TECHNICAL SPECIFICATION

1. Purpose

Build **IT Service Firm Suite — MVP (v0.2.1)** to support SMB IT service contracts by providing:

1. **MonitorHub**: endpoint inventory + monitoring dashboard + incidents + ticketing + in-app notifications + client portal
2. **RemoteAssist**: **explicit end-user consent** remote support sessions with **VIEW-ONLY** streaming (no keyboard/mouse)
3. **Windows agent**: Windows 10/11 tray app that registers endpoints, uploads metrics, and runs **allowlisted** diagnostics only.

---

2. Product overview

MVP goals

- Inventory and monitor Windows 10/11 endpoints for multiple client organizations.
- Generate incidents from alert rules and manage tickets linked to devices/incidents.
- Provide **consent-based** remote support sessions with **view-only** screen streaming and chat.
- Provide a **ClientViewer** portal with strictly scoped read-only views (org-only).

In-scope (MVP)

- Orgs (client companies), optional sites/branches, users/roles.
- Devices inventory + status + last-seen + basic HW/OS/IP.
- Agent: register + token + heartbeat + metrics batch upload + diagnostics allowlist runner.
- Metrics: CPU/RAM/Disk free/Uptime + agent version + last reboot (where available).
- Alert rules: threshold + duration + severity; incident open/resolve (ack implied by UI task).
- Ticketing: create/assign/close + comments; link to device and/or incident.
- RemoteAssist: request → end-user approves → view-only session + technician chat; user stop + timeout.
- Client portal: devices/incidents/tickets scoped to their org; client ticket create/comment is “recommended”.

Out-of-scope (post-MVP)

- Full remote input control, unattended access, patch mgmt, software deployment, AV/EDR actions, vuln scanning, SIEM/MDM, advanced discovery.
- SaaS superadmin across orgs enabled by default (architecture-ready only).

Locked decisions (non-negotiable)

- **Remote control: VIEW-ONLY** in MVP.
- **Target OS: Windows 10/11 only** in MVP.
- **Alert channels: in-app only** (no email/SMS/Telegram).
- **Client portal: YES** (ClientViewer, org-scoped).
- **Agent install: manual installer** (MSI/EXE) with onboarding code.
- **Deployment: VPS (Ubuntu) + Docker Compose** (API, DB, Redis, Celery, Nginx).

---

3. Users, roles, and permissions + permission matrix

Definitions

- **Organization (Org):** a client company being managed/monitored.
- **MSP Console:** web mode for OrgAdmin/Technician to operate across org context.
- **Client Portal:** web mode for ClientViewer, strictly scoped to their org.

Roles

- **OrgAdmin (MSP side):** manage org settings, users, devices, policies.
- **Technician (MSP side):** view devices, run diagnostics, manage tickets, start sessions.
- **ClientViewer (client side):** read-only views of their org's devices/incidents/tickets; no commands/sessions.

Permission matrix (MVP)

Legend: ✅ allowed, ❌ denied, ⚠️ limited/scope rules

Capability	OrgAdmin	Technician	ClientViewer
Authenticate (login/me)	✅	✅	✅
Manage organizations list/detail	✅	⚠️ (view)	❌
Manage org users (CRUD)	✅	❌	❌
Create ClientViewer users	✅	❌	❌
Manage sites (optional)	✅	❌	❌
Devices: list/detail (MSP)	✅	✅	❌
Devices: create/update/delete	✅	❌	❌

Metrics: view charts	✅	✅	✅ (org-scoped)
Alert rules CRUD	✅	❌/⚠️ (if allowed by OrgAdmin)	❌
Incidents list/detail	✅	✅	✅ (org-scoped)
Incidents acknowledge/resolve	✅	✅	❌
Tickets CRUD	✅	✅	⚠️ (create/comment optional; org-scoped)
Ticket comments	✅	✅	⚠️ (own org; optional create/comment)
Request RemoteAssist session	✅	✅	❌
View RemoteAssist session (viewer)	✅	✅	❌
End session (technician stop)	✅	✅	❌
Approve session (end-user consent UI helper)	N/A (helper)	N/A (helper)	N/A (helper)
Notifications list/mark read	✅	✅	✅

Hard enforcement: ClientViewer is **never** allowed to run diagnostics or initiate sessions.

---

4. Core user flows (step-by-step)

4.1 MSP Console — onboarding and operations

1. **Login** with phone + password.
2. System loads **/me** to identify role and allowed navigation.
3. OrgAdmin selects an **Organization** context (or navigates from org list).
4. OrgAdmin creates users (Technician/ClientViewer) as needed.
5. OrgAdmin provisions agent onboarding code/org token and provides installer instructions.

4.2 Client portal — read-only monitoring + tickets (optional)

1. ClientViewer logs in.
2. Portal shows org-scoped KPIs and lists: devices/incidents/tickets.
3. ClientViewer opens a device to view last-seen and metric charts.
4. (Optional per recommendation) ClientViewer creates a ticket and/or posts comments.

4.3 Agent — install → register → steady-state

1. User runs manual installer (MSI/EXE).
2. On first launch, agent prompts for **onboarding code/org token**.
3. Agent calls **register** and receives a device token (AgentKey).
4. Agent enters steady-state loop:
    - Heartbeat every 60s (updates last_seen + minimal metrics)
    - Batch metrics upload every 5 minutes
    - Retry/backoff on failures
5. Agent rotates token per server instruction/policy.
6. Agent runs **allowlisted diagnostics only** when requested; logs audit events.

4.4 Remote session lifecycle (MVP view-only)

1. Technician requests session from device page (**request**).
2. Endpoint helper shows consent prompt: who requested, device, duration.
3. End-user approves (helper calls **approve**); session starts view-only stream.
4. Technician views stream and uses chat; may request allowlisted diagnostics.
5. End-user can **stop** anytime; technician can **end** session.
6. Auto-timeout ends session if duration expires; audit log captures lifecycle.

---

5. Functional requirements (FR-001…)

All FRs are atomic, testable, and non-overlapping. IDs are grouped by module.

5.1 MonitorHub (orgs/users/sites/devices, metrics, alerts/incidents, ticketing, notifications, audit)

FR-001 Organizations CRUD (MVP UI includes org list/detail)

- Acceptance: OrgAdmin can create/update/deactivate an Organization; inactive org data is not ingestible by agent and not operable in UI.

