

Telco ADB Automation – Installation & User Guide (v1.0.0)

This guide covers the exact steps to install and run the **Telco ADB Automation** desktop suite on Windows and Ubuntu, followed by a concise user manual describing day-to-day operations (device onboarding, module runs, workflow execution, and troubleshooting).

1. Architecture Recap

- **Desktop shell:** Electron 28 (Node.js 18) bootstrapping the React front-end and FastAPI backend (`simple-server.py`).
- **Frontend:** React 18 + Vite, Material UI, TypeScript (living under `mon-projet/src/frontend`).
- **Backend:** FastAPI + Pydantic + Python 3.11, modules orchestrate ADB scripts (`mon-projet/src/backend` and `modules/telco_modules.py`).
- **Runtime requirements:** Android Debug Bridge (`adb`), Node.js, Python, and device USB drivers.

Keep this structure in mind when installing; each platform section ensures both Node/Electron and Python/ADB pieces are configured.

2. Windows Installation

2.1 Prerequisites

Requirement	Recommended Version	Notes
Windows	10 21H2+ / 11	64-bit
Node.js	18.x LTS	Required for Electron + React build
npm	Bundled with Node	Used for dependency install
Python	3.11.x	Backend (<code>simple-server.py</code>)
pip	Latest	<code>python -m pip install --upgrade pip</code>
ADB / Platform Tools	r34+	Install via Android SDK or packaged zip
USB Drivers	OEM specific	Samsung, Google, etc.

Ensure `node`, `npm`, `python`, and `adb` are all resolvable from PowerShell (Get-Command `node` etc.).

2.2 Clone / Update the Repository

```
git clone https://github.com/<org>/ADB-automation-tool.git  
cd ADB-automation-tool\mon-project
```

If already cloned, pull latest: `git pull`.

2.3 Backend Setup

```
python -m venv .venv  
.venv\Scripts\Activate.ps1  
python -m pip install --upgrade pip  
pip install -r requirements-mongodb.txt
```

The requirements file bundles FastAPI, uvicorn, Pydantic, and telco module dependencies. Repeat `pip install -r requirements.txt` if you maintain an alternate lock file.

2.4 Frontend / Electron Setup

```
# From mon-project/  
npm install  
npm run build --prefix src/frontend      # builds React assets  
npm run build                            # optional: compile TypeScript/Electron preload
```

2.5 Installing the Packaged .exe

1. Download TelcoADBFramework-Setup-1.0.0.exe (NSIS) or TelcoADBFramework-1.0.0-portable.exe from the release artifacts.
2. **NSIS installer**
 - Right-click → “Run as administrator” (ensures drivers/services can be registered).
 - Follow the wizard: accept license → choose install directory → allow creation of Start menu/Desktop shortcuts.
 - Optionally enable “Launch at login” to keep the tray app available after boot.
 - At the end, keep “Launch Telco ADB Automation” checked to auto-start.
3. **Portable executable**
 - Extract the provided ZIP (if applicable) to C:\ADBFramework.
 - Double-click `TelcoADBFramework.exe`; create your own shortcut if needed.
4. First launch prompts Windows Defender SmartScreen (unsigned builds); click **More info** → **Run anyway**.
5. Verify installation:

- Start menu entry “Telco ADB Automation” opens the Electron shell.
- %LOCALAPPDATA%\Programs\TelcoADBFramework\resources\ contains packaged frontend + backend.

Silent install for enterprise scripting: TelcoADBFramework-Setup-1.0.0.exe
/S /D=C:\TelcoADBFramework.

2.6 Running in Development

Two options exist:

1. One-shot helper script

```
./launch-app.ps1      # starts backend + Electron renderer
```

2. Manual steps

```
# Terminal 1
cd mon-projet
.\venv\Scripts\Activate.ps1
python simple-server.py

# Terminal 2
cd mon-projet/src/frontend
npm start           # Vite dev server (optional)

# Terminal 3
cd mon-projet
npx electron .
```

2.7 Packaging (Optional)

To produce an installer:

```
npm run dist          # invokes electron-builder (see package.json)
```

Artifacts land under mon-projet/dist/ (NSIS) and mon-projet/build/.

3. Ubuntu Installation (x86_64 / AMD64)

3.1 Prerequisites

```
sudo apt update
sudo apt install -y curl git build-essential python3.11 python3.11-venv python3-pip adb
```

Install Node.js 18 (via NodeSource):

```
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs
```

Confirm versions: node -v, npm -v, python3.11 -V, adb version.

