SLA app's own screens and should be completed from that application, which is a separate deployment from the Command Center. The KPI and pulse sections are verified against the Command Center codebase.

---

1. Purpose

The **Operations SLA Monitor** tracks service levels across operations against their targets — turnaround times, availability commitments, and other measured service standards. It answers one executive question: *are we meeting the service levels we committed to, and if not, which ones and by how much?*

2. Users

- **Service-level owners** — record each measure's actuals and explain misses.
- **Team leads** — review attainment, chase owners on breaches.
- **Executives** — read overall attainment and the count of measures in breach

(surfaced in the Command Center brief).

3. Dashboard / screen overview [owned by the subsystem team]

Describe the measures list, per-measure trend, and target-vs-actual views.

4. Filters [owned by the subsystem team]

Describe period, category, and owner filters.

5. KPIs

The Command Center reads the SLA app's `/api/data`, which returns a set of **measures**, each with a `target` and a monthly series of actuals (`m[]`).

KPI	Definition	Calculation	Bands (Command Center read)
Compliance / attainment %	Overall service-level attainment for the latest month with figures	**Mean of the actuals across all measures** for the latest populated month, ×100	`< 95%` → **critical**; `< 98%` (or **>2 measures in breach**) → **watch**; otherwise **healthy**
Measures in breach	How many measures are below their target this month	Count of measures where the latest actual `< target`	Contributes to the watch band above
Measures tracked	Total measures under management	Count of measures	Context

Notes on the calculation (as implemented in the Command Center adapter):

- Attainment uses the **mean of actuals**, not the share hitting target. With

very high targets (e.g. 99.99%) a "share hitting target" reads as near-zero and misleads; mean attainment is the honest read.

- The **latest month with figures** is used — a trailing empty month is skipped

so it never reads as 0%.

- With no figures at all, the system is reported as **unknown** (a blind spot),

never as 0%.

6. Analysis views [owned by the subsystem team]

Per-measure trend, category roll-ups, worst-performers.

7. Reports [owned by the subsystem team]

Monthly SLA attainment report; breaches with owner and commentary.

8. Common workflows

1. **Record the month's actuals** — enter each measure's actual for the period.

2. **Review breaches** — filter to measures below target; confirm owner and

root cause.

1. **Chase or accept** — for each breach, chase the owner or formally accept the

risk (the Command Center frames this as *"Chase the owners on the service levels being missed, or accept the risk."*).

9. FAQs

Why does overall attainment differ from "how many hit target"? Attainment is the mean of actuals across measures, which reflects *how close* you are, not just pass/fail — more useful when targets are very high.

A month shows no data — is that a breach? No. An empty latest month is skipped; if nothing is populated the system reads as *unknown* in the brief, prompting a data-entry fix rather than a false alarm.

10. How this system appears in the Command Center

- **Launcher tile:** *Operations SLA Monitor* — "Service levels across

operations".

- **Executive Brief headline: compliance %** (higher is better). Breach

template: *"Service levels are being missed — N of M service levels are below target (overall X%)."*

- **Domain:** Compliance.
- **Pulse endpoint:** built-in adapter → `GET /api/data` on the SLA app.

Revision history

Version	Date	Change
v1.0	2026-07-06	Initial guide.

Reconciliation — User Guide

Document version: v1.0 **Audience:** Reconciliation officers, finance operations, executives **Last updated:** 2026-07-06 **System URL:** set in Command Center → Settings (default `https://asu-beige.vercel.app/`)

> Sections marked **[owned by the subsystem team]** describe the Reconciliation app's own screens and should be completed from that application, which is a separate deployment from the Command Center. The KPI and pulse sections are verified against the Command Center codebase.

---

1. Purpose

The **Reconciliation Tracker** follows GL, suspense, and settlement reconciliations — how many items remain outstanding, their value, and across how many accounts. It answers: *how much unreconciled money is sitting open, and is the backlog clearing or growing?*

2. Users

- **Reconciliation officers** — maintain the weekly workbook, clear items.
- **Finance operations leads** — review the outstanding backlog and value.
- **Executives** — read the outstanding items/value trend in the brief.

3. Dashboard / screen overview [owned by the subsystem team]

This is a **workbook-based** system (a maintained `.xlsx`), not a form app. Describe the workbook layout and any published view.

4. Filters [owned by the subsystem team]

Describe week/account filtering in the workbook or published view.

5. KPIs

The Reconciliation system is a **workbook app**. The Command Center reads its published workbook (or the built-in static workbook) and reduces it to per-week totals; it does not parse a JSON API. Metrics are taken from the **latest week with activity** (a trailing empty week is ignored).

KPI	Definition	Calculation	Bands (Command Center read)
Outstanding items	Unreconciled items open in the latest active week	Sum of open items for that week	`> 3000` → **watch**; otherwise **healthy**
Outstanding value (JMD)	Value of those open items	Sum of value for that week	Reported alongside items
Accounts	Accounts with open items	Count for that week	Context
Week-over-week change	Change in items vs the previous active week	`items[latest] − items[previous]`	Drives the "growing / high" framing

The **headline metric** the Command Center trends is **outstanding items** (lower is better), across the workbook's per-week series.

6. Analysis views [owned by the subsystem team]

Backlog-over-time, ageing of outstanding items, by-account breakdown.

7. Reports [owned by the subsystem team]

Weekly reconciliation status; aged outstanding items with value.

8. Common workflows

1. **Update the weekly figures** — record this week's outstanding items, value,

and accounts in the workbook.

1. **Clear items** — reconcile and close outstanding items; the next week's

totals should fall.

1. **Escalate settlement risk** — where the backlog grows, the Command Center

frames the decision as *"Clear the outstanding items, or accept the settlement risk."*

9. FAQs

Why does the brief sometimes lag the latest week? It reads the **latest week with activity**; a blank future week is intentionally skipped so an empty week never reads as "zero outstanding".

Where does the value come from? It is summed from the workbook's per-week value column (JMD) and shown in the brief as J$…M / J$…B for readability.