FR-002 Sites CRUD (optional)

- Acceptance: OrgAdmin can CRUD Sites within an org; devices may optionally reference a site.

FR-003 Users CRUD with roles

- Acceptance: OrgAdmin can create/update/disable users and assign role ∈ {OrgAdmin, Technician, ClientViewer}. Disabled users cannot authenticate.

FR-004 ClientViewer creation flow

- Acceptance: OrgAdmin can create a ClientViewer user linked to an organization; user can log in and only access org-scoped client endpoints.

FR-005 Devices CRUD + list filters

- Acceptance: OrgAdmin can CRUD devices; OrgAdmin/Technician can list devices filtered by status/site/last_seen (filtering rules defined in API section).

FR-006 Device “last seen” and status

- Acceptance: Heartbeats update device.last_seen_at; device status reflects recent heartbeat presence (definition in Data model).

FR-007 Metrics display

- Acceptance: Web UI can request and display device metric samples for the retention window.

FR-008 Alert rules CRUD

- Acceptance: OrgAdmin can CRUD alert rules per org, specifying metric/operator/threshold/duration/severity/is_active.

FR-009 Incident creation and resolution

- Acceptance: When an alert rule condition holds for required duration, an Incident opens; when condition clears, incident resolves. Severity is stored.

FR-010 Incident acknowledge (implied by UI task)

- Acceptance: OrgAdmin/Technician can acknowledge an open incident; acknowledged state is visible in lists/details.

FR-011 Ticket CRUD

- Acceptance: OrgAdmin/Technician can create tickets, assign to a user, and close; ticket can link to device and/or incident.

FR-012 Ticket comments

- Acceptance: Authorized users can add comments; comments show author and timestamp.

FR-013 Notifications (in-app only)

- Acceptance: System creates Notification records for key events (incidents opened/resolved, ticket updates, remote session lifecycle as applicable); users can list and mark read (read_at set).

FR-014 Audit logging

- Acceptance: System records AuditEvent for (a) diagnostics requests/results, (b) agent token issuance/rotation/revocation, (c) remote session lifecycle events.

5.2 Agent protocol (register, rotation, heartbeat, metrics batch, allowlisted diagnostics)

FR-020 Agent register

- Acceptance: Agent submits onboarding code/org token + device attributes; server creates/links Device and returns an AgentKey token for subsequent calls.

FR-021 Token rotation + revocation

- Acceptance: Server can revoke tokens; agent calls fail with auth error when revoked; rotation produces a new token and logs AuditEvent.

FR-022 Heartbeat

- Acceptance: Agent sends heartbeat every 60s; server updates device.last_seen_at and may accept minimal metrics.

FR-023 Metrics batch upload

- Acceptance: Agent uploads batches every 5 minutes; server validates and stores MetricSample records.

FR-024 Allowlisted diagnostics only

- Acceptance: Only predefined diagnostics commands are executable (e.g., ping, ipconfig, systeminfo, disk usage). Server rejects non-allowlisted requests and logs AuditEvent.

5.3 RemoteAssist (request/approve/end, consent UI, view-only session, chat, stop/timeout)

FR-030 Request session

- Acceptance: Technician requests session for a device; session is created in pending state with requester recorded.

FR-031 End-user consent required every time

- Acceptance: A session cannot become active unless helper approves; approval UI must show requester identity and duration.

FR-032 View-only stream (no input control)

- Acceptance: Session viewer provides read-only display; no keyboard/mouse control is exposed in MVP.

FR-033 Chat during session

- Acceptance: Technician can exchange chat messages during active session (persistence requirements defined in Data model/API).

FR-034 Stop/end/timeout

- Acceptance: End-user stop ends session; technician end ends session; timeout ends session automatically; each transition is audit-logged.

5.4 Client portal (scoped read-only views + optional ticket create/comment)

FR-040 Client-scoped device views

- Acceptance: ClientViewer can list/view only devices in their org, including last-seen and metric charts.

FR-041 Client-scoped incidents views

- Acceptance: ClientViewer can list/view only incidents in their org; read-only.

FR-042 Client-scoped tickets views

- Acceptance: ClientViewer can list/view only tickets in their org.

FR-043 (Recommended/optional) Client ticket create/comment

- Acceptance: If enabled, ClientViewer can create a ticket and add comments; cannot assign, close, or link arbitrary incidents/devices beyond their org.

---

6. Data model (entities/fields/relationships/constraints; statuses/state machines)

6.1 Entities (MVP)

(Names reflect the draft; fields abbreviated to MVP essentials.)

- **Organization**(id, name, city, industry, is_active)
- **Site**(id, organization_id FK, name, address_optional) *(optional)*
- **User**(id, phone UNIQUE, password_hash, role ENUM, organization_id FK?, is_active)
- **Device**(id, organization_id FK, site_id FK NULL, hostname, os, os_version, serial, ip, last_seen_at, status, agent_version)
- **AgentKey**(id, device_id FK, token_hash, issued_at, revoked_at NULL)
- **MetricSample**(id, device_id FK, ts, cpu_pct, ram_pct, disk_free_gb, uptime_sec)
- **AlertRule**(id, organization_id FK, metric ENUM, operator ENUM, threshold, duration_sec, severity ENUM, is_active)
- **Incident**(id, organization_id FK, device_id FK, rule_id FK, opened_at, resolved_at NULL, severity, status ENUM, last_event_at)
- **Ticket**(id, organization_id FK, device_id FK NULL, incident_id FK NULL, title, description, priority ENUM, assignee_user_id FK NULL, status ENUM, created_at, closed_at NULL)
- **TicketComment**(id, ticket_id FK, author_user_id FK, body, created_at)
- **RemoteSession**(id, organization_id FK, device_id FK, requester_user_id FK, status ENUM, started_at NULL, ended_at NULL, consent_method, audit_blob)
- **Notification**(id, user_id FK, type, payload_json, read_at NULL, created_at)
- **AuditEvent**(id, organization_id FK, actor_user_id FK NULL, actor_device_id FK NULL, type, metadata_json, created_at)

6.2 Constraints

- User.phone unique; user.role ∈ {OrgAdmin, Technician, ClientViewer}.
- Device belongs to exactly one Organization; optional Site.
- AgentKey tokens stored hashed; only one active token per device (recommended) — revoke old on rotate.
- MetricSample retention: raw samples kept **30 days**.

