Backend Technical Guide

This guide describes how the FastAPI backend is structured, how modules/workflows are executed, and how to extend or operate the service safely.

---

1. Stack Overview

Item	Details
Framework	FastAPI + Uvicorn
Language	Python 3.11
Entry point	`simple-server.py` (dev) / PyInstaller bundle (prod)
Modules engine	`modules/telco_modules.py` (ADB commands)
Storage	In-memory + local JSON caches today (future DB planned)
Packaging	PyInstaller for the Electron bundle; also runnable as standalone API

---

2. Project Structure (Backend)

```
mon-projet/
  src/
      backend/
          api/
              devices.py / devices_v2.py
              modules.py
              workflows.py
              dashboard.py
          modules/
              telco_modules.py          # Core ADB routines
              device_actions.py         # Shared helpers
          services/
              task_queue.py (future scheduling hook)
              device_manager.py       # Device discovery/state
          simple-server.py             # FastAPI app + wiring
```

Key dependencies: `fastapi`, `uvicorn`, `pydantic>=2`, `adbutils` (invoked via subprocess), `aiofiles` (for future streaming).

---

3. API Surface

3.1 Devices

- `GET /api/v1/devices`: enumerate currently detected ADB devices (status, metadata).
- `POST /api/v1/devices/{device_id}/disconnect|reboot|logs`: trigger device actions.
- WebSocket `/ws/devices`: pushes real-time status updates to the frontend (managed via `resolveBaseUrl` + `websocket.ts`).

3.2 Modules

- `GET /api/modules`: metadata catalog (id, description, parameters).
- `POST /api/modules/{id}/execute`: execute a module for a specific `device_id`. Body includes `parameters`.
  - For telco modules, `modules/telco_modules.py` executes the corresponding ADB routine.
  - Ping, airplane mode, SMS, etc. run synchronously; errors bubbled via `success`, `error`.
  - Voice Call test now runs synchronously (removed background task) so workflows respect sequential execution.

3.3 Workflows

- `POST /api/workflows/run`: triggered from frontend workflow runner.
- Workflows themselves are orchestrated client-side for now (looping over modules and calling `/api/modules/...`).
- Future plan: move orchestration server-side (queue + workers) for the Scheduler.

3.4 Dashboard / Reports

- `/api/v1/dashboard/summary`, `/api/v1/dashboard/activity/recent`: aggregated stats for the dashboard Panels.
- `/api/v1/search`: global search across modules/workflows/devices (backend fallback when local search empty).

---

4. Module Execution Pipeline

1. FastAPI endpoint validates `device_id`, parameters (Pydantic models with `@field_validator` on `duration`, etc.).
2. Instantiate `TelcoModules(device_id)` (wraps ADB CLI calls).
3. Execute the requested action (`voice_call_test`, `ping_target`, `enable_airplane_mode…`).

4. Return structured response: `{ success: bool, result: {...}, error?: str }`.
5. Frontend renders success/failure and logs activity in Device Manager history.

Adding a New Module

1. Implement method inside `telco_modules.py`.
2. Extend `modules_db` catalog in `api/modules.py`.
3. Wire new `module_id` inside the POST handler (parameter validation + executor call).
4. Update frontend `MODULE_CATALOG` with metadata so it shows up in Test Modules/workflows.

---

5. Workflow Orchestration

Although the loop lives in `FlowComposer.tsx`, the backend exposes hooks that enable future server-side orchestration:

- Abort support: each module request can be cancelled via `AbortController`; backend handles `asyncio.CancelledError`.
- Pause/Resume: tracked client-side via `workflowPauseRef`, but backend modules are synchronous—once a call is in progress, pause waits for the HTTP request to complete.
- Scheduler integration (planned):
  - Store definitions (`workflow_id`, cron expression, device group).
  - Worker reads due jobs, calls the same module execution API, and records run history.

---

6. Telemetry & Logging

- Backend logs to stdout; Electron main process prefixes `[electron]` / `Backend Error` lines.
- Use `logger` in modules API to capture success/failure.
- Device Manager history relies on frontend local storage today; when backend persistence is introduced, add `/api/devices/{id}/activity` endpoint.

---

7. Versioning & Migration

- Maintain `CHANGELOG.md` (semantic versioning, e.g., `1.3.0`).
- Pydantic v1 decorators (`@validator`) must be migrated to `@field_validator`/`@model_validator`.

- Scheduler + notifications + live logs will require a persistent store (SQLite/Postgres). Plan migrations using Alembic once DB layer added.

---

8. Ops Checklist

1. `python simple-server.py` (dev) or run packaged backend on port `8007`.
2. Ensure ADB is installed and `adb devices` lists the hardware.
3. Electron launches backend automatically in production; check logs if port occupied.
4. For scheduler/notifications prototypes, run auxiliary workers separately to avoid blocking FastAPI event loop.

Last updated: 2025-12-12