3.2 Clone Repo

```
git clone https://github.com/<org>/ADB-automation-tool.git
cd ADB-automation-tool/mon-projet
```

3.3 Backend Setup

```
python3.11 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
pip install -r requirements-mongodb.txt
```

3.4 Frontend / Electron Setup

```
npm install
cd src/frontend && npm install && npm run build && cd ../../
```

3.5 Installing the Linux Package / AppImage

When using the release artifacts:

- **Deb package** (telco-adb-automation_1.0.0_amd64.deb)
`sudo apt install -y ./telco-adb-automation_1.0.0_amd64.deb`
This installs binaries under /opt/telco-adb-automation and desktop entry Telco ADB Automation.
- **AppImage**
`chmod +x TelcoADBFramework-1.0.0-x86_64.AppImage
./TelcoADBFramework-1.0.0-x86_64.AppImage`
Optional integration prompt lets you create a launcher entry in
~/.local/share/applications.
Both artifacts target AMD/Intel 64-bit CPUs (Ubuntu 22.04+). For
ARM platforms you must run from source.

3.6 Development Run

```
# Terminal 1 - backend
source .venv/bin/activate
python simple-server.py      # listens on 0.0.0.0:8007 by default

# Terminal 2 - Electron (uses built frontend if present)
npx electron .
```

If you prefer Vite hot reload:

```
cd src/frontend  
npm run dev -- --host
```

Launch Electron with `npm run electron-dev` (see `package.json` scripts) to point at the dev server.

3.7 Packaging

Electron Builder can produce AppImage/DEB on Ubuntu:

```
npm run dist:linux
```

Configure `electron-builder.yml` (or `package.json > build`) for Linux targets before distributing.

4. User Manual (v1.0.0)

4.1 First Launch

1. Connect target Android devices via USB with USB debugging enabled.
2. Start the app (`launch-app.ps1`, `start-app.bat`, or `npx electron .`).
3. Wait for the backend banner: `Uvicorn running on http://0.0.0.0:8007`.
The dashboard then populates within ~3 seconds.

4.2 Dashboard

- Device cards/table display Serial Alias FriendlyName.
- Click the **edit icon** to rename alias; changes persist via API.
- Quick actions (Ping, Call, Airplane) are available per device.
- **Start Workflow** button opens workflow selector with the currently highlighted devices.

4.3 Device Manager

- Shows all devices ever connected (active + historical).
- Each row offers: status, tags, last seen timestamp, and a **notebook icon** that opens the activity timeline (modules/workflows run with timestamps).
- Use filters/search to locate QA collections. (Pinned/Collections widgets were removed for clarity in this release.)

4.4 Modules Page

- Lists each telco module with description and parameter form.
- **RUN behavior:** When you click RUN on Ping, button turns green for the full backend execution (no modal). Other modules show inline status chips.
- Failures are logged in device history; watch the console for backend traces if needed (`mon-projet/simple-server.py`).

4.5 Workflows

1. Drag modules from the left palette to the workflow canvas (drag-and-drop works L R).
2. Arrange modules; double-click to edit parameters.
3. Press **Start Workflow**:
 - The runner ensures module $n+1$ waits until module n reports success (fix for multi-call sequences).
 - Progress indicator highlights only the truly running module.
 - Use **Stop Workflow** to cancel; UI halts progression when backend acknowledges the abort.

4.6 Live Activity & Logs

- Device history captures: module/workflow name, result, timestamp, device serial.
- To inspect backend logs, check the Electron console or the terminal running `simple-server.py`.
- Known behavior: we intentionally suppress toast popups such as “Airplane Mode On – latest results...” to avoid clutter; refer to history instead.

4.7 Troubleshooting

Symptom	Fix
Device takes long to appear/disappear	Ensure adb server running (<code>adb kill-server && adb start-server</code>); app already polls faster (1s connect / 2s disconnect).

Symptom	Fix
Ping shows “Failed” instantly	Check that device responds to <code>adb shell ping</code> ; verify module script in <code>modules/telco_modules.py</code> .
Stop workflow ignored	Confirm backend log shows <code>Received stop signal</code> ; if not, ensure WebSocket connection alive or restart backend.
Electron “Renderer entry not found”	Run <code>npm run build --prefix src/frontend so src/frontend/build/index.html</code> exists before launching Electron prod mode.
Pydantic validator warning	Current backend still uses V1 validators; warning is harmless but upgrade path is documented in backlog.

4.8 Shutdown

- Use `Stop Workflow` to halt all executions.
 - Quit via system tray or `Ctrl+C` on backend terminal.
 - Disconnect devices only after workflow completion to preserve log continuity.
-

5. Appendices

- **Scripts of interest:**
 - `launch-app.ps1`, `start-app.bat`, `launch-without-node.ps1` (Windows helpers)
 - `simple-server.py` (FastAPI)
 - `modules/` directory for telco-specific automation
 - `docs/exports/` for PDF/DOCX deliverables
- **Command summary:**

```
# Windows
.\launch-app.ps1

# Ubuntu
```

```
source .venv/bin/activate && python simple-server.py  
npx electron .
```

- **Support:** Log issues in the repo's tracker with backend logs + device history screenshots for faster triage.

Last updated: 2025-12-12 (aligned with v1.0.0 feature set)