6.3 Status/state machines

Device.status (derived/managed):

- `ONLINE` if last_seen_at within heartbeat tolerance; else `OFFLINE`.

Incident.status (implied by “created/resolved” + UI “acknowledge/resolve”):

- `OPEN` → `ACKNOWLEDGED` → `RESOLVED`
- `RESOLVED` is set when rule condition clears; `resolved_at` recorded.

Ticket.status (create/assign/close):

- `OPEN` → `IN_PROGRESS` → `CLOSED`
- `assignee_user_id` optional; `closed_at` set on CLOSED.

RemoteSession.status (request/approve/end + stop/timeout):

- `PENDING_CONSENT` → `ACTIVE` → `ENDED`
- Terminal variants: `DECLINED`, `TIMED_OUT` (each sets ended_at and writes audit event)

---

7. API specification (REST /api/v1)

7.1 Auth/JWT

- JWT access/refresh (as per backend choice in draft context; logout may be a client-side token discard).
- `Authorization: Bearer <access>` for user endpoints.
- Agent endpoints use device token (AgentKey) in `Authorization: Bearer <agent_token>`.

7.2 Standard error shape

All errors return JSON:

```
{
  "error": {
    "code": "string",
    "message": "string",
    "details": { "field": ["msg"] }
  }
}
```

7.3 Pagination/filtering rules

- List endpoints support pagination (DRF-style recommended): `?page=1&page_size=20`
- Filtering via query params where applicable (devices/incidents/tickets): e.g. `?status=OPEN&device_id=...`

7.4 Endpoints (from draft; CRUD expands to standard REST verbs)

Auth / Identity

- `POST /api/v1/auth/login`
- `POST /api/v1/auth/logout`
- `GET /api/v1/me`

Org management (OrgAdmin unless noted)

- CRUD `/api/v1/org/users`
- CRUD `/api/v1/org/sites` *(optional)*
- CRUD `/api/v1/org/devices`
- CRUD `/api/v1/org/alert-rules`
- `GET /api/v1/org/incidents`
- CRUD `/api/v1/org/tickets`
- `POST /api/v1/org/tickets/{id}/comments`

Agent protocol

- `POST /api/v1/agent/register`
- `POST /api/v1/agent/heartbeat`
- `POST /api/v1/agent/metrics/batch`
- `POST /api/v1/agent/diagnostics/run`

RemoteAssist

- `POST /api/v1/org/remote-sessions/request`
- `POST /api/v1/remote-sessions/{id}/approve` *(called by helper on endpoint)*
- `POST /api/v1/remote-sessions/{id}/end`

Notifications

- `GET /api/v1/notifications`
- **Assumption (needed to set read_at):** `PATCH /api/v1/notifications/{id}` body `{ "read_at": "<iso>" }`

Client portal (ClientViewer scope)

- `GET /api/v1/client/devices`
- `GET /api/v1/client/incidents`
- `GET /api/v1/client/tickets`
- **Assumption (only if FR-043 enabled):** `POST /api/v1/client/tickets` and `POST /api/v1/client/tickets/{id}/comments`

---

8. Architecture overview

8.1 Backend (Django + DRF + Postgres + Redis + Celery + Nginx)

Modules/services (logical):

- **Auth/RBAC**: phone+password auth, JWT issuance, role permissions, org scoping.
- **Org & Users**: organizations, sites, user management.
- **Devices & Agent**: device CRUD, agent register/heartbeat/metrics/diagnostics.
- **Monitoring**: alert rules, incident engine, incident lifecycle.
- **Ticketing**: tickets, comments, workflows.
- **RemoteAssist**: session lifecycle endpoints + audit logging (streaming mechanism abstracted).
- **Notifications**: in-app notifications list/mark read.
- **Audit**: append-only audit events.

Celery jobs (from draft requirements):

- Metrics retention job (delete raw MetricSample older than 30 days).
- Incident engine job (evaluate alert rules; open/resolve incidents).
- (Optional) Remote session timeout enforcement job.

8.2 Flutter Web (DDD + Clean Architecture + feature-first)

Hard constraints:

- Feature-first structure
- DDD layers: domain/data/presentation
- auto_route routing + guards
- Dio networking
- get_it DI container
- **Assumption:** Bloc/Cubit state management

8.3 Agent (Flutter Windows tray app)

- Tray runtime, auto-start on boot.
- Collectors using win32 APIs (CPU/RAM/Disk/Uptime).
- Heartbeat loop + metrics batch loop.
- Retry/backoff on network failures.
- Diagnostics runner constrained to allowlist; outputs posted to API.
- Local log file for troubleshooting.

---

9. Non-functional requirements

- **Scale target (6 months):** 1–10 companies × 10–50 endpoints (≈100–500 endpoints).
- **Ingestion:** handle ~500 endpoints uploading 5-min batches (~12 uploads/hour/endpoint).
- **Retention:** keep raw metric samples **30 days**; longer aggregates post-MVP.
- **Security:** TLS everywhere, RBAC on all endpoints, rate limits, strong passwords, token rotation/revocation, audit logging.

- **Privacy:** explicit consent for every remote session; no stealth/unattended capability in MVP.
- **Observability:** health endpoints, structured logs; basic error tracking optional (Sentry optional).

---

10. Deployment & environments

- Target: **VPS (Ubuntu) + Docker Compose** with services:
  - `api` (Django/DRF)
  - `db` (PostgreSQL)
  - `redis`
  - `celery-worker`
  - `celery-beat`
  - `nginx` (TLS termination + reverse proxy)
- Secrets via environment variables (no plaintext in repo).
- Staging deploy notes: deploy compose stack, run migrations, create initial admin/org, verify health.

---

11. Testing strategy

- Backend: API tests, permission tests for MSP vs ClientViewer, agent auth tests, incident engine behavior tests, ticket workflow tests, remote session lifecycle tests.
- Agent protocol tests: request/response contract tests where feasible; integration tests using dev API environment.
- Frontend: smoke tests for auth + key pages; UX states (loading/empty/error/no-permission).
- QA checklist for MVP plus automated basics.