10. How this system appears in the Command Center

- **Launcher tile:** *Reconciliation Tracker* — "GL, suspense & settlement

recons".

- **Executive Brief headline: outstanding items** (lower is better).

Template: *"Reconciliation backlog is high/growing — N items still unreconciled (J$…) across M accounts."*

- **Domain:** Reliability.
- **Pulse endpoint:** built-in adapter → reads the app's workbook

(`/api/data` returning `xlsxB64`, else the static workbook) and reduces it to per-week totals server-side.

Revision history

Version	Date	Change
v1.0	2026-07-06	Initial guide.

Risk & Audit — User Guide

Document version: v1.0 **Audience:** Risk officers, internal audit, executives **Last updated:** 2026-07-06 **System URL:** set in Command Center → Settings (default `https://my-risk.vercel.app/`)

> Sections marked **[owned by the subsystem team]** describe the Risk & Audit app's own screens and should be completed from that application, which is a separate deployment from the Command Center. The KPI and pulse sections are verified against the Command Center codebase.

---

1. Purpose

The **Risk & Audit Register** tracks key risk indicators (KRIs) against their limits and audit remediation actions against their due dates. It answers: *which key risks are over their limit, and which audit actions are overdue?*

2. Users

- **Risk officers** — maintain the risk register and KRI bandings per period.
- **Internal audit** — track findings and remediation actions to closure.
- **Executives** — read breached KRIs and overdue actions in the brief.

3. Dashboard / screen overview [owned by the subsystem team]

Describe the risk register, KRI banding view, and the audit-actions tracker.

4. Filters [owned by the subsystem team]

Describe period, risk category, and action-status filters.

5. KPIs

The Command Center reads the Risk & Audit app's `/api/data`, which returns `periods`, an entered-risk list `er[]` (each with a `bands[]` series per period), and an `audit[]` list (each with a `status`).

KPI	Definition	Calculation	Bands (Command Center read)
Key risks over limit	KRIs banded "Breach" in the latest period	Count of `er[]` where the latest `bands` value is `Breach`	Drives severity (see below)
Overdue audit actions	Audit actions past their due date	Count of `audit[]` with status `Past Due`	Any overdue → **critical**
Open audit actions	Actions still open	Count of `audit[]` with status `Open` or `Past Due`	Context
Open risks	Total risks on the register	Count of `er[]`	Context

Status logic (as implemented):

- **Critical** if **any** audit action is overdue, or **more than 3** key risks

are over their limit.

- **Watch** if there is at least one key risk over its limit.
- **Healthy** otherwise.

The **headline metric** trended over periods is the count of **key risks over limit** (lower is better).

6. Analysis views [owned by the subsystem team]

KRI trend by period, breaches by category, ageing of open audit actions.

7. Reports [owned by the subsystem team]

Risk register export; audit remediation status with due dates and owners.

8. Common workflows

1. **Update KRI bandings** — set each risk's band for the current period.
2. **Progress audit actions** — move actions through Open → closed; watch for

any slipping to Past Due.

1. **Close the gap** — the Command Center frames the decision as *"Close out the

overdue audit actions and the risks over their limit, or accept the gap."*

9. FAQs

Why does one overdue action turn the whole area critical? Overdue audit actions are treated as a hard signal — a single one is enough to raise the area to critical in the brief, by design.

What counts as a "key risk over limit"? A risk whose band for the latest period is `Breach`.

10. How this system appears in the Command Center

- **Launcher tile:** *Risk & Audit Register* — "KRIs and audit remediation".
- **Executive Brief headline: key risks over limit** (lower is better).

Template: *"Risk and audit need attention — N key risks are over the limit, and M audit actions are overdue."*

- **Domain:** Risk.
- **Pulse endpoint:** built-in adapter → `GET /api/data` on the Risk & Audit app.

Revision history

Version	Date	Change
v1.0	2026-07-06	Initial guide.

Branch Defects — User Guide

Document version: v1.0 **Audience:** Branch operations, standards/quality team, executives **Last updated:** 2026-07-06 **System URL:** set in Command Center → Settings (default `https://defect-m2uv.vercel.app/`)

> Sections marked **[owned by the subsystem team]** describe the Branch Defect & Standards app's own screens and should be completed from that application, which is a separate deployment from the Command Center. The KPI and pulse sections are verified against the Command Center codebase.

---

1. Purpose

The **Branch Defect & Standards Tracker** records defects and operational- standards checks by branch: how many items were reviewed, how many had defects, how many are resolvable/resolved, and how many recur. It answers: *how good is branch execution, which branches are worst, and are defects being cleared?*

2. Users

- **Branch operations** — record reviews, defects, and resolutions per branch.
- **Standards / quality team** — track the defect rate and recurring items.
- **Executives** — read the overall defect rate and worst branches in the brief.

3. Dashboard / screen overview [owned by the subsystem team]

Describe the per-branch defect grid, period selector, and area breakdown.

4. Filters [owned by the subsystem team]

Describe period, branch, and area filters.

5. KPIs

The Command Center reads the Defects app's `/api/data`, which returns a `defects` object with `periods`, `branches`, `areas`, and `rows` (columns: `period, branch, area, reviewed, instances, defects, resolvable, resolved, recurring`). Metrics use the **latest period with activity** (empty future months are skipped).

KPI	Definition	Calculation	Bands
Defect rate % *(primary)*	Share of checked items that had defects	`defects ÷ instances × 100` for the latest active period	`< 2%` on target, `< 5%` within limit, `< 10%` elevated, `≥ 10%` breach
Resolution rate %	Share of resolvable defects actually resolved	`resolved ÷ resolvable × 100`	Higher is better
Recurring	Defects that keep recurring	Sum of `recurring` in the latest period	`> 5` contributes to **watch**
Unresolved	Open resolvable defects	`max(0, resolvable − resolved)`	Feeds per-branch correlation

Command Center status logic:

- **Critical** if defect rate `≥ 10%`.
- **Watch** if defect rate `≥ 5%` **or** recurring `> 5`.
- **Healthy** otherwise.

