Job Mailer — Flow Documentation (End-to-End)

A) Big picture

This project has **two major execution paths**:

1) **Automation (headless)**

- Watch `data/recipients.csv` for new entries → send immediately
- Also run a scheduled job (cron) to send all pending recipients

2) **Interactive UI (web)**

- Send a single email
- Send bulk emails (Excel or pasted list)
- Manage defaults (smtp/subject/body/default resume)
- HR Finder (discover recruiting emails)
- ATS score/optimize resume vs JD

B) Data flow (persistent files)

- `data/recipients.csv` → source list for automation mode
- `data/sent.json` → idempotency + status by email
- `data/sent.xlsx` → append-only-ish record of sends for reporting/download
- `data/ui-settings.json` → UI-saved defaults
- `data/default-resume.pdf` → default resume used when UI send has no upload
- `data/companies.json` → HR Finder company dropdown list

C) Automation mode flows

C1) Cron scheduled send **Trigger**: cron tick based on `SCHEDULE_CRON` and `TIMEZONE`

Sequence: 1. `backend/src/index.js` starts scheduler. 2. scheduler tick → calls `sendPending({source:"cron"})` 3. `sendPending`: - reads recipients from `data/recipients.csv` - reads status from `data/sent.json` - filters pending recipients - creates SMTP transporter (verify + fallback 465) - sends each email with delay - logs: - JSON (`sent.json`) mark sent/error - Excel (`sent.xlsx`) append row

Output: - recipients get emails - logs updated

C2) Watcher immediate send **Trigger**: file change in `data/recipients.csv`

Sequence: 1. `watcher.js` loads recipients and keeps a set of known emails. 2. On change, it reloads recipients. 3. It computes "added emails". 4. It filters

out those already marked sent in `sent.json`. 5. It sends only the new ones. 6. It logs status to `sent.json`.

Why this is useful: - You can paste/add new recipients in the CSV and the system automatically sends without waiting for cron.

D) UI flows

D1) Login gate (optional) **If** `UI_AUTH_USER` and `UI_AUTH_PASS` exist: - user must login at `/login` - server sets `jm_sid` cookie - all UI + API routes (except `/health`, `/login`, `/api/login`) require auth

---

D2) Send single email (UI) **Trigger**: user fills Send form and clicks "Send Email"

Sequence: 1. Browser builds `FormData`: - `email`, `name`, `subject`, `body`, optional `resume` 2. `POST /api/send` 3. Server builds effective settings: - SMTP settings (env + UI overrides) - From name/email - Default subject/body - Default resume if no upload 4. Server decides content: - If body is provided → treat as override - Else → use default template builder 5. Nodemailer `sendMail`. 6. Server appends to `sent.xlsx`. 7. UI shows result including whether defaults were used.

D3) Bulk send via Excel **Trigger**: user uploads `.xlsx` and clicks "Send Bulk Emails"

Sequence: 1. UI sends `POST /api/send-bulk` with `excel` (+ optional `resume`). 2. Server parses first sheet: - expected columns: email, recipient name, subject, body - flexible header matching; dedupe by email 3. For each row: - subject/body override if present, else defaults - send with shared transporter - log to `sent.xlsx` 4. UI shows summary: total/sent/failed + samples.

D4) Bulk send via pasted list **Trigger**: user pastes comma/newline emails and clicks "Send Bulk Emails"

Sequence: 1. UI sends `POST /api/send-list`. 2. Server parses & dedupes emails. 3. Uses defaults subject/body. 4. Sends and logs like Excel flow.

D5) Defaults management **Trigger**: user edits defaults and clicks Save

Sequence: 1. `GET /api/settings` loads saved settings and indicates if password/resume are set. 2. `POST /api/settings` saves settings (password is not returned back). 3. `POST /api/settings/resume` uploads and stores `data/default-resume.pdf`. 4. Send tabs automatically reflect default subject/body if empty.

D6) HR Finder **Trigger**: user enters company and/or domain and clicks Search

Sequence: 1. UI calls `GET /api/hr-lookup?company=...&domain=...&provider=hunter|apollo` 2. Server determines domain: - If domain provided, use it. - Else use Clearbit autocomplete to infer domain. - If company has suffixes (Pvt/Ltd/etc), server retries simplified variants. 3. Provider query: - Hunter: domain-search → filter recruiting titles → fallback to all - Apollo: people search → normalize contact fields 4. UI renders cards with copy/send buttons.

D7) ATS Score + Optimize **Trigger**: user uploads resume (or pastes text) and JD

Sequence: 1. UI posts JD + resume to `/api/ats-score` or `/api/ats-optimize`. 2. Server extracts text. 3. Computes score + suggestions. 4. Optimize path generates a downloadable PDF at `/api/ats-optimized.pdf`.

E) Failure & recovery flows

- SMTP network blocks → transporter fallback 465 attempt (587 + START-TLS case)
- Per-recipient failure doesn't stop bulk send → errors recorded per email
- Logs are written atomically to avoid corruption on crash

F) "Why it works" summary

This system is reliable because: - **Idempotency**: `sent.json` prevents re-sending in automation mode - **Resilience**: SMTP verify + fallback strategy - **Operational visibility**: Excel log + UI status - **Automation + UI**: supports both headless workflows and interactive workflow