Job Mailer — How to Explain to an Interviewer (Story + Highlights)

1) 20–30 second pitch (elevator pitch)

"I built a Node.js tool called Job Mailer that automates job application emails. It supports both a headless automation mode and a clean local web UI. The automation mode reads a recipients CSV, sends only the pending emails on a schedule, and also watches the CSV so adding a new recipient triggers an immediate send. The UI lets me send single or bulk emails, manage SMTP/default templates, and even discover HR emails using external provider APIs. Everything is logged locally to JSON and Excel so I can track what was sent and avoid duplicates."

---

2) The problem I solved (what motivated the project)

- Sending job applications repeatedly is repetitive and error-prone.
- Manual sending leads to:
  - duplicated emails
  - inconsistent templates
  - poor tracking of who was contacted
  - time wasted managing attachments and bulk lists

This project turns that into a reliable workflow with idempotency + logging.

---

3) What the user can do (demo script)

If you're demoing live:

1) **Show UI** (`npm run ui`)

- Login (optional)
- Go to Send tab → send one email
- Show that you can leave subject/body blank and defaults are used

2) **Show Bulk**

- Upload Excel template
- Send bulk and show summary + `sent.xlsx` download

3) **Show Defaults**

- Update SMTP/from/subject/body
- Upload default resume

4) **Show HR Finder**

- Enter company or domain → fetch HR contacts

- Copy email / send directly

5) **(Optional) Show Automation mode**

- Open `data/recipients.csv`
- Add a new email row and show watcher sends immediately

---

4) Best functionalities to highlight (the "best of best")

A) Idempotent sending (no duplicates)

- Uses `data/sent.json` as a per-email state store.
- The scheduler sends only pending recipients.

Why it impresses: shows you understand real automation needs (statefulness).

B) SMTP reliability improvement

- Before sending, it verifies SMTP.
- If a common failure occurs on 587 STARTTLS (network blocked), it retries 465 SSL.

Why it impresses: shows practical production-like handling of flaky networks.

C) Bulk workflows with good UX

- Excel upload parsing (flexible headers, dedupe by email)
- Pasted list mode (comma/newline)
- Shows results and saves a downloadable log

Why it impresses: it's not just backend; it's end-to-end product thinking.

D) HR Finder integration

- External APIs (Hunter / Apollo)
- Domain resolution from company name
- UI contacts card rendering with "send now" action

Why it impresses: integration engineering + mapping inconsistent API shapes.

E) Defaults + template system

- Saved defaults reduce repeated typing.
- Signature injection prevents broken/duplicated signatures.

Why it impresses: attention to content quality and maintainability.

---

5) Challenges faced (and how you solved them)

1) "How do I prevent duplicates?"

- Solution: `sent.json` idempotency store keyed by email + status transitions.

2) "SMTP networks are unreliable"

- Solution: verify transporter + conditional fallback to port 465.

3) "Bulk parsing is messy"

- Excel has inconsistent column names and duplicates.
- Solution: normalize headers, accept multiple variants, dedupe while preserving best info.

4) "External HR APIs return inconsistent shapes"

- Solution: normalize fields, map common structures, cap results, and handle missing data.

5) "UI dropdown selection didn't work for some company names"

- Root cause: long legal suffixes reduce domain-detection accuracy.
- Solution: server retries simplified variants (strip Pvt/Ltd/etc; use first words).

---

6) Trade-offs (good engineering maturity)

- UI auth uses in-memory sessions → simple for local use, but not for multi-instance production.
- Logs are local files → easy, portable, but not multi-user/centralized.
- SMTP is convenient but can be blocked → email API provider would be more reliable.

---

7) How you would improve it (future scope)

- Add queue + retries with exponential backoff.
- Add a database (SQLite/Postgres) for scalable logs and reporting.
- Add rate limiting + provider abstraction layer.
- Add automated tests for parsers and key flows.

---

8) Strong closing line

"This project demonstrates I can design a full workflow end-to-end: file-based automation + web UI + external API integrations + reliable sending + logging/idempotency, with clear trade-offs and room for scaling."