Per-branch entities: for the latest period, each branch with open defects (`resolvable − resolved > 0`) is emitted as a correlation entity tagged with the branch name — this is what lets a branch appear as *"flagged in more than one system"* when it also shows up in ICT (or a connected system).

The **headline metric** trended over periods is the **defect rate %** (lower is better).

6. Analysis views [owned by the subsystem team]

Defect-rate trend, worst branches, by-area breakdown, recurring-defect list.

7. Reports [owned by the subsystem team]

Monthly branch-standards report; worst-branch and recurring-defect exports.

8. Common workflows

1. **Record a review** — enter reviewed/instances/defects for a branch and area.
2. **Resolve defects** — mark resolvable defects resolved; watch recurring ones.
3. **Focus the worst branches** — the Command Center frames the decision as

"Focus the worst branches to bring defects down, or accept the quality risk."

9. FAQs

Which period does the headline use? The latest period with any instances recorded — matching the app's own "last active" period, so empty future months don't dilute the figure.

Why is a branch showing in the cross-system item? It has open defects here *and* was flagged by another system (e.g. an ICT outage) in the same window — a shared signal worth investigating together.

10. How this system appears in the Command Center

- **Launcher tile:** *Branch Defect & Standards Tracker* — "Defects &

operational standards by branch".

- **Executive Brief headline: defect rate %** (lower is better). Template:

"Branch defect rate is high — X% of checked items had defects (N of M)."

- **Domain:** Quality. Contributes **branch-name entities** to cross-system

correlation.

- **Pulse endpoint:** built-in adapter → `GET /api/data` on the Defects app.

Revision history

Version	Date	Change
v1.0	2026-07-06	Initial guide.

ICT Availability — User Guide

Document version: v1.0 **Audience:** ICT operations, infrastructure owners, executives **Last updated:** 2026-07-06 **System URL:** set in Command Center → Settings (default `https://ict-eta.vercel.app/`)

> Sections marked **[owned by the subsystem team]** describe the ICT app's own screens and should be completed from that application, which is a separate deployment from the Command Center. The KPI and pulse sections are verified against the Command Center codebase.

---

1. Purpose

ICT System Availability tracks system uptime and the lifecycle (end-of-life / end-of-support) status of the estate. It answers two questions: *are systems available as committed?* and *how many systems are ageing out of support?*

2. Users

- **ICT operations** — record availability and outage events per system/branch.
- **Infrastructure owners** — track EOL/EOS dates and plan replacements.
- **Executives** — read availability and ageing-systems risk in the brief.

3. Dashboard / screen overview [owned by the subsystem team]

Describe the availability trend, per-system status, and lifecycle (EOL/EOS) register.

4. Filters [owned by the subsystem team]

Describe month, system/application, and branch filters.

5. KPIs

The Command Center reads the ICT app's `/api/data`, which returns `months`, an `overall[]` availability series, a `lifecycle[]` list (each with an `eos` date), `applications`, and per-branch outage events under `branches`.

KPI	Definition	Calculation	Bands
Availability %	Overall uptime in the latest month	Last value of `overall[]` × 100	`< 98%` → **critical**
Past end-of-life	Systems already unsupported	Count of `lifecycle[]` whose `eos` date is in the past	Any `> 0` → **critical**
EOL within 90 days	Systems losing support soon	Count of `lifecycle[]` with `eos` in the next 90 days	Any `> 0` → **watch**
EOL within 180 days	Longer-horizon ageing	Count with `eos` in the next 180 days	Context
Systems	Applications tracked	Count of `applications`	Context

Command Center status logic (in priority order):

1. **Critical** if any system is **past EOL**, or availability `< 98%`.
2. **Watch** if any system loses support **within 90 days**.
3. **Healthy** otherwise.

Lifecycle risk is treated as a distinct *forward* risk: even when the area is critical for another reason, the ageing-systems finding is kept so it can't be hidden.

Per-branch entities: outage events in the latest month (where `met = false` or availability `< 99.9%`) are emitted as correlation entities tagged with the branch/location — enabling *"flagged in more than one system"* when the same branch appears in Branch Defects.

The **headline metric** trended over months is **availability %** (higher is better).

6. Analysis views [owned by the subsystem team]

Availability trend, outages by branch, EOL/EOS runway by system.

7. Reports [owned by the subsystem team]

Monthly availability report; lifecycle/replacement-plan register.

8. Common workflows

1. **Record monthly availability** and any outage events by system/branch.
2. **Maintain the lifecycle register** — keep `eos` dates current per system.

3. **Plan replacements** — the Command Center frames the decision as *"Plan to replace or upgrade the ageing systems, or formally accept the risk."*

9. FAQs

Why does an ageing system still show even when availability is fine? Lifecycle (EOL/EOS) is a separate, forward-looking risk — the brief keeps it visible independently of current uptime.

What counts as an outage for correlation? A branch event where the target wasn't met, or availability fell below 99.9%, in the latest month.

10. How this system appears in the Command Center

- **Launcher tile:** *ICT System Availability* — "Uptime & EOL / EOS lifecycle".
- **Executive Brief headline: availability %** (higher is better), plus a distinct **lifecycle** finding. Templates: *"IT availability is low — systems were available X% of the time."* and *"Ageing IT systems — N systems are already unsupported, and M more lose support within 90 days."*
- **Domain:** Lifecycle. Contributes **branch-name entities** to correlation.
- **Pulse endpoint:** built-in adapter → `GET /api/data` on the ICT app.

Revision history

Version	Date	Change
v1.0	2026-07-06	Initial guide.

Cards Operations — User Guide

Document version: v1.0 (draft — connected-system template) **Audience:** Cards operations team, executives **Last updated:** 2026-07-06 **System URL:** added in Command Center → Settings → Connected Systems

> **Cards Operations is a* connected system*. **It is not one of the five built-in systems; it is onboarded at runtime from the Command Center's Settings and appears in the Executive Brief by publishing a** Pulse **endpoint that follows the standard contract. This guide documents that integration. Screen-level detail of the Cards app itself is** [owned by the subsystem team]** and completed from that application.

