Technical Assessment – Full-Stack Developer (Next.js/TypeScript) – Remote

Objective

Evaluate the candidate's ability to deliver a small but production-minded Next.js feature in **4–8 hours**. Focus: clean server/client boundaries, correct CRUD, pragmatic validation, and minimal yet meaningful tests.

Timebox: Target **~4 hours**; allow up to **8 hours maximum**. Favor a polished, minimal scope over breadth.

Task Overview — "Technician Work Orders" (Slim Scope)

Build a tiny web app to manage **Work Orders**.

Must-have User Stories

1. **List Work Orders:** Table/list with `title`, `priority` (Low/Medium/High), `status` (Open/In Progress/Done), `updatedAt`.
2. **Create Work Order:** Form with `title`, `description`, `priority`; default `status = Open`.
3. **Edit/Delete:** Update or remove an order.
4. **Detail View:** Page with full content.

5. **Search/Filter:** Filter by `status` **or** text search by `title` (pick one to keep scope tight and document the choice).

Data & Persistence (Simplified)

- **File-based JSON only** (no DB). Provide a tiny data module that reads/writes `data/work-orders.json`.

- Include a script to **seed** sample data (e.g., `pnpm seed` / `npm run seed`).

Architecture & Next.js Use

- Use **Next.js App Router**.
- Prefer **Server Components**; use **Client Components** only where interaction is needed (forms, filters).
- Implement CRUD with **Route Handlers** (`app/api/work-orders/route.ts`, etc.).
- Optional: Add a simple **cache hint** (`revalidate` or `cache: 'no-store'`) and explain in README.

Validation & Security (Pragmatic)

- Validate inputs server-side (e.g., **Zod**). Return field-level errors to the form.
- Render descriptions safely (avoid raw `dangerouslySetInnerHTML`).

UI/UX

- Clean, responsive UI using **Tailwind CSS** (or CSS Modules).
- Keyboard-friendly; clear success/error messages.

Testing (Right-sized)

- **Minimum:** 1–2 **unit/component** tests (Vitest/Jest + Testing Library) covering a core flow (e.g., create → list shows item).
- **Optional:** 1 **Playwright** happy-path E2E (create → detail → edit → list). Skip if timeboxed.

Non-Goals (Scope Guardrails)

- No database (Prisma/SQLite removed).
- No image/file uploads (make it a **Bonus** only if time remains).
- No authentication, RBAC, analytics, or real-time.

API Shape (Reference)

```
export type WorkOrder = {
  id: string;           // uuid
  title: string;        // 2–80 chars
  description: string;   // up to ~2,000 chars
  priority: 'Low' | 'Medium' | 'High';
  status: 'Open' | 'In Progress' | 'Done';
  updatedAt: string;     // ISO string
};
```

Suggested endpoints:

- `GET /api/work-orders` (optional query: `status` or `q`)
- `POST /api/work-orders`
- `PUT /api/work-orders/:id`
- `DELETE /api/work-orders/:id`

Acceptance Checklist

- App Router with server components; client components only where needed.
- CRUD via Route Handlers.
- List + create + edit + delete + detail implemented.
- Either status filter **or** text search implemented; decision noted in README.
- File-based JSON persistence with seed script.
- Server-side validation (Zod or equivalent) and safe rendering.
- TypeScript types for components, API payloads, and data module.
- Tests: 1–2 unit/component; optional Playwright E2E.
- README with setup, run, seed, and testing instructions; brief note on cache choice.
- **Self-presentation video** (max 1 minute) introducing the candidate.

How to Deliver

Provide a **GitHub repository** containing:

- Source code (Next.js + TypeScript, App Router).
- **README.md** with setup/run/seed, testing, and a short note on decisions made to stay within **4–8 hours**.
- **Tests** (unit/component; optional Playwright E2E).
- **Short demo video** (≤ 5 minutes) showing create/edit/delete and the chosen filter/search flow.
- **Self-presentation video** (≤ 1 minute) introducing yourself and motivation.

Evaluation Criteria (Weighted)

- **Architecture & Code Quality (30%)** – Clear server/client boundaries, neat modules, meaningful types.
- **Correctness (25%)** – CRUD works end-to-end; persistence reliable; validation enforced.
- **UX & Accessibility (15%)** – Usable, responsive, keyboard-friendly.
- **Testing (15%)** – Right-sized, reliable tests for the core path.
- **Documentation & Judgment (15%)** – Crisp README; explicit trade-offs to hit the 4–8h window.

Bonus (Optional)

- Tiny performance note (e.g., `revalidate` reasoning or memoization where it mattered).
- A11y touches (ARIA labels, focus management on submit).
- Minimal i18n scaffolding (static dictionary).

Keep it lean. Show good judgment, solid code, and a smooth developer experience within the timebox.