CMPDI RI-4 Digital Operations Platform

1. Purpose of This Document

This document serves as the **foundational reference** for the CMPDI RI-4 Digital Operations Platform. It consolidates all architectural, technological, SDLC, governance, and design decisions discussed so far. Any future design, development, or review must align with this document.

---

2. Vision & Scope

CMPDI RI-4 Digital Operations Platform is an **internal, enterprise-grade, multi-module system** intended to digitize operational, residential, and campus services for CMPDI RI-4.

Target Platforms

- Android (mobile)
- iOS (mobile)
- Web browsers (desktop & mobile)

Key Characteristics

- API-first
- Role-based access control (RBAC)
- Modular and scalable
- Secure and auditable

---

3. High-Level Architecture

Architectural Style

Single API-only Backend using Modular Monolith architecture

Reasons: - Single organization, shared identity - Reduced operational complexity - Strong data consistency - Easier governance and auditing

Client–Server Model

```
Flutter Apps (Android / iOS / Web)
        ↓
API-only Backend (FastAPI / Spring Boot)
        ↓
Modular Domain Services
        ↓
Relational Database (PostgreSQL)
```

---

4. Domain & Module Structure

4.1 Top-Level System

CMPDI RI-4 Digital Operations Platform - Unified authentication - Global dashboard - Centralized identity & roles

4.2 Level-1 Domains

A. Enterprise Operations Domain (formerly Office)

Handles all official and operational systems.

Modules: 1. **ICT Asset & Infrastructure Management System (AIMS)** - Asset inventory - Software licenses - Maintenance and audits

2. **Fleet & Transport Management System (FTMS)**
    - Vehicle booking
    - Driver allocation
    - Trip, fuel, and maintenance logs

B. Residential & Campus Services Domain (formerly Colony)

Handles employee residential and campus-related services.

Modules: 1. **Residential Maintenance & Complaints System (RMCS)** - Complaint registration and tracking - Sub-units: - Electrical Maintenance Unit - Civil Maintenance Unit

2. **Campus Access & Visitor Management System (CAVMS)**
    - Visitor entry/exit
    - Gate access control
    - Security logs

---

5. Roles & Identity Model

Centralized Identity

- Single user identity across the platform
- No duplicate users per module

Decentralized Role Assignment

- Roles are **module-specific**
- Permissions enforced strictly by backend

Example:

```
User: Employee X
Roles:
- FTMS → Requestor
- RMCS → Resident
```

```
- CAVMS → Resident
- AIMS → No access
```

Governance Levels

- Platform Admin (system-wide authority)
- Domain Admin
- Module Admin
- Manager / Supervisor
- Operator / Staff
- End User

---

6. Backend Design Principles

Backend Type

- API-only
- Stateless
- JSON-based

Authentication

- JWT (access + refresh tokens)
- No server-side sessions

Backend Structure (Logical)

```
core/
  ├── auth
  ├── users
  ├── roles & permissions
  └── global dashboard

domains/
  ├── enterprise_operations/
  │    ├── aims
  │    └── ftms
  └── residential_services/
       ├── rmcs
       └── cavms

shared/
  ├── database
  ├── security
  └── logging
```

---

7. Database Strategy

Core Tables (Shared)

- users
- roles
- permissions
- user_role_mapping
- audit_logs

Domain-Owned Tables

- assets, vehicles, bookings
- complaints, maintenance_units
- visitors, gate_logs

Rules: - No cross-domain table coupling - Clear ownership per domain

---

8. Frontend Strategy

Primary Choice: Flutter

- Single codebase for Android, iOS, and Web
- Mobile-first experience
- Web used mainly for internal access

State Management

- Riverpod or Bloc
- Clear separation: UI → State → API

Flutter Module Mapping

Each Flutter module maps directly to a backend domain.

---

9. SDLC & Tooling Stack

Selected Toolchain

- GitHub Repository (source of truth)
- GitHub Issues (requirements & tasks)
- GitHub Projects (Kanban workflow)
- GitHub Wiki (documentation)
- GitLab (CI/CD execution)

SDLC Workflow

```
Backlog → Ready → In Progress → Code Review → Testing → Done
```

Traceability Rule

```
Requirement → Issue → Branch → Commit → PR → CI → Deploy
```

---

10. CI/CD Strategy

GitHub → GitLab Flow

- GitHub: planning and source control
- GitLab: build, test, scan, deploy

Pipeline Stages

- Lint
- Test
- Build
- Security Scan
- Deploy

Merges blocked on CI failure.