---

1. Purpose

Cards Operations monitors the cards estate — for example authorisation health, declines, disputes/chargebacks, and card-production/issuance backlogs. It answers: *is the cards service healthy, and what needs attention today?*

2. Users

- **Cards operations team** — run the day-to-day; own the underlying app.
- **Executives** — read the cards headline in the Command Center brief.

3. Dashboard / screen overview [owned by the subsystem team]

4. Filters [owned by the subsystem team]

5. KPIs

Define the cards KPIs (e.g. authorisation success %, decline rate, open disputes, issuance backlog) with calculations and bands. Choose **one headline metric** to expose to the Command Center via the Pulse endpoint.

6. Analysis views [owned by the subsystem team]

7. Reports [owned by the subsystem team]

8. Common workflows [owned by the subsystem team]

9. FAQs

How does Cards get into the Executive Brief? By publishing a Pulse endpoint and pasting its URL into the connected-system's **Pulse URL** field in Settings (Section 10).

10. How this system appears in the Command Center

Onboard Cards Operations from **Settings → Connected Systems → Add a system**:

- **Name:** *Cards Operations*
- **Short description:** e.g. *Authorisations, disputes & issuance*
- **Link:** the Cards app URL
- **Pulse URL:** an endpoint returning the standard contract:

```
{
  "status": "healthy | watch | critical",
  "headline": "312 disputes open, 18 aged",
  "detail": "Optional longer explanation shown when the row is expanded.",
  "series": [280, 300, 312],
  "tags": ["Portmore", "May Pen"],
  "goodDirection": "down"
}
```

- `status` + `headline` are **required** (colour + one-liner).
- `series` gives a trend; `tags` (e.g. branch names) join cross-system

correlation; `goodDirection` says whether rising is good.

- Connected systems go through the **same synthesis engine** as the built-ins,

so trends and correlation work identically. With no Pulse URL, Cards is a launcher tile only.

Full contract and behaviour: see the **Technical / Architecture Guide** and the Command Center guide's **Adding connected systems**.

Revision history

Version	Date	Change
v1.0 (draft)	2026-07-06	Initial connected-system template.

ATM Operations — User Guide

Document version: v1.0 (draft — connected-system template) **Audience:** ATM operations team, executives **Last updated:** 2026-07-06 **System URL:** added in Command Center → Settings → Connected Systems

> **ATM Operations is a* connected system ***— onboarded at runtime from Settings and surfaced in the Executive Brief via a standard*** *Pulse* ***endpoint. This guide documents that integration; the ATM app's own screens are*** *[owned by the subsystem team]**.

---

1. Purpose

ATM Operations monitors the ATM network — uptime, cash availability (out-of-cash events), and fault/outage counts by location. It answers: *how many ATMs are down or out of cash right now, and where?*

2. Users

- **ATM operations team** — run the network; own the underlying app.
- **Executives** — read the ATM headline in the Command Center brief.

3. Dashboard / screen overview [owned by the subsystem team]

4. Filters [owned by the subsystem team]

5. KPIs

Define the ATM KPIs (e.g. ATM uptime %, ATMs offline, out-of-cash events, mean time to repair) with calculations and bands. Choose **one headline metric** to expose via the Pulse endpoint (e.g. *ATMs offline*, lower is better).

6. Analysis views [owned by the subsystem team]

7. Reports [owned by the subsystem team]

8. Common workflows [owned by the subsystem team]

9. FAQs

How does ATM get into the Executive Brief? Publish a Pulse endpoint and paste its URL into the connected system's **Pulse URL** field in Settings (Section 10).

10. How this system appears in the Command Center

Onboard from **Settings** → **Connected Systems** → **Add a system**:

- **Name:** *ATM Operations* (or *ATM Network*)
- **Short description:** e.g. *Uptime & cash availability*
- **Link:** the ATM app URL
- **Pulse URL:** an endpoint returning the standard contract:

```
{
  "status": "healthy | watch | critical",
  "headline": "42 ATMs offline",
  "detail": "Optional longer explanation shown when the row is expanded.",
  "series": [10, 20, 42],
  "tags": ["Portmore", "May Pen"],
  "goodDirection": "down"
}
```

- `status` + `headline` are **required**; `series` gives a trend; `tags`

(branch/location names) join cross-system correlation — an ATM location that also appears in Branch Defects or ICT surfaces as one *"flagged in more than one system"* item.

- ATM goes through the **same synthesis engine** as the built-ins. No Pulse URL

→ launcher tile only.

Full contract: see the **Technical / Architecture Guide**.

Revision history

Version	Date	Change
v1.0 (draft)	2026-07-06	Initial connected-system template.

Credit Operations — User Guide

Document version: v1.0 (draft — connected-system template) **Audience:** Credit operations team, executives **Last updated:** 2026-07-06 **System URL:** added in Command Center → Settings → Connected Systems

> **Credit Operations is a* connected system ***— onboarded at runtime from Settings and surfaced in the Executive Brief via a standard*** *Pulse* ***endpoint. This guide documents that integration; the Credit app's own screens are*** *[owned by the subsystem team]**.

---

1. Purpose

Credit Operations monitors credit processing — for example application turnaround, disbursement backlog, arrears/past-due, and exceptions. It answers: *is credit processing keeping pace, and where is the risk building?*

2. Users

- **Credit operations team** — run processing; own the underlying app.
- **Executives** — read the credit headline in the Command Center brief.

3. Dashboard / screen overview [owned by the subsystem team]

4. Filters [owned by the subsystem team]

5. KPIs

Define the credit KPIs (e.g. average turnaround days, applications in backlog, % past due, exceptions open) with calculations and bands. Choose **one headline metric** to expose via the Pulse endpoint.