---

12. Task backlog (cleaned; keep existing Task IDs; coherent dependencies)

Task ID	Task	Depends on
S0-1	Repo + Docker Compose (api/db/redis/celery/nginx)	—
S0-2	CI lint/test pipeline (backend + flutter web)	S0-1
DS1-1	Design system + components	—

DS1-2	MSP Console screens designs	DS1-1
DS1-3	Client Portal screens designs	DS1-1
BE1-1	Auth (phone+password) + JWT + RBAC middleware	S0-1
BE1-2	Orgs + users CRUD + create ClientViewer	BE1-1
BE1-3	Device CRUD + filters + site optional	BE1-2
BE1-4	Agent register + token issuance/rotation	BE1-3
AG1-1	Agent skeleton + config + auto-start + logging	S0-1, BE1-4 (for integration)
AG1-2	Metrics collectors + heartbeat + batch upload + retry/backoff	AG1-1, BE2-1
BE2-1	Metrics ingestion + retention job	BE1-4
BE2-2	Alert rules + incident engine job	BE2-1
BE2-3	Ticketing CRUD + comments + workflow	BE1-2
BE2-4	Client portal endpoints (scoped queries)	BE2-3, BE2-2, BE1-3
BE3-1	Remote sessions lifecycle endpoints	BE1-3, BE1-1
AG2-1	RemoteAssist helper UI consent + approve + stop	AG1-1, BE3-1
FE1-1	Web shell + auth + role nav	S0-2, BE1-1
FE1-2	Dashboard + devices + charts	FE1-1, BE1-3, BE2-1
FE2-1	Incidents + alert rules UI	FE1-1, BE2-2
FE2-2	Tickets UI + comments	FE1-1, BE2-3
FE2-3	Client portal pages	FE1-1, BE2-4
FE3-1	Remote session viewer + stop	FE1-1, BE3-1
QA-1	Smoke + permission tests	BE/FE/AG core done
REL-1	Demo dataset + release checklist + operator runbook	QA-1

---

13. Risks, Assumptions, Open questions

Risks

- Consent correctness (must never start without explicit approval).
- Token security (agent tokens and rotations).
- Ingestion/retention job correctness at 500 endpoints.
- RBAC scoping bugs leaking org data.
- Remote session lifecycle edge cases (timeout/stop).

Assumptions (minimal)

1. **MSP users vs org scoping:** `User.organization_id` is **nullable** for MSP roles (OrgAdmin/Technician) so they can operate across orgs; **ClientViewer** always has `organization_id`. (Draft contains org list/detail screens but user model ties to an org.)
2. Notifications “mark read” uses a `PATCH /notifications/{id}` endpoint (needed to set `read_at`).
3. Client ticket create/comment is **enabled** in MVP because FE2-3 DoD implies “Client can submit/view tickets,” despite being labeled “optional/recommended” elsewhere.

Open questions

- Onboarding code/org token: format, expiration, and whether it maps to org only or org+site.
- Invite flow for ClientViewer: how do they receive credentials (no email/SMS in MVP)?
- Remote stream technology and chat transport (kept abstract in draft).
- Exact incident “acknowledge” semantics (does it affect auto-resolve?).
- Rate limit thresholds (per-user/per-agent) and heartbeat tolerance for ONLINE/OFFLINE.

---

2) BACKEND PROMPT PACK (optimized for Claude / DeepSeek)

Execute prompts in order. Each prompt is copy-paste ready.

2.1 S0-1 — Repo structure + Docker Compose (api/db/redis/celery/nginx)

Objective: Create repo skeleton and production-like Docker Compose for VPS Ubuntu.
Inputs: FR-013/014 (audit/notifications infra), NFR deployment stack; Entities: all core.
Files/folders:

- `/docker-compose.yml`

- `/nginx/` (conf)
- `/backend/` (Django project)
- `/.env.example`
- `/README.md`
  Implementation steps:

1. Initialize repo with `/backend`, `/nginx`, `/scripts`, `/docs`.
2. Create Django project `it_suite` + apps placeholders: `accounts`, `orgs`, `devices`, `monitoring`, `tickets`, `remoteassist`, `notifications`, `audit`.
3. Write `docker-compose.yml` services: `api`, `db`, `redis`, `celery-worker`, `celery-beat`, `nginx`.
4. Add env var wiring (`DATABASE_URL` or discrete vars), Redis, Django secret key.
5. Nginx reverse proxy to `api:8000`, serve health endpoint.
6. Add staging notes to `/docs/deploy_staging.md`.
   Verification:

docker compose up -d --build
docker compose ps
docker compose logs -n 50 api

Expected: all containers healthy/running; API container starts without crash.
DoD checklist:

- `docker compose up` succeeds
- Nginx proxies to API container
- `/docs/deploy_staging.md` exists with minimal steps

2.2 S0-2 — CI lint/test pipeline (backend + flutter web)

Objective: Add CI pipeline that runs backend lint + tests (and placeholder for flutter web).
Inputs: Testing strategy section; Tasks S0-2.
Files/folders: `/.github/workflows/ci.yml` (or equivalent), `/backend/pyproject.toml` or `requirements.txt` updates.
Implementation steps:

1. Add backend lint tooling (e.g., format + lint) and pytest.
2. CI job: checkout → set up Python → install deps → run lint → run tests.
3. Add a second job placeholder for Flutter Web (runs `flutter test` and/or `flutter build web`) but keep minimal until FE exists.
   Verification:

python -m pytest -q

Expected: pipeline config validates; local tests runnable (even if empty baseline).
DoD:

- CI runs on PR
- Backend lint + tests steps exist
- Flutter job placeholder exists

2.3 BE1-1 — Auth (phone+password) + JWT + RBAC middleware/permissions

Objective: Implement phone-password auth, JWT login, `/me`, and role-based permissions.
Inputs: FR-003; Endpoints: `/auth/login`, `/auth/logout`, `/me`; Entities: User.
Files/folders:

- `/backend/accounts/models.py`
- `/backend/accounts/auth.py` (or permissions)
- `/backend/accounts/api/` (views/serializers/urls)
- `/backend/it_suite/settings.py`
  Implementation steps:

1. Create custom `User` model with `phone` as unique identifier, `role`, `organization_id` nullable (ASSUMPTION), `is_active`.
2. Configure JWT auth for DRF; implement:
   - `POST /api/v1/auth/login` (phone+password → tokens)
   - `POST /api/v1/auth/logout` (document as no-op/token discard unless you add blacklist)
   - `GET /api/v1/me` returns user profile + role + org
3. Implement permission classes:
   - `IsOrgAdmin`, `IsTechnician`, `IsClientViewer`
   - `OrgScopedQueryMixin` to enforce org scoping for ClientViewer and org-context endpoints.
4. Add tests for:
   - login success/fail
   - `/me` requires auth
   - role checks
     Verification:

docker compose exec api python manage.py makemigrations
docker compose exec api python manage.py migrate

docker compose exec api python -m pytest -q

Expected: migrations apply; tests pass.
DoD:

- Can login with phone/password
- `/me` returns role/org
- Role permission tests pass

2.4 BE1-2 — Organizations + users CRUD + create ClientViewer

Objective: Org + user management endpoints for OrgAdmin including ClientViewer creation.
Inputs: FR-001/003/004; Endpoints: CRUD `/org/users`; Entities: Organization, User.
Files/folders:

- `/backend/orgs/models.py`, `/backend/orgs/api/...`
- `/backend/accounts/api/...`
  Implementation steps:

1. Implement `Organization` model and (optional) basic list/detail endpoints for OrgAdmin (needed by MSP org list screen).
2. Implement `/api/v1/org/users` CRUD (OrgAdmin only).
3. Add “create ClientViewer” path: create user with role `ClientViewer` and `organization_id` set.
4. Add permission/scoping tests: ClientViewer cannot list org users; Technician cannot manage users.
   Verification:

docker compose exec api python -m pytest -q

Expected: org/user tests pass.
DoD:

- OrgAdmin manages users
- ClientViewer creation works and is org-scoped
- Permission tests pass

2.5 BE1-3 — Device CRUD + list filters + site optional

Objective: Devices inventory with optional site and list filtering.
Inputs: FR-002/005/006; Endpoints: CRUD `/org/sites`, CRUD `/org/devices`; Entities: Site, Device.
Files/folders: `/backend/devices/models.py`, `/backend/devices/api/...`, `/backend/orgs/models.py`
Implementation steps:

1. Implement `Site` (optional) and CRUD `/api/v1/org/sites` (OrgAdmin).
2. Implement `Device` model with fields from spec; CRUD `/api/v1/org/devices`.
3. Add filters: status, site_id, last_seen range (minimal query params).
4. Add computed/managed `status` from `last_seen_at` tolerance (server-side).
5. Tests: list filters + role access.
   Verification:

docker compose exec api python -m pytest -q

DoD:

- Devices CRUD works
- Filters work
- Status updates derived from last_seen rules

2.6 BE1-4 — Agent register + token issuance/rotation

Objective: Agent registration with onboarding code/org token, plus AgentKey issuance/rotation/revocation logging.
Inputs: FR-020/021; Endpoints: `/agent/register`; Entities: Device, AgentKey, AuditEvent.
Files/folders: `/backend/devices/agent_api/...`, `/backend/audit/...`
Implementation steps:

1. Implement onboarding token validation (minimal placeholder if format unknown; track as Open Question).
2. `POST /api/v1/agent/register`: create/update device record, issue AgentKey, return token once.
3. Store token hashed, support rotate (server-triggered policy; minimal: rotate on register, later add rotate flag response).
4. Emit AuditEvent for token issuance/rotation/revocation.
5. Tests: agent register success; revoked token fails.
   Verification:

docker compose exec api python -m pytest -q

DoD:

- Agent registers and receives token
- Token stored hashed
- Audit events created for token actions

2.7 BE2-1 — Metrics ingestion + retention Celery job

Objective: Heartbeat and metrics batch ingest + 30-day pruning job.
Inputs: FR-022/023; Endpoints: `/agent/heartbeat`, `/agent/metrics/batch`; Entities: MetricSample, Device.
Files/folders: `/backend/monitoring/models.py`, `/backend/monitoring/tasks.py`
Implementation steps:

1. Implement `POST /agent/heartbeat`: auth via agent token; update `device.last_seen_at`; accept minimal metrics if provided.
2. Implement `POST /agent/metrics/batch`: validate payload list, create MetricSample rows efficiently.
3. Implement Celery beat task: delete MetricSample older than 30 days.
4. Tests: ingest stores data; retention prunes.
   Verification:

docker compose exec api python -m pytest -q
docker compose exec api celery -A it_suite inspect ping

Expected: celery responds; tests pass.
DoD:

- Metrics visible in DB after ingest
- Retention task prunes old samples
- Heartbeat updates last_seen_at

2.8 BE2-2 — Alert rules + incident engine Celery job

Objective: CRUD alert rules + periodic evaluation producing Incidents and resolving them.
Inputs: FR-008/009/010; Endpoints: CRUD `/org/alert-rules`, `GET /org/incidents`; Entities: AlertRule, Incident, MetricSample.
Files/folders: `/backend/monitoring/api/...`, `/backend/monitoring/engine.py`,

`/backend/monitoring/tasks.py`
Implementation steps:

1. Implement `AlertRule` model/enums (metric/operator/severity).
2. CRUD endpoints for OrgAdmin.
3. Implement incident engine job:
    - Evaluate recent metrics within duration window
    - Open incident if threshold breach sustained
    - Resolve when condition clears
4. Add incident acknowledge endpoint or minimal PATCH on incident (if implied) and tests.
   Verification:

docker compose exec api python -m pytest -q

DoD:

- Alert rules CRUD works
- Incidents open/resolve via Celery job
- Acknowledge works for MSP roles

2.9 BE2-3 — Ticketing CRUD + comments + status workflow

Objective: Tickets + comments with status transitions and linking device/incident.
Inputs: FR-011/012; Endpoints: CRUD `/org/tickets`, `POST /org/tickets/{id}/comments`; Entities: Ticket, TicketComment.
Files/folders: `/backend/tickets/models.py`, `/backend/tickets/api/...`
Implementation steps:

1. Implement Ticket + TicketComment models and serializers.
2. CRUD endpoints with role permissions (OrgAdmin/Technician).
3. Implement comment create endpoint.
4. Enforce workflow transitions: OPEN → IN_PROGRESS → CLOSED.
5. Tests: create ticket, assign, comment, close; permission boundaries.
   Verification:

docker compose exec api python -m pytest -q

DoD:

- Full ticket workflow works
- Comments recorded with author/timestamp

- Role scoping enforced

2.10 BE2-4 — Client portal endpoints (scoped queries + permissions)

Objective: Implement `/client/*` endpoints with strict org scoping for ClientViewer.
Inputs: FR-040/041/042/043; Endpoints: `/client/devices`, `/client/incidents`, `/client/tickets`; Entities: Device/Incident/Ticket.
Files/folders: `/backend/client_portal/api/...`
Implementation steps:

1. Create client portal views returning org-scoped data only.
2. Ensure ClientViewer cannot access `/org/*` endpoints.
3. If enabling FR-043 (ASSUMPTION), add:
    - `POST /client/tickets`
    - `POST /client/tickets/{id}/comments`
4. Tests: ClientViewer sees only their org; forbidden elsewhere.
   Verification:

docker compose exec api python -m pytest -q

DoD:

- ClientViewer endpoints return scoped data only
- No cross-org leakage in tests

2.11 BE3-1 — Remote sessions lifecycle endpoints + state machine + audit logging

Objective: Request/approve/end session endpoints with strict consent enforcement and audit logs.
Inputs: FR-030..034; Endpoints: `/org/remote-sessions/request`, `/remote-sessions/{id}/approve`, `/remote-sessions/{id}/end`; Entities: RemoteSession, AuditEvent, Notification.
Files/folders: `/backend/remoteassist/models.py`, `/backend/remoteassist/api/...`, `/backend/audit/...`, `/backend/notifications/...`
Implementation steps:

1. Implement RemoteSession model + status enum.

2. Implement `request`: only MSP roles; create `PENDING_CONSENT`.
3. Implement `approve`: called by endpoint helper (auth method is an Open Question; implement minimal token-in-body approach and document). Transition to `ACTIVE`, set started_at.
4. Implement `end`: allowed for technician or helper stop; transition to ENDED and set ended_at.
5. Implement timeout enforcement (Celery beat) if duration exceeded.
6. Emit AuditEvent for request/approve/end/timeout; create in-app Notifications for requester and relevant users.
7. Tests: cannot activate without approve; stop/timeout behavior; audit events exist.
   Verification:

docker compose exec api python -m pytest -q

DoD:

- Consent required to reach ACTIVE
- View-only enforced at API level (no “control” endpoints)
- Audit + notifications created for lifecycle events

2.12 Notifications endpoints + read_at

Objective: List notifications and mark as read.
Inputs: FR-013; Endpoint: `GET /notifications`, (ASSUMPTION) `PATCH /notifications/{id}`; Entity: Notification.
Files/folders: `/backend/notifications/api/...`
Implementation steps:

1. Implement list endpoint with pagination.
2. Implement mark-read update that sets `read_at`.
3. Tests: user sees own notifications only; read_at updated.
   Verification:

docker compose exec api python -m pytest -q

DoD:

- Notifications list works
- Mark read updates read_at
- Scoped to authenticated user

2.13 AuditEvent creation for commands/token/session events

Objective: Centralize audit creation helpers and ensure coverage.
Inputs: FR-014/021/024/034; Entities: AuditEvent.
Files/folders: `/backend/audit/service.py`
Implementation steps:

1. Add helper `audit_log(org, actor_user, actor_device, type, metadata)` used by agent/remote/tickets/monitoring where required by spec.
2. Ensure diagnostics run, token actions, and session lifecycle call audit helper.
3. Tests: audit event count/assertions for key flows.
   Verification:

docker compose exec api python -m pytest -q

DoD:

- Audit events exist for required flows
- Tests assert audit creation

2.14 QA/REL — Operator runbook basics + demo seed script (REL-1)

Objective: Provide a repeatable demo dataset and operator runbook basics.
Inputs: REL-1; Entities: Organization/User/Device/Metrics/AlertRule/Incident/Ticket/Notification.
Files/folders: `/scripts/seed_demo.py`, `/docs/runbook.md`
Implementation steps:

1. Create demo seed script creating:
   - 2 orgs, MSP admin + tech, 1 clientviewer per org
   - 5 devices per org with metric samples
   - 2 alert rules and at least 1 incident
   - 2 tickets with comments
2. Runbook: how to deploy, migrate, seed, smoke test endpoints.
   Verification:

docker compose exec api python /scripts/seed_demo.py
docker compose exec api python -m pytest -q

DoD:

- Seed script runs idempotently (or documented reset)
- Runbook contains deploy + smoke steps

---

3) FLUTTER WEB PROMPT PACK (optimized for Gemini; DDD + Clean Architecture + auto_route + Dio + get_it)

Architecture requirements (must follow)

- **Feature-first structure**
- For each feature:
  - `domain/` (entities, repository interfaces, use cases)
  - `data/` (DTOs, mappers, API client calls, repository impl)
  - `presentation/` (pages/widgets/state)
- **Routing:** auto_route + route guards for role-based navigation
- **Networking:** Dio + auth interceptors + error mapping to domain failures
- **DI:** get_it container + explicit module registration
- **ASSUMPTION (state):** Bloc/Cubit

Prompts follow Task IDs order.

3.1 FE1-1 — Web shell + auth + role-based navigation

Goal: Flutter Web app scaffold with login, JWT storage, guards, MSP vs Client portal mode navigation.
Req IDs + endpoints: FR-003/040; `/auth/login`, `/me`, `/notifications` (later).
Folder/file changes:

- `/web_app/lib/app/` (router, di, theme, error mapping)
- `/web_app/lib/features/auth/...`
- `/web_app/lib/features/shell/...`

Implementation steps:

1. Create Flutter Web project in `/web_app`.
2. Set up get_it container and register:
   - Dio client with auth interceptor
   - Auth repository + use cases
   - Cubits/Blocs
3. Configure auto_route:
   - Public routes: Login
   - Protected shell routes: MSP console vs Client portal

    - Guard reads `/me` and role to route accordingly
4. Implement Auth feature:
    - domain: `UserSession`, `AuthRepository`
    - data: DTOs for login/me; token storage
    - presentation: Login page + auth cubit
5. UX states: loading/error/no-permission for guarded routes.
   Verification:

flutter pub get
flutter run -d chrome
flutter build web

Expected UI: login works; redirects to MSP or Client portal based on role; unauthorized shows login.
DoD:

- auto_route guard enforces auth and role
- Dio attaches JWT
- Error states rendered

