React Technical Assignment (Front-End Only)

PROJECT: TALENTFLOW – A MINI HIRING PLATFORM (NO BACKEND)

Build a React application that lets an HR team manage:

1. **Jobs** (create, edit, archive, reorder)
2. **Candidates** (apply to jobs, progress through stages)
3. **Assessments** (job-specific quizzes/forms the candidate completes)

Core flows to implement

1. **Jobs board**
    - List with server-like pagination & filtering (title, status, tags).
    - Create/Edit job in a modal or route. Validation (title required, unique slug).
    - Archive/Unarchive. Reorder via drag-and-drop with **optimistic updates** and rollback on failure.
    - "Deep link" to a job: /jobs/:jobId.
2. **Candidates**
    - Virtualized list (1000+ seeded candidates) with **client-side search** (name/email) and **server-like** filter (current stage).
    - Candidate profile route: /candidates/:id showing timeline of status changes.
    - Move candidate between stages with a kanban board (drag-and-drop).
    - Attach notes with @mentions (just render; suggestions from local list).
3. **Assessments**
    - Assessment builder per job: add sections and questions (single-choice, multi-choice, short text, long text, numeric with range, and file upload *stub*).
    - Live preview pane that renders the assessment as a fillable form.
    - Persist builder state and candidate responses locally.
    - Form runtime with validation rules (required, numeric range, max length) and conditional questions (e.g., show Q3 only if Q1 === "Yes").

Data & "API" (no real server)

Use **MSW or MirageJS** to simulate a REST API with the following resources:

- GET /jobs?search=&status=&page=&pageSize=&sort=
- POST /jobs → { id, title, slug, status: "active"|"archived", tags: string[], order: number }
- PATCH /jobs/:id
- PATCH /jobs/:id/reorder → { fromOrder, toOrder } (occasionally return 500 to test rollback)
- GET /candidates?search=&stage=&page=
- POST /candidates → { id, name, email, stage: "applied"|"screen"|"tech"|"offer"|"hired"|"rejected" }
- PATCH /candidates/:id (stage transitions)
- GET /candidates/:id/timeline
- GET /assessments/:jobId
- PUT /assessments/:jobId
- POST /assessments/:jobId/submit (store response locally)

Seed data

- 25 jobs (mixed active/archived), 1,000 candidates randomly assigned to jobs and stages, at least 3 assessments with 10+ questions each.
- Inject **artificial latency (200–1200ms)** and a **5–10% error rate** on write endpoints.

Important: All persistence must be **local** (e.g., IndexedDB via Dexie or localForage). Treat MSW/Mirage as the "network" layer but write-through to IndexedDB. On refresh, the app restores state from IndexedDB.

Deliverables

- Deployed App Link
- GitHub Repository Link
- README with setup, architecture, issues, and technical decisions

Evaluation Criteria

- Code Quality
- App Structure
- Functionality

- UI/UX
- State Management
- Deployment
- Documentation
- Bonus Features