6. Analysis views [owned by the subsystem team]

7. Reports [owned by the subsystem team]

8. Common workflows [owned by the subsystem team]

9. FAQs

How does Credit get into the Executive Brief? Publish a Pulse endpoint and paste its URL into the connected system's **Pulse URL** field in Settings (Section 10).

10. How this system appears in the Command Center

Onboard from **Settings** → **Connected Systems** → **Add a system**:

- **Name:** *Credit Operations*
- **Short description:** e.g. *Turnaround, backlog & arrears*
- **Link:** the Credit app URL
- **Pulse URL:** an endpoint returning the standard contract:

```
{
  "status": "healthy | watch | critical",
  "headline": "Backlog 210 apps, TAT 4.2 days",
  "detail": "Optional longer explanation shown when the row is expanded.",
  "series": [180, 195, 210],
  "tags": ["Portmore", "May Pen"],
  "goodDirection": "down"
}
```

- `status` + `headline` are **required**; `series` gives a trend; `tags` join

cross-system correlation; `goodDirection` says whether rising is good.

- Credit goes through the **same synthesis engine** as the built-ins. No Pulse

URL → launcher tile only.

Full contract: see the **Technical / Architecture Guide**.

Revision history

Version	Date	Change
v1.0 (draft)	2026-07-06	Initial connected-system template.

<System Name> — User Guide

Document version: v1.0 **Audience:** <who uses this system day to day> **Last updated:** YYYY-MM-DD **System URL:** <link, or "set in Command Center → Settings">

> This guide follows the standard VMBS subsystem template. Keep the section order identical across every subsystem so the set stays uniform and easy to maintain. Sections marked **[owned by the subsystem team]** describe the system's own screens and must be filled in from the subsystem application itself, which is a separate deployment from the Command Center.

---

1. Purpose

What this system is for, in two or three sentences. The operational decision it exists to support.

2. Users

Who uses it and for what — e.g. operations officers (data entry), team leads (review), executives (the headline). Note who has edit rights vs. read-only.

3. Dashboard / screen overview [owned by the subsystem team]

The main screens, what each shows, and how to move between them. A labelled screenshot per screen is ideal.

4. Filters [owned by the subsystem team]

Every filter the system offers (period, branch, area, status, …), what each does, and how they combine.

5. KPIs

The metrics this system reports, each with: **definition**, **how it is calculated**, **the bands / thresholds** that make it green/amber/red, and **the period** it is measured over. The KPIs the

Command Center reads from this system (its *pulse*) are listed explicitly — these are verified against the Command Center codebase and must stay in sync with it.

6. Analysis views [owned by the subsystem team]

Trend, breakdown, comparison, and drill-down views — what question each answers.

7. Reports [owned by the subsystem team]

Exports and printable reports: what they contain, format, and how to generate them.

8. Common workflows

Step-by-step for the handful of things users do most (e.g. "record this month's figures", "clear an outstanding item", "close an action"). Numbered steps.

9. FAQs

The recurring questions, answered plainly.

10. How this system appears in the Command Center

- **Launcher tile:** name, description, and where the tile links.
- **Executive Brief:** the headline metric the Command Center reads, the bands

that decide healthy/watch/critical, and any entities (e.g. branch names) it contributes to cross-system correlation.

- **Pulse endpoint:** the URL/shape the Command Center pulls (built-in adapter,

or the standard Pulse contract for connected systems).

Revision history

Version	Date	Change
v1.0	YYYY-MM-DD	Initial guide.

System Administration Guide

Product: VMBS Command Center **Document version:** v1.0 **Audience:** Whoever installs, hosts, and supports the platform (GICT / IT) **Last updated:** 2026-07-06

---

1. Overview for administrators

The Command Center is a **Next.js 15** application (App Router, React 18, TypeScript). It has a small footprint and, deliberately, **no database**:

- It stores only a small **config** (system links, connected-system definitions,

and the admin password hash) and a small rolling **snapshot store** used to draw trends in the Executive Brief.

- Both persist either to **Vercel Blob** (on serverless/read-only hosts) or to a

local JSON file (on an intranet Node host) — auto-detected.

Your job as administrator is to: host it, point it at each operations system, keep the shared store writable, secure the admin password, and restore visibility when a system goes dark.

2. Installing / configuring

Requirements

- **Node.js 18+** (for `next start` on an intranet host), or a **Vercel**

project.

- Network path from **wherever the Command Center runs** to **each operations

system's URL **— the Executive Brief pulls each system** server-to-server*, *so the* server*, not the user's browser, must be able to reach them.

Local / intranet host

```
npm install
npm run build
npm run start      # serves on http://localhost:3000 (put a reverse proxy in front)
```

Ensure the working directory has a **writable, persistent `./data` folder** — that is where the shared config and snapshots live (Section 4). Override the location with `CC_DATA_FILE` if needed.

Vercel host

Deploy the repo as a Vercel project. Connect a **Vercel Blob** store to the project so saves are global (Section 4). No other services are required.

Environment variables

Variable	Purpose	When to set
`BLOB_READ_WRITE_TOKEN`	Enables Vercel Blob as the shared store (global saves on serverless). Added automatically when a Blob store is connected.	Vercel / serverless hosts
`CC_DATA_FILE`	Overrides the config file path (default `./data/command-center.json`).	Intranet host, if `./data` isn't suitable
`CC_SNAPSHOTS_FILE`	Overrides the brief snapshot-store path (default alongside the config file).	Rarely

> The Blob token is also detected if any environment variable's value begins with `vercel_blob_rw_`, matching how the other VMBS apps are wired.

3. Registering new systems

There are three ways links reach the Command Center, in order of precedence:

1. **In-app Settings (recommended, runtime, global).** Gear icon → admin

password → set each system's URL/path → **Save**. On a host with a shared store this updates the portal **for every user at once**. This is the normal way to register or re-point a system.

1. **Build-time default — `public/config.js`.** Sets

`window.__VMBS_LINKS__` as the initial default before any admin edit. Useful to preset links at upload time.

1. **Baked fallback — `components/modules.tsx` → `DEFAULT_LINKS`

(`lib/defaults.ts`).** The final fallback compiled into the app.

