Frontend Technical Guide

This guide documents the React/Electron frontend so contributors can navigate the codebase confidently, extend major pages, and understand build/deployment workflows.

---

1. Stack Summary

Item	Details
Framework	React 18 (CRA) + TypeScript
UI Toolkit	MUI (primary) + Chakra theme tokens for global styling
State	React hooks (`useState`, `useReducer`), custom hooks (`useDevices`), localStorage caches
Routing	`react-router-dom` (`BrowserRouter` for web, `HashRouter` when running inside Electron)
Bundling	`react-scripts build` for production assets (copied into `src/frontend/build`)
Desktop shell	Electron (renderer loads `build/index.html`, main process handles backend lifecycle)

---

2. Project Structure (Frontend)

```
mon-projet/src/frontend/src/
  App.tsx                    # Router + theme providers
  pages/
     Dashboard.tsx           # Device overview, activity, alias management
     DeviceManager.tsx       # Historical registry, analytics, bulk actions, timelines
     FlowComposer.tsx        # Workflow list, builder, runner (pause/stop)
     TestModules.tsx         # Module catalog, drag/drop workflow builder, ping config
     Reports.tsx (lazy)      # Placeholder for future reporting
  components/
     Layout.tsx              # App chrome (sidebar + top bar)
     Dashboard/...           # (legacy components)
     CallTestDialog.tsx      # Multi-step call test modal
  hooks/
     useDevices.ts           # Device polling + WebSocket integration
  services/
     deviceApi.ts / dashboardApi.ts
     websocket.ts            # Broadcast updates
```

```
utils/
    workflows.ts           # Stored workflows, repeat settings
    deviceSelection.ts     # localStorage for selected devices
    deviceMetadata.ts      # Alias/tags/notes/checklist utilities
    deviceRegistry.ts      # Historical device log
    deviceActivity.ts      # Module/workflow run history per device
styles/
    dashboardTheme.ts      # Design tokens (colors, radius, shadows)
    Dashboard.css          # Legacy CSS (gradients, backgrounds)
```

---

3. Key Pages & Features

3.1 Dashboard

- Displays device cards (live + historical), KPI cards, quick actions, recent activity, alias editor.
- `useDevices` polls backend every 2 seconds and listens to WebSocket events for immediate status updates.
- Alias data retrieved via `deviceMetadata.ts`, editable per device (dialog writes to localStorage + dispatches events).
- Search bar performs client-side filtering; fallback hits `/api/dashboard/search` for devices when backend reachable.

3.2 Device Manager

- Combines live devices with registry entries (devices seen in the past). Historical devices are flagged and have actions disabled.
- Filters: status, tag, manufacturer, historical/live, last-seen range. View toggle (cards vs. table).
- Bulk selection ribbon enables mass actions (disconnect/reboot/logs/pin/unpin).
- Each card features:
  - Alias + tag management (localStorage).
  - Notes textarea + checklist (drivers/USB debugging/SIM).
  - Activity summary (counts of workflows/modules).
  - Notebook icon opens a timeline modal (entries from `deviceActivity.ts`).

3.3 Workflows (FlowComposer)

- Lists stored workflows, supports search/filter (active/draft).
- Editor dialog to create/update workflows with repeat settings.
- Runner handles start/pause/resume/stop:
  - Maintains run state via `runningWorkflows`, `workflowActiveModuleIndex`, `workflowPauseRequests`.
  - Uses `AbortController` to cancel in-flight module fetches when Stop is triggered.

– Displays status messages per workflow, with snackbar notifications on success/failure.

3.4 Modules / Test Modules

- Grid view of module catalog (draggable chips, add-to-workflow).
- Ping module uses persisted config (target/duration/interval) with validation helpers.
- Ping "Run" button reflects execution state (green + "Running…" while backend call in progress).
- Workflow builder supports drag/drop reorder, storage of drafts, and checklist for waiting time module.

3.5 Electron Integration

- `App.tsx` detects `window.electronAPI` to switch to `HashRouter` and retrieve backend URL.
- Preload exposes `getBackendUrl`, `getAppVersion`, `restartBackend`, `trackEvent`.
- When packaging, `npm run build` must run so Electron can load `src/frontend/build/index.html`; otherwise `ERR_FILE_NOT_FOUND`.

---

4. State & Persistence

Data	Storage	Notes
Selected devices	localStorage (`deviceSelection`) + events	Kept in sync between Dashboard and other pages
Device metadata (alias, tags, notes, checklist)	localStorage (`deviceManagerMetadata`)	Device Manager + Dashboard read/write
Workflow drafts	localStorage (`workflowBuilderDraft`)	Restored automatically when builder reopened
Device registry/activity	localStorage via `deviceRegistry.ts`, `deviceActivity.ts`	Enables historical view/timelines
Telemetry	`telemetry.ts` (navigator.sendBeacon or Electron IPC)	Events logged for dashboard view/search

---

5. Styling & UX Guidelines

- Use `dashboardTheme` tokens for card spacing, colors, shadows.
- Cards should respect `radius: 16px`, `spacing: 24px`, `card.shadows` to keep UI consistent.
- Avoid inline hex values when token exists (e.g., `tokens.colors.primary`).
- For responsive layouts, prefer CSS grid with `auto-fit/minmax` to handle expanding device lists.
- Accessibility: buttons must remain reachable via keyboard; focus states should remain visible (MUI defaults + custom overrides).

---

6. Build & Deployment

1. `npm install` (root + `src/frontend` + `src/electron`).
2. `npm run build` inside `src/frontend` to produce `build/`.
3. Electron dev mode: `node_modules/.bin/electron.cmd src/electron`.
4. Production bundling: copy `build/` assets into backend static folder (`src/backend/static`) via deploy script (PowerShell helper available).
5. CI should run lints/tests (future), then package Electron app and backend artifacts.

---

7. Extending the Frontend

- **Adding a page**: create component under `pages/`, add lazy import + route in `App.tsx`, update sidebar nav (`Layout.tsx`).
- **Adding API call**: create service entry under `services/`, reuse `resolveBaseUrl` to respect backend override.
- **Shared state**: prefer hooks/local storage over introducing global state until we adopt a store (Zustand/Redux) for scheduler & chat features.
- **Large features (scheduler/chat/live logs)**: consider dedicated contexts or portals to stream real-time data without overloading existing hooks.

Last updated: 2025-12-12