Mini Task Board

Candidate Technical Test

Goal

Build a small full-stack application that demonstrates:

- Clean React UI with Tailwind CSS
- Django REST API
- Good UX including empty states, loading indicators, error handling, and responsiveness
- Sensible code structure and basic tests

Timebox (Recommended)

6 to 10 hours. Candidates may invest additional time if desired, but evaluation will be judged within this window.

Required Features

1. Authentication

Implement authentication using one of the following approaches:

Option A (Fast): No traditional auth. Require an X-API-KEY header checked in Django middleware.

Option B (Standard): Django authentication with token or JWT.

Must Include

- Login screen (or API key screen) with validation and error display
- Logged-in state persists (localStorage is acceptable)

2. Task Board UI (React + Tailwind)

Create a single page "Board" with three columns: Backlog, In Progress, and Done.

Task Fields

- title (required)
- description (optional)
- status (one of the three column values)
- priority (Low, Medium, or High)
- due_date (optional)
- created_at (auto-generated)

UI Behaviors

- Create task (modal or drawer preferred)
- Edit task (same modal)

- Delete task (with confirmation)
- Move task between columns: Drag and drop (preferred) OR status dropdown (acceptable)
- Search tasks by title (client-side filtering is acceptable)
- Filter by priority (client-side filtering is acceptable)

UX Requirements

- Loading states (skeleton or spinner)
- Empty state per column ("No tasks here yet...")
- Form validation messages
- Responsive layout (mobile friendly)
- Basic visual polish: spacing, typography hierarchy, buttons, hover states

3. Django API (REST)

Use Django REST Framework (recommended).

Endpoints (Minimum)

- GET /api/tasks/ (list)
- POST /api/tasks/ (create)
- GET /api/tasks/:id/ (detail)
- PUT/PATCH /api/tasks/:id/ (update)
- DELETE /api/tasks/:id/ (delete)

Rules

- Validate required fields
- Ensure status and priority are enum-like
- Return meaningful error responses

Optional Bonus Features

Select 2 to 3 of the following to stand out:

- Pagination or server-side filtering
- Optimistic UI updates
- Keyboard accessibility (focus trap in modal, ESC to close)
- Dark mode toggle
- Animations (subtle transitions)
- Audit log ("Task moved to Done")
- docker-compose for setup

Testing Requirements

Backend Tests (Required)

At least 3 tests covering:

- Create task validation
- Update task status
- Delete task

Frontend Tests (Optional but Recommended)

At least 1 test covering:

- Create task form validates required title OR renders tasks list

Delivery Requirements

Candidate must provide:

1. Git repository
2. README.md with:
   - Setup steps (backend and frontend)
   - Environment variables
   - How to run tests
   - Short "decisions" section explaining choices made
3. Sample .env.example file
4. Clean commits (not one large commit)

Evaluation Rubric

Category	Weight	Criteria
Backend	30%	Clean models, serializers, views; validation and errors; API consistency; tests pass
Frontend	40%	Component structure; state management clarity; Tailwind quality and responsiveness; UX polish (empty, loading, error states)
Engineering	20%	Readable code, naming, linting; README quality and reliability; git hygiene
Product Thinking	10%	Small UX touches; sensible defaults and constraints

Notes for Candidate

- You may use any libraries (React Query, Zustand, Redux Toolkit, etc.)
- Prefer clarity over cleverness
- Do not overbuild - ship something polished and working
- Use JWT authentication or API key authentication (mid-level difficulty)

Good luck!