Registering a *connected* system (Cards, ATM, Credit, …)

Settings → **Connected Systems** → **Add a system**: name, description, link, and an optional **Pulse URL**. It appears as a tile immediately and, with a Pulse URL, also in the Executive Brief. Up to **20** connected systems. See the **Technical Guide** for the Pulse contract, and the connected-system subsystem guides for per-system setup.

4. Where configuration persists (storage model)

The Command Center keeps two small stores; both auto-select the same backend:

Store	File (intranet)	Blob key (serverless)	Contents
Config	`./data/command-center.json` (`CC_DATA_FILE`)	`command-center/config.json`	`links`, `custom` (connected systems), `passwordHash`
Snapshots	`./data/cc-snapshots.json` (`CC_SNAPSHOTS_FILE`)	*(intranet file only)*	rolling per-system pulse history for trends (capped ~240/system)

Backend selection (in `lib/serverConfig.ts`):

- **Blob token present** → Vercel Blob (global for every user; works on

read-only hosts).

- **No token** → local filesystem (`./data`). Writable + persistent → global for

every user on that host.

- **Neither writable** → the client falls back to **per-device** saves, and the

Settings banner says so.

> **The single most common admin issue** is a save that only sticks on one device. That means no shared store is available: on Vercel, connect a Blob store; on intranet, make `./data` writable and persistent across restarts.

5. Pulse URLs

The Executive Brief is built by **pulling** each system:

- **Built-in systems** are read by bespoke adapters at each app's `GET /api/data`

(Reconciliation is read from its workbook). You configure only the **base URL**; the adapter knows the path.

- **Connected systems** are read from the **Pulse URL** you paste in Settings,

which must return the standard contract (`status`, `headline`, optional `detail`/`series`/`tags`/`goodDirection`).

Operational notes:

- Pulls are **server-to-server** — the *Command Center's server* must reach each

system. A link that works in a user's browser but not from the server will read as a **blind spot**.

- An **intranet path** (e.g. `/sla`) is fine for the *launcher tile* but **cannot

be pulled **for the brief — the brief needs an** absolute `http(s)` URL**.

- Each pull has a **6-second timeout**; a slow system reads as *timed out* (a

blind spot) rather than hanging the page.

- To keep trends flowing on a schedule, hit **`GET /api/brief?force=1`** from a

cron (Vercel Cron or intranet cron). Normal page loads collect at most once per **10 minutes**.

6. Authentication

- A **single admin password** guards Settings. Default: `pa55w0rd` — **change it

before go-live** (Settings → Admin password). The default is published in this documentation.

- The password is stored **hashed (SHA-256)** in the config store — never in

plaintext. Verification hashes the input and compares.

- **Reset to default** is available in Settings if the password is lost (this

restores `pa55w0rd`; change it again immediately).

- There is no per-user login — this is an internal operations portal. Restrict

reach with your network/reverse-proxy controls and, if needed, front it with your existing SSO/proxy. Viewing the portal and the brief requires no password; only **editing** does.

7. Troubleshooting

Symptom	Likely cause	Fix
Tile shows **"Coming soon"**	No link configured for that system	Settings → set the system's URL/path
Brief says **"No data from X"**	Server can't reach X, X was slow (>6s), X returned non-JSON, or X's link is an intranet *path*	Use an absolute URL reachable **from the server**; confirm X's `/api/data` (or Pulse) responds with JSON
Saves don't apply for other users	No shared store	Vercel: connect a Blob store; intranet: make `./data` writable/persistent
Brief trends look flat	Only one pull collected so far	Schedule `GET /api/brief?force=1`; trends also seed from each system's own in-app history
"Incorrect password" when you're sure it's right	Password changed on another device with no shared store, or store not readable	Confirm the shared store; use **Reset password to default** as last resort
Brief never appears	Brief build failed	The launcher still works by design; check server logs for `/api/brief`, confirm systems are reachable
Connected system won't enter the brief	Missing/invalid **Pulse URL**	Ensure the Pulse URL returns the contract with at least `status` + `headline`

Diagnostics:

- `GET /api/config` — returns the current `links` + `custom` (what the portal

thinks it's pointing at).

- `GET /api/brief` — returns the current brief JSON, including `blindspots`.
- `GET /api/brief?force=1` — forces a fresh collection cycle.

8. Backup and recovery

Everything that needs backing up is **two small JSON blobs**:

- **Config:** `command-center.json` (links, connected systems, password hash).
- **Snapshots:** `cc-snapshots.json` (brief trend history — nice to keep, not

critical; it self-rebuilds over time).

Backup procedure:

- **Intranet host:** include the `./data` directory (or the `CC_DATA_FILE` /

`CC_SNAPSHOTS_FILE` locations) in your normal file backups.

- **Vercel Blob:** periodically download `command-center/config.json` from the

Blob store (e.g. via `GET /api/config` for a live snapshot of links + custom; the password hash is only in the Blob object itself).

Recovery:

- Restore the config JSON to the same location/key; the portal picks it up on

the next read (a tiny 3-second cache means changes appear almost immediately).

- If config is lost entirely, the portal falls back to the **baked defaults**

(`DEFAULT_LINKS`, default password) — re-enter links and reset the password from Settings. No operational data is lost because the Command Center never held any.

9. FAQs

Do I need a database? No — two JSON stores, no DB, no schema migrations.

Can I run it fully on the intranet with no cloud? Yes — `next start` on a Node host with a writable `./data`. Blob is only for serverless/read-only hosts.

Does it phone home / use AI? No. The only outbound calls are the server reading the systems you configured. The brief is rule-based, no LLM.

How do I force a brief refresh for a demo? `GET /api/brief?force=1`.

Revision history

Version	Date	Change
v1.0	2026-07-06	Initial guide.

