FrontEnd Assessment - Candidate Instructions

Objective

Build a small, production-style slice of a real-time dashboard:
- Virtualized device table with filtering
- Details page with a small live CPU chart (30s window)
- Reboot control with optimistic UI and idempotency
- Basic resilience under streaming load

Stack

Pre-wired: Next.js (App Router), React 19, TypeScript, Tailwind, Zustand, react-virtuoso, Vitest.

Setup

1) Requirements: Node 20 (nvm use 20)
2) Install: npm i
3) Start simulator (WebSocket + control API): npm run sim
   - WebSocket: ws://localhost:4001
   - Control API: POST http://localhost:4002/control/reboot
4) Start app: npm run dev -> open http://localhost:3000/devices
5) Run tests: npm test

Build This

1) Devices table (/devices)
   - Virtualized list: ID, status, CPU%, RAM%, last seen, Reboot
   - Filter by ID or status (basic debounce is fine)
   - Keyboard reachable. Visible focus. Correct table semantics

2) Details page (/devices/[id])
   - 30 second rolling CPU line chart (SVG is fine; throttle to about 5-10 fps)
   - Show latest CPU, RAM, status, and timestamp

3) Reboot action
   - Implement using a Next.js Server Action (stub provided)
   - Optimistic update: set status to "rebooting" immediately
   - Rollback on server error (the simulator may NACK)
   - Include an idempotency key to guard against double submit

4) Resilience and performance
   - Auto-reconnect WebSocket with backoff (already scaffolded)
   - Batch incoming telemetry to keep typing responsive

- Tolerate duplicates and out-of-order events (simulator includes "seq")

Acceptance Criteria

- Correctness:
  * Optimistic reboot applies immediately and rolls back on NACK
  * Auto-reconnect restores streaming without a manual refresh
  * No obvious regressions from duplicate or out-of-order telemetry

- Performance:
  * Filtering remains responsive while telemetry streams
  * Chart updates are throttled (10 fps or less) and do not block input

- Accessibility:
  * Table uses appropriate semantics and is keyboard reachable
  * Focus states are visible

- Testing:
  * Provide 2 to 3 unit tests (for example: store upsert, filter logic, or optimistic flow)

- Docs:
  * README.md with run instructions
  * NOTES.md explaining state choice, batching, trade-offs
  * PERF.md with brief notes on responsiveness and next steps

Submission

Choose ONE of the following:

A) Git repository link (public or invite the reviewer)
   - Recommended if you want us to review commit history
   - Ensure the project runs with: "npm run sim", "npm run dev", "npm test"

B) ZIP archive of the project root
   - Name the file like: YOURNAME-FrontEnd-Assessment.zip
   - Include all source files and docs (README.md, NOTES.md, PERF.md)
   - EXCLUDE heavy build folders: node_modules, .next, dist
   - OPTIONAL: Include the ".git" folder if you want us to see your commit history
   - The project should run after unzip with: "npm i", "npm run sim", "npm run dev"