---

11. Audit, Logging & Compliance

Mandatory logging: - Authentication attempts - Role and permission changes - Approvals and rejections - Deletions and updates

Purpose: - Accountability - Security - Administrative trust

---

12. Error Handling & API Contract

- Standard response format
- Clear error codes
- User-safe messages
- Dev-safe logs

Frontend must never infer permissions.

---

13. Performance & Scalability Principles

- Pagination by default
- Indexed queries
- Scale-readiness (Redis later)
- No premature microservices

---

14. Offline & Reliability Considerations

- Graceful handling of poor networks
- Draft support where applicable (complaints, requests)
- Sync-on-reconnect strategy

---

15. Backup & Disaster Recovery

- Scheduled database backups
- Secure off-site storage
- Defined restore procedures

---

16. Guiding Rules (Non-Negotiable)

1. Backend is the source of truth
2. Roles are enforced server-side
3. If it's not documented, it doesn't exist
4. No feature without audit consideration
5. Simplicity over buzzwords

---

17. Auth + RBAC + Audit Database Schema

17.1 Core Authentication Tables

users - id (PK) - employee_id - name - email - phone - status (active, suspended) - created_at - updated_at

auth_credentials - user_id (FK → users.id) - password_hash - last_login - failed_attempts

---

17.2 Role-Based Access Control (RBAC)

roles - id (PK) - name (Platform Admin, Module Admin, Manager, User) - scope (platform / domain / module)

permissions - id (PK) - code (FTMS_CREATE_BOOKING, RMCS_APPROVE_COMPLAINT) - description

role_permissions - role_id (FK) - permission_id (FK)

user_roles - user_id (FK) - role_id (FK) - domain - module

Rule: Permissions are evaluated server-side only.

17.3 Audit & Compliance Tables

audit_logs - id (PK) - user_id - action - entity_type - entity_id - timestamp - ip_address

Every sensitive action must generate an audit record.

---

18. Module-wise Permission Matrix

18.1 Fleet & Transport Management System (FTMS)

Role	Permissions
Platform Admin	Full access
Transport Admin	Configure vehicles, drivers
Manager	Approve / reject bookings
Driver	View assigned trips
Employee	Create booking request

18.2 Residential Maintenance & Complaints System (RMCS)

Role	Permissions
Colony Admin	Full access
Electrical Supervisor	Manage electrical complaints
Civil Supervisor	Manage civil complaints
Technician	Update work status
Resident	Raise complaints

18.3 Campus Access & Visitor Management System (CAVMS)

Role	Permissions
Security Admin	Configure gates
Gate Operator	Check-in / check-out
Resident	Approve visitors
Visitor	Limited temporary access

---

19. API Response & Error Contract

19.1 Standard Success Response

```
{
  "success": true,
  "data": {},
  "message": "Operation successful"
}
```

19.2 Standard Error Response

```
{
  "success": false,
  "error": {
    "code": "FTMS_403",
    "message": "Access denied"
  }
}
```

19.3 Error Principles

- No stack traces to client
- Stable error codes
- User-safe messages

---

20. Flutter App Structure Aligned With Domains

```
lib/
├── core/
│   ├── auth
│   ├── network
│   ├── routing
│   └── theme
├── domains/
│   ├── enterprise_ops/
│   │   ├── ftms
│   │   └── aims
│   └── residential_services/
│       ├── rmcs
│       └── cavms
├── shared/
│   ├── widgets
│   └── utils
└── main.dart
```

Rules: - One Flutter domain = one backend domain - No cross-domain UI dependencies

---

21. CI/CD & Environment Strategy

21.1 Environments

- Development
- Staging
- Production

21.2 GitHub → GitLab Flow

- GitHub: source control, planning
- GitLab: CI/CD execution

21.3 CI/CD Pipeline Stages

- Lint
- Test
- Build
- Security Scan
- Deploy

21.4 Environment Rules

- Secrets via environment variables
- No production access from dev builds
- CI failure blocks merge

---

22. Final Enforcement Rules

1. No code without issue
2. No issue without role mapping
3. No feature without audit consideration
4. Backend is the single source of truth

This document is now the authoritative system blueprint.