Interview Process (Stage III)

TODO App — Full-Stack Take-Home (8h total)

A small **Tasks (TODO) app** with a **React (TS)** frontend and an **Express.js (TS)** backend. It includes: multi-step creation with validation/autosave, safe HTML preview, live updates (SSE), simple auth with refresh rotation, background CSV export via a lightweight queue, and basic API hardening.

Deliverables (what to submit)

- Repo with /frontend and /backend.
- Seed script to create **one demo user** and a few tasks.
- Short **README** (setup, how to run, design notes: SSE vs WS; queue vs worker; server vs client state).
- **1 frontend test** (form flow) and **1 backend test** (reorder or validation).

Product scope (user stories)

1. **Sign in**

- Email + password login to access personal tasks (no registration).
- Short-lived **access token** + **refresh** cookie.

2. **Tasks list**

- See my tasks; filter by **status** (Open/Done).
- Toggle “Done”.
- **Reorder** via drag-and-drop; persisted order.

3. **Create task (wizard)**

- 3 steps:
    1. Title (required), Priority (Low/Med/High)
    2. Description (allows HTML), Due date (≥ today)
    3. Review & Create
- **Autosave** progress locally so a refresh doesn’t lose it.

- On success, show **toast**.

4. **Safe HTML preview**
- Preview sanitized description before saving.
5. **Live updates**
- Task list **updates in real time** across two tabs using **SSE**.
6. **Export CSV**
- "Export CSV" button creates a CSV export of **current filter**.
- Show progress (queued/running/done/failed) and enable **Download** when ready.

Tech constraints

Frontend

- **React + TypeScript + Vite**
- **@tanstack/react-query** for server state; local UI with React state/context.
- **react-hook-form + zod** for forms & validation.
- **React Router** (minimal).
- **Testing**: Vitest + React Testing Library.
- Optional libs: DOMPurify, a lightweight DnD (e.g., @dnd-kit/core) or simple up/down buttons if time is tight.

Backend

- **Node 18+, Express.js + TypeScript**
- **Prisma + SQLite** (portable and fast).
    - Use **optimistic locking** for reorders (version field) to avoid DB-specific locks.
- **SSE** endpoint for updates.
- **Validation**: zod/joi at the route boundary.
- **Auth**: **JWT** access token (≈10 min TTL) + **httpOnly+Secure** refresh cookie with **rotation** + **reuse detection** (simple DB table).
- **Security middleware**: helmet, cors (restrict origin), express-rate-limit, hpp.
- **Background export**: in-memory queue with retry/backoff (no Redis).

Data model (minimum)

```
User {
 id          string
 email       string (unique)
 passwordHash  string
}

Task {
 id          string
 userId      string (FK)
 title       string
 priority    enum('low','med','high')
 description   string (HTML allowed)
 dueDate     date | null
 done        boolean
 orderIndex    number
 version     number     // for optimistic locking on reorder
 createdAt    datetime
 updatedAt    datetime
}

RefreshToken {
 id          string
 userId      string
 tokenHash    string
 familyId     string     // for rotation family
 revoked      boolean
 createdAt    datetime
}

ExportJob {
 id          string
 userId      string
 scope       json       // filter snapshot
 status      enum('queued','running','done','failed')
 filePath     string | null
 error       string | null
 createdAt    datetime
 updatedAt    datetime
}
```

API (contract kept small)

Auth

- POST /auth/login → { accessToken } + sets refresh cookie.
- POST /auth/refresh → rotates refresh; returns new { accessToken }.
- POST /auth/logout → revokes current refresh.

Tasks

- GET /tasks?status=open|done → list (own tasks only).
- POST /tasks → create (zod-validated).
- PATCH /tasks/:id → partial update (done/title/priority/description/dueDate).
- POST /tasks/reorder → { items: [{id, orderIndex, version}] }
    - Use optimistic locking: if version mismatches, return 409.

Realtime

- GET /stream (SSE) → emits task_created|task_updated|task_reordered|export_update.