3.2 FE1-2 — MSP Dashboard + devices list/detail + metrics charts

Goal: MSP devices views and basic KPI dashboard, metrics chart rendering.
Req + endpoints: FR-005/007; CRUD `/org/devices`, (metrics endpoint implied via API design for MetricSample retrieval).
Folder/file changes:

- `/web_app/lib/features/devices/...`
- `/web_app/lib/features/dashboard/...`
  Implementation steps:
1. Devices feature: repository + use cases for list/detail; filters; loading/empty/error states.
2. Device detail: metrics chart widget (read-only) fed by metric samples endpoint (implement API contract matching backend).
3. Dashboard: show simple KPI tiles (counts of devices ONLINE/OFFLINE, open incidents, open tickets) using existing endpoints as available.
   Verification:

flutter run -d chrome

Expected: devices list renders; device detail shows last seen + charts; handles empty dataset.
DoD:

- Devices list/detail works for MSP roles
- Charts render from API data
- UX states present

3.3 FE2-1 — Incidents list/detail + alert rules UI (+ acknowledge/resolve)

Goal: Incidents and alert rules management UI.
Req + endpoints: FR-008/009/010; CRUD `/org/alert-rules`, `GET /org/incidents`.
Folder/file changes: `/web_app/lib/features/incidents/...`, `/web_app/lib/features/alert_rules/...`
Implementation steps:

1. Alert rules: CRUD screens with validation.
2. Incidents: list/detail views; actions for acknowledge/resolve if backend supports.
3. Enforce permissions: ClientViewer routes hidden/blocked.
   Verification:

flutter run -d chrome

Expected: alert rules CRUD works; incidents list/detail loads; action buttons show only for MSP roles.
DoD:

- Incidents and alert rules pages function end-to-end
- Role-based access enforced
- UX states complete

3.4 FE2-2 — Tickets board/list + ticket detail + comments

Goal: Ticket workflow UI with comments.
Req + endpoints: FR-011/012; CRUD `/org/tickets`, `POST /org/tickets/{id}/comments`.
Folder/file changes: `/web_app/lib/features/tickets/...`
Implementation steps:

1. Tickets list/board (minimal columns/status buckets).
2. Ticket detail: fields + status transitions; comment composer + list.
3. Error mapping (permission denied vs validation).
   Verification:

flutter run -d chrome

Expected: tickets render; comment posting updates list; status transitions reflect in UI.
DoD:

- Tickets workflow supported
- Comments supported
- States: loading/empty/error/no-permission

3.5 FE2-3 — Client portal pages (devices/incidents/tickets + optional create/comment)

Goal: ClientViewer portal mode pages with strict read-only behavior, plus ticket create/comment if enabled.
Req + endpoints: FR-040..043; `/client/devices`, `/client/incidents`, `/client/tickets` (+ optional create/comment endpoints).
Folder/file changes: `/web_app/lib/features/client_portal/...`
Implementation steps:

1. Create client portal shell routes and pages mirroring MSP read-only variants.
2. Use client endpoints only; no `/org/*` calls.
3. If backend enables client ticket create/comment, add UI; otherwise hide with clear “not available” messaging.
   Verification:

flutter run -d chrome

Expected: ClientViewer sees only client pages; cannot navigate to MSP routes; ticket creation behaves per backend availability.
DoD:

- Client portal uses only client endpoints
- No-permission state for MSP pages
- Optional ticket create/comment gated

3.6 FE3-1 — Remote session viewer (view-only) + stop session

Goal: View-only remote session viewer page and stop session action for MSP roles.
Req + endpoints: FR-030..034; `/org/remote-sessions/request`,

`/remote-sessions/{id}/end`.
Folder/file changes: `/web_app/lib/features/remoteassist/...`
Implementation steps:

1. Device page action: request remote session and show “awaiting consent” state.
2. Session viewer UI: view-only container (stream rendering abstracted; integrate once backend defines stream URL/transport).
3. Stop session button triggers end endpoint; display audit state transitions.
   Verification:

flutter run -d chrome

Expected: request shows pending; stop ends session; UI reflects status changes.
DoD:

- View-only enforced in UI (no control inputs)
- Stop session works
- Status states handled (pending/active/ended/timeout)

3.7 Notifications center + read/unread

Goal: Notifications list with read/unread and mark-read action.
Req + endpoints: FR-013; `GET /notifications`, `PATCH /notifications/{id}` (assumption).
Folder/file changes: `/web_app/lib/features/notifications/...`
Implementation steps:

1. Notifications feature with repository + paging.
2. UI for unread badge and mark-as-read.
3. Show empty/error states.
   Verification:

flutter run -d chrome

Expected: notifications load, unread marked read updates UI.
DoD:

- Read/unread works
- Correct scoping to logged-in user

---

4) AGENT + REMOTEASSIST HELPER PROMPT PACK (optimized for Gemini; Flutter Windows Desktop)

4.1 AG1-1 — Flutter Windows tray agent skeleton + config + auto-start + logging

Goal: Windows tray app skeleton that runs on boot, configurable API URL, logs to file.
Protocol/endpoints: `/agent/register`, `/agent/heartbeat`, `/agent/metrics/batch` (later).
Files/folders:

- `/agent_app/` Flutter Windows project
- `/agent_app/lib/config/...`
- `/agent_app/lib/logging/...`
- `/agent_app/windows/...` (startup integration)

Implementation steps:

1. Create Flutter Windows desktop project in `/agent_app`.
2. Implement tray icon + background/minimized runtime.
3. Config: API base URL, onboarding code storage (secure storage).
4. Auto-start: Startup folder or Task Scheduler entry (as per draft note).
5. Logging: append logs to local file with rotation (simple size-based).

Verification (Windows):

- Run app manually, confirm tray icon, confirm log file created.
- Reboot (or simulate auto-start) and confirm app starts.