Technical / Architecture Guide

Product: VMBS Command Center **Document version:** v1.0 **Audience:** Developers maintaining or extending the platform **Last updated:** 2026-07-06

---

1. Overall architecture

The Command Center is a **Next.js 15 App Router** application (React 18, TypeScript). It is deliberately thin: a launcher UI, an admin-config API, and a synthesis pipeline that builds the Executive Brief by **pulling** each operations system server-to-server. It owns **no operational data** and has **no database**.

```
Browser
  │  GET /                      → CommandCenter (launcher + Settings)
  │  GET /api/config            → links + connected systems
  │  POST /api/config           → admin edits (password-gated)
  │  GET /api/brief             → the synthesised Executive Brief
  ▼
Next.js server (Node runtime)
  ├─ lib/serverConfig.ts        shared config store (Blob | file)
  └─ lib/synthesis/             the brief pipeline
       collect.ts   pull each system (server-to-server)
       apps.ts      built-in adapters (SLA, recon, risk, defects, ict)
       recon.ts     xlsx → per-week totals (SheetJS)
       store.ts     rolling snapshot history (trends)
       engine.ts    trend → weight → correlate → prioritise
       nlg.ts       template library + firing rules (no LLM)
  ▼
Operations systems (separate deployments)
   SLA · Reconciliation · Risk & Audit · Branch Defects · ICT · + connected
   each serves its own GET /api/data (or a Pulse endpoint / workbook)
```

Key properties:

- **Pull-based, no coupling.** The systems are never modified. The Command

Center reads what each already publishes.

- **No LLM, no external calls.** The brief is computed by deterministic rules on

your own server; identical input → identical output.

- **Fail-soft.** An unreadable system becomes a *blind spot* finding; the brief

never blocks the launcher.

Source layout

Path	Responsibility
`app/page.tsx`, `app/layout.tsx`	Entry; renders `CommandCenter`.
`app/api/config/route.ts`	GET/POST config (password-gated writes).
`app/api/brief/route.ts`	GET the brief; `?force=1` forces collection.
`components/CommandCenter.tsx`	Launcher grid + Settings modal (client).
`components/ExecutiveBrief.tsx`	Renders `/api/brief` (client).
`components/modules.tsx`	`APPS` registry (the five built-in tiles).
`lib/defaults.ts`	`DEFAULT_LINKS`, `CustomApp`, `sanitizeCustom`.
`lib/serverConfig.ts`	Shared config store (Blob or filesystem).
`lib/synthesis/*`	The brief pipeline (see below).

2. Data model

The Command Center persists only three things, all small JSON.

Config store (`Store` in `lib/serverConfig.ts`)

```
interface Store {
  links?: Record<string, string>;   // built-in system key → URL/path
  passwordHash?: string;            // SHA-256 of the admin password
  custom?: CustomApp[];             // connected systems
}
```

`CustomApp` (`lib/defaults.ts`, max 20, sanitised on every read/write):

```
interface CustomApp {
  id: string;
```

```
  title: string;      // ≤ 60 chars
  blurb: string;      // ≤ 80 chars
  url: string;        // launcher target, ≤ 500 chars
  pulseUrl?: string;  // optional brief source, ≤ 500 chars
  accent?: string;
}
```

Built-in link keys are fixed: `sla`, `recon`, `risk`, `defects`, `ict` (`LINK_KEYS`). `mergeLinks()` always overlays saved links on `DEFAULT_LINKS`, so a missing key falls back to its default.

Snapshot store (`Snapshot` in `lib/synthesis/engine.ts`)

Each pull of each system is stored as a `Snapshot` (status, severity, metrics, entities, and the system's own in-app `series`). History is capped at ~240 snapshots per system (`store.ts`) and used to compute trends. This is the Command Center's *only* memory of the numbers — a rolling summary, not a copy of the systems' data.

3. APIs

`GET /api/config`

Returns `{ links, custom }` — the merged current configuration. No auth (read).

`POST /api/config`

Body: `{ password, links?, custom?, newPassword?, resetPassword? }`.

- `password` is required and verified (`verifyPassword`, SHA-256) → **401** if

wrong.

- Applies link edits, connected-system edits, password change, or reset, then

persists. Returns `{ ok, links, custom, persisted }`. `persisted:false` means the store wasn't writable (client falls back to device-local).

`GET /api/brief` · `GET /api/brief?force=1`

Runs `collectAndBrief()` and returns the `BriefResponse` (pre-rendered strings: `statusLine`, `items[]`, `blindspots[]`, `hasData`). `?force=1` forces a fresh collection cycle regardless of the 10-minute minimum — use it from a scheduler.

Both routes are `runtime = "nodejs"` and `dynamic = "force-dynamic"` (they touch the filesystem / do server-to-server fetches and must never be statically cached).

4. The Pulse interface

Two ways a system feeds the brief:

A. Built-in adapters (`lib/synthesis/apps.ts`). Each built-in system already serves its full dataset at `GET /api/data`; a bespoke `normalise()` reduces that to headline metrics + correlation entities + an in-app `series`, and `assess()` maps metrics to `{ status, severity }`. Reconciliation is the exception: it is a workbook app, so `fetch: { kind: "xlsx" }` tells the collector to read the `.xlsx` and reduce it via `recon.ts`. The per-system KPI/band definitions are documented in each subsystem guide and live in `SYNTH_APPS`.

B. Connected-system Pulse contract (the standard path for new systems). A connected system publishes an endpoint returning:

```
{
  "status": "healthy | watch | critical",   // required
  "headline": "42 ATMs offline",             // required
  "detail": "optional expandable text",
  "series": [10, 20, 30],                    // optional → trend
  "tags": ["Portmore", "May Pen"],           // optional → correlation
  "goodDirection": "up | down"               // optional (default: down is good)
}
```

`pullCustomOne()` (`collect.ts`) maps this into the same `Snapshot` / `AppSynthDef` shape the built-ins use, so **connected systems are first-class**: `series` yields a trend, and each `tag` becomes a correlation entity. Severity is derived from `status` (critical 72 / watch 48 / healthy 12). An unreachable Pulse simply omits the system from the brief (its tile still shows).