Export

- POST /exports/csv → { scope } (current filters). Returns { jobId }.
- GET /exports/csv/:jobId → { status, downloadUrl? }.
- GET /exports/csv/:jobId/file → downloads CSV (auth required).

Frontend features to implement (and why)

1. **Rendering & re-renders**
- Memoize derived lists (filtered/sorted tasks) with useMemo.
- Stable handlers for row items (useCallback) so rows don't thrash.
2. **useEffect correctness**
- One effect to subscribe/unsubscribe SSE; correct deps; cleanup on unmount.
- No fetch in render; use React Query for data and invalidations.
3. **Server vs client state**
- Tasks always from React Query; UI stuff (wizard step, filters) in component state.

- **Optimistic update** on toggle-done & reorder (rollback on 409).

4. **XSS prevention**

- Sanitize HTML via **DOMPurify** before dangerouslySetInnerHTML.
- Note a basic CSP in README (no need to configure in dev).

5. **Wizard + autosave + validation**

- LocalStorage autosave of {stepData} by user id.
- zod schema enforces title required, due ≥ today, max lengths.

6. **Export CSV (client flow)**

- Button “Export CSV” in header.
- Start export → toast “Export started”.
- Poll job status every 3s **or** listen to export_update via SSE; enable Download when ready.

7. **Minimal tests**

- **Form test**: invalid → shows errors; fix → submit; mock API; success toast appears.

Backend implementation notes

- **Event loop**: no blocking CPU in request handlers. Queue job does I/O; if you add CPU-ish work, comment that you’d offload to a Worker Thread (not required).
- **Queue vs Worker**: implement CSV as **queue** with retry/backoff (I/O + possible transient errors).
- **Realtime (SSE)**: one endpoint, send keep-alive comment every ~20–30s; broadcast on create/update/reorder/export status change.
- **Consistency on reorder**: optimistic locking using version; on conflict return 409 → client refetches & retries (React Query rollback).
- **Security**: helmet, cors (allow only http://localhost:5173 by default), rate limit (per IP), hpp, input validation (zod), cookies flagged httpOnly+Secure (document that Secure requires HTTPS).
- **JWT vs PASETO**: use **JWT** to keep libs simple; 10-minute access; refresh rotation stored in RefreshToken with family id. Reuse detection: if a refresh is presented after it’s already rotated (previous token revoked), revoke the whole family and force login.

Acceptance criteria (checklist)

- ✅ Login works; refresh rotation works; protected routes require a valid access token.
- ✅ Tasks list loads via React Query; filter by status.
- ✅ Toggle done updates optimistically; reconciles with server.
- ✅ Reorder persists and is **robust to concurrent edits** (409 path handled).
- ✅ Wizard validates; autosaves; submit creates task; success toast shows.
- ✅ Description preview is XSS-safe.
- ✅ Second tab receives **SSE** updates for create/update/reorder.
- ✅ Clicking "Export CSV" creates a job; status visible; final CSV downloads.
- ✅ One FE test (form) and one BE test (reorder or validation) pass.
- ✅ README briefly justifies SSE vs WS and queue vs worker; outlines server vs client state.

Time guidance

Backend (~4h)

- Scaffolding + Prisma schema + seed: 45m
- Auth (login/refresh/logout) + middleware: 60m
- Tasks CRUD + reorder (optimistic lock): 60m
- SSE stream + broadcast hooks: 30m
- Export queue + endpoints + CSV writer: 30m
- Security middleware + 1 test: 15m

Frontend (~4h)

- Vite scaffold, routing, React Query setup: 30m
- Login view + token plumbing: 30m
- Tasks list (filter, toggle) + SSE subscribe: 60m
- Reorder (DnD or up/down) + optimistic mutation: 45m
- Wizard with RHF+zod, autosave, toast: 45m
- Export button + status polling + download: 20m
- 1 form test (RTL + Vitest): 10m

Stretch (only if time remains)

- Presence of basic **rate-limit headers** in UI.
- **Retry** button on failed export.
- **Pagination** on tasks.