DoD:

- Tray app runs minimized
- API URL configurable
- Logs written to file
- Auto-start configured

4.2 AG1-2 — Metrics collectors + heartbeat + batch upload + retry/backoff

Goal: Collect CPU/RAM/Disk/Uptime via win32, send heartbeat every 60s and batch metrics every 5 minutes with retry/backoff.
Protocol/endpoints: `/agent/heartbeat`, `/agent/metrics/batch`
Files/folders: `/agent_app/lib/metrics/...`, `/agent_app/lib/networking/...`,

`/agent_app/lib/scheduler/...`
Implementation steps:

1. Implement collectors using win32 APIs: cpu%, ram%, disk free GB, uptime sec.
2. Implement networking client that includes agent token in Authorization header.
3. Heartbeat loop (60s): updates last_seen + minimal metrics.
4. Batch loop (5m): send array of MetricSamples; persist queue locally if offline.
5. Retry/backoff strategy and visible log traces.
   Verification (Windows):

- Run agent; verify periodic log entries.
- Confirm backend receives heartbeat/batch (check API logs / device last_seen changes).
  DoD:
- Collectors return valid values on Win10/11
- Heartbeat and batch calls occur on schedule
- Offline retry/backoff works without crashing

4.3 Diagnostics allowlist runner + audit events

Goal: Execute only allowlisted diagnostics and post results; no arbitrary shell.
Protocol/endpoints: `/agent/diagnostics/run`
Files/folders: `/agent_app/lib/diagnostics/...`
Implementation steps:

1. Hardcode allowlist commands aligned with draft examples: ping, ipconfig, systeminfo, disk usage.
2. Implement request polling or push trigger handling (depending on backend contract); execute requested allowlisted command only.
3. Post stdout/stderr + exit code + timestamps; log locally.
   Verification (Windows):

- Trigger a diagnostics request from backend; verify execution and posted result.
  DoD:
- Non-allowlisted commands rejected locally
- Results posted to API
- Logs show execution trail

4.4 AG2-1 — RemoteAssist helper UI: consent prompt + approve session token + session status + stop

Goal: Endpoint helper that displays consent prompt, approves session, shows status, and allows stop.

Protocol/endpoints: `/remote-sessions/{id}/approve`, `/remote-sessions/{id}/end`
Files/folders: `/agent_app/lib/remoteassist_helper/...`
Implementation steps:

1. Implement UI modal/popover on incoming session request showing requester and duration.
2. Approve button calls approve endpoint; show ACTIVE status.
3. Stop button calls end endpoint; show ENDED/TIMED_OUT accordingly.
4. Ensure helper cannot start without explicit user click.
   Verification (Windows):

- Request session from MSP console → helper prompt appears → approve → status changes → stop works.
  DoD:
- Consent required to approve
- Stop ends session reliably
- Session status is visible and logged

---

5) FINAL PROJECT MANAGEMENT PACKAGE

Milestones mapped to Task IDs + FR IDs

1. **Foundation Ready**

- Tasks: S0-1, S0-2
- FRs: enables all

2. **Secure Core + Inventory**

- Tasks: BE1-1, BE1-2, BE1-3, FE1-1
- FRs: FR-001..006

3. **Monitoring Pipeline**

- Tasks: BE1-4, BE2-1, BE2-2, AG1-1, AG1-2, FE1-2, FE2-1
- FRs: FR-020..023, FR-007..010

4. **Ticketing + Client Portal**

- Tasks: BE2-3, BE2-4, FE2-2, FE2-3
- FRs: FR-011..012, FR-040..043

5. **RemoteAssist MVP**

- Tasks: BE3-1, AG2-1, FE3-1
- FRs: FR-030..034

6. **Release Readiness**

- Tasks: QA-1, REL-1
- FRs: all acceptance criteria verified

Critical path dependencies (backend vs web vs agent)

- **Backend-first blockers:** BE1-1 → BE1-2 → BE1-3 → BE1-4 → BE2-1 → BE2-2; BE2-3; BE3-1
- **Agent integration blockers:** AG1-2 depends on BE2-1; AG2-1 depends on BE3-1
- **Frontend integration blockers:** FE1-1 depends on BE1-1; FE1-2 depends on BE1-3/BE2-1; FE2-1 depends on BE2-2; FE3-1 depends on BE3-1

Definition of Done checklist for MVP release

- **Locked constraints**
  - Windows 10/11 only agent support validated
  - RemoteAssist is view-only (no input control anywhere)
  - Explicit consent required per session; no unattended sessions
  - Notifications are in-app only
- **Security/RBAC**
  - Role permissions enforced on every endpoint
  - ClientViewer cannot access MSP endpoints; org scoping tests pass
  - Agent token hashing + revocation/rotation works
- **Core features**
  - Devices inventory + last-seen + status
  - Metrics ingestion + charts
  - Alert rules + incidents open/ack/resolve
  - Ticketing + comments + workflow
  - Remote session lifecycle request/approve/end + timeout
  - Audit events for token/diagnostics/session events
  - Notifications list + mark read
- **Ops**
  - Docker Compose deploy works on Ubuntu VPS
  - Runbook + seed demo data available
  - QA-1 smoke checklist completed

Risk register + mitigations

- **Consent correctness risk:** session becomes ACTIVE without explicit approval
  - Mitigation: backend state machine forbids ACTIVE without approve; tests for forbidden transitions; helper UI requires click.
- **Token security risk:** agent token leakage or reuse

    - Mitigation: store hashed tokens; rotate; revoke on suspicion; rate limit agent endpoints.
- **Ingestion scale risk:** DB growth and slow queries at 30-day retention
    - Mitigation: retention job mandatory; index on (device_id, ts); batch inserts.
- **RBAC scoping risk:** data leakage across organizations
    - Mitigation: centralized org-scoping filters; exhaustive permission tests (QA-1).
- **Remote session lifecycle edge cases:** stop/timeout race conditions
    - Mitigation: idempotent end endpoint; terminal states guarded; audit every transition.

Demo dataset + scripted demo scenario (REL-1 aligned)

Demo dataset (seed script):

- 2 Organizations (OrgA, OrgB)
- MSP OrgAdmin + Technician
- ClientViewer for each org
- 5 devices/org with metric samples over last hour
- 2 alert rules; at least 1 open incident
- 2 tickets with comments and one linked to an incident
- 3 notifications (incident opened, ticket updated, session requested)

Scripted demo (10–12 minutes):

1. OrgAdmin logs in → selects OrgA → shows devices list/detail with charts.
2. Show alert rules → show incident list → acknowledge incident.
3. Technician opens ticket board → comments → moves ticket status.
4. Start RemoteAssist request from a device → show “pending consent”.
5. On endpoint, helper prompts → approve → web viewer shows ACTIVE (view-only) → chat → stop session.
6. ClientViewer logs in for OrgA → sees devices/incidents/tickets (and creates ticket if enabled).
7. Notifications center: mark items read; show audit events existence via admin tooling/logs.