Collection cycle (`lib/synthesis/collect.ts`)

1. Read existing snapshots; collect a fresh cycle if `force`, or if the last

collection is older than **10 minutes** (`MIN_CYCLE_MS`), or if there is no history yet.

1. For each built-in: resolve the base URL from config, pull `/api/data` (or the

workbook), `normalise` + `assess`. Non-`http(s)` links, non-JSON responses, HTTP errors, and a **6-second timeout** all produce a **blind spot** snapshot (`status: "unknown"`) rather than an omission.

1. Persist the cycle best-effort (`store.appendCycle`) — but **synthesize from

the in-memory snapshots** regardless, so trends still work on read-only hosts where the write is a no-op (each snapshot carries its own in-app `series`).

1. Pull connected systems (`pullCustom`), then `synthesize` over built-ins +

connected, and `buildBrief`.

5. The correlation & synthesis engine

`lib/synthesis/engine.ts` is *judgement as code* — no language, no LLM.

- ** `trend()` / `trendFromSeries()` ** — prefer the latest snapshot's embedded

in-app `series`; else the metric across the Command Center's own pulls. Computes direction, magnitude, **streak** (consecutive same-direction steps), and `sentiment` (`improving` / `worsening` / `neutral`) relative to the metric's `goodDirection`. `from` is the value at the *start of the current run*, so "fell from A to B over N periods" is always internally consistent.

- ** `correlate()` **— groups entities by** tag** across systems, ignoring

generic tags (`branch`, `system`, `sla`, …). A tag hitting **≥2 systems** becomes an `incident` finding (e.g. a branch flagged in Defects *and* ICT). Severity scales with how many systems share it.

- ** `synthesize()` ** — builds a per-system finding, folds systems already

captured by an incident (except lifecycle risk, which is always kept as a distinct forward risk), then ranks:

- **attention** = incidents + bad standalone findings ≥ severity 40, sorted by

severity.

- **good** / **steady** = the rest.
- **domain roll-up** across the fixed domains (quality, reliability,

compliance, risk, lifecycle); domains that can't be read are counted apart so an all-unreadable estate never reads as "stable".

- **key number** = the most *sustained* worsening trend (streak ≥ 2).
- **overall** = `awaiting data` / `stable` / `stable, one issue` /

`attention needed`.

Narration (`lib/synthesis/nlg.ts`)

A **template library with firing rules**. Each `Template` has a `when()` predicate and a `priority`; `selectTemplate()` picks the highest-priority match. Every sentence is assembled from computed numbers — plain, everyday language (system jargon mapped via `SHORT`/`AREA`), a **RAG** dot per item derived from severity, a punchy `summary` (collapsed row) and a `detail` (expanded body with *Why it matters* / *What to do*). Cross-system incidents render as one grouped item with a sub-list. `buildBrief()` returns fully pre-rendered strings so the client (`ExecutiveBrief.tsx`) does no logic beyond toggling rows.

Determinism: no clocks or randomness in the ranking/narration; same snapshots in → same brief out. This is what makes it auditable.

6. Security

- **Admin edits are password-gated** (`POST /api/config`, 401 on mismatch). The

password is stored **SHA-256 hashed**, never plaintext; reset restores the default hash.

- **No operational data at rest** — only links, connected-system definitions,

the password hash, and rolling metric snapshots.

- **No external/AI calls** — the only outbound requests are the server reading

the configured systems. Nothing is sent anywhere.

- **Input is sanitised** — `sanitizeCustom()` bounds field lengths and count

(≤20); link fields are trimmed; the config route only accepts known link keys.

- **Server-to-server pulls** use a 6-second timeout and `AbortController`;

failures degrade to blind spots, so a hostile or dead endpoint can't hang the page.

- **Threat notes for deployers:** the SHA-256 password hash is unsalted and the

portal has no per-user auth or rate-limiting — treat it as an *internal* portal behind your network controls / reverse proxy / SSO, not an internet-facing app. Pulse URLs are fetched by the server, so validate/curate who can add connected systems (only admins can, via the password gate).

7. Deployment

- **Stack:** Next.js 15 + React 18 + TypeScript; deps `@vercel/blob` (shared

store) and `xlsx`/SheetJS (recon workbook). Hand-written CSS design system (Sora + IBM Plex Sans) to match the VMBS apps.

- **Serverless (Vercel):** connect a **Blob store** → `BLOB_READ_WRITE_TOKEN`

is injected → global saves on a read-only filesystem. The brief synthesizes from in-memory snapshots each request, so trends still work without a writable disk. Add a **Vercel Cron** hitting `GET /api/brief?force=1` for steady trend history.

- **Intranet Node host:** `npm run build && npm run start`; ensure a **writable,

persistent `./data`** (or set `CC_DATA_FILE` / `CC_SNAPSHOTS_FILE`). Use an OS cron for `?force=1`. Put a reverse proxy in front for TLS and access control.

- **Config precedence at runtime:** server store (Settings) → `public/config.js`

(`window.__VMBS_LINKS__`) → baked `DEFAULT_LINKS`.

Build / develop

```
npm install
npm run dev      # http://localhost:3000
npm run build    # production build
npm run lint
```

8. Extending the platform

- **Add a built-in system:** add a tile to `APPS` (`components/modules.tsx`) with

a matching key in `LINK_KEYS`/`DEFAULT_LINKS`, then add an `AppSynthDef` to `SYNTH_APPS` (`apps.ts`) with `normalise()` + `assess()` for its `/api/data` shape. Prefer this only when a system's data doesn't fit the Pulse contract.

- **Add a system without code:** onboard it as a **connected system** from

Settings and have it publish a Pulse endpoint (Section 4B). This is the intended path for Cards, ATM, Credit, and future systems.

- **Tune thresholds / mappings:** per-system bands live in each `assess()` and

the field mappings in each `normalise()` in `apps.ts`. Narration wording lives in `TEMPLATES` (`nlg.ts`).

Revision history

Version	Date	Change
v1.0	2026-07-06	Initial guide.