[PRD] Developer Assessment: Task Board System - 2026-01

Status: Published
Audience: Developer Candidates
Last Updated: 2026-01-09
Owner: @engineering-team
Collection: Hiring & Assessment

Executive Summary

This is a **2-hour technical assessment**. You will build a task management system with multiple boards and tasks. Each board contains multiple tasks that can be created, updated, and deleted.

What You Will Build:

1. Dashboard page showing all boards
2. Board detail page showing all tasks in a board
3. Backend API for boards and tasks
4. Working application with database

What You Must Submit:

1. Working code (GitHub repository or ZIP file)
2. AI workflow document (how you used AI tools)
3. Architecture decisions document (why you made your technical choices)

Required Tools

AI-Integrated IDE (Required)

You **must** use one of these AI development tools:

- **Cursor** (with Claude)
- **Claude Code**

- **AntiGravity**

This is required because we use AI tools in our daily work. We want to see how you use AI to build applications.

Other Resources You Can Use

Yes, you can use:

- AI assistants and AI agents
- Documentation websites (React docs, Next.js docs, etc.)
- Stack Overflow
- npm packages
- Your own code templates

No, you cannot use:

- Complete task management templates
- Another person writing code for you
- Copying full solutions without understanding them

Technology Stack (Required)

You **must** use these technologies:

Frontend:

- React 18+
- Next.js 14+ (App Router or Pages Router)
- TypeScript (required)

Backend:

- Next.js API Routes OR Server Actions

Database:

- Choose one: PostgreSQL, MySQL, or SQLite
- Use Prisma ORM (recommended) or another ORM

Styling:

- Choose one: Tailwind CSS, CSS Modules, or styled-components

Why These Requirements: This is our production stack. We need to see that you can work with these specific technologies.

Time Limit

Total Time: 2 hours

This is designed to be completable in 2 hours with AI tools. You don't need to build everything perfectly - we want to see your process, priorities, and how you use AI to work efficiently.

Data Models

You need two database tables with a relationship.

Board Table

A board is a container for tasks.

Required Fields:

- `id` (primary key, auto-generated)
- `name` (text, required)
- `created_at` (timestamp, auto-generated)

Optional Fields (your choice):

- `description` (text)
- `color` (text)
- `updated_at` (timestamp)

Task Table

A task belongs to one board.

Required Fields:

- `id` (primary key, auto-generated)
- `board_id` (foreign key to Board, required)
- `title` (text, required)

- `status` (must be one of: "todo", "in_progress", "done")
- `created_at` (timestamp, auto-generated)

Optional Fields (your choice):

- `description` (text)
- `assigned_to` (text)
- `priority` (text: "low", "medium", "high")
- `due_date` (date)
- `updated_at` (timestamp)

Database Rules:

- One Board can have many Tasks
- One Task belongs to only one Board
- When you delete a Board, decide what happens to its Tasks (delete them too, or prevent deletion)

What You Must Build

Required Features

These features are required. You must build all of these:

1. Dashboard Page (Homepage)

Route: `/` or `/dashboard`

Must show:

- List of all boards (display as cards or list)
- Button to create a new board
- Click a board to go to that board's detail page

Must work:

- Create a new board with a name
- See all boards that exist in the database
- Click a board to navigate to its page

2. Board Detail Page

Route: `/board/[id]` or `/boards/[id]`

Must show:

- Board name at the top
- All tasks in this board
- Tasks organized by status (todo, in_progress, done)
- Button or form to create a new task
- Way to go back to dashboard

Must work:

- Create a new task in this board
- See all tasks for this board
- Change a task's status (todo → in_progress → done)
- Edit a task's title
- Delete a task
- Changes appear immediately (no page refresh needed)

3. Backend API

Required API Endpoints:

For Boards:

- Create a board (POST)
- Get all boards (GET)
- Get one board with all its tasks (GET)
- Delete a board (DELETE)

For Tasks:

- Create a task (POST)
- Get tasks for a board (GET)
- Update a task (PATCH or PUT)
- Delete a task (DELETE)

Rules:

- Use correct HTTP methods (GET, POST, PUT/PATCH, DELETE)
- Return correct status codes (200, 201, 400, 404, 500)
- Validate data (check for required fields)
- Handle errors (missing data, invalid IDs, database errors)

4. Data Interaction

Must work on the frontend:

- Filter tasks by status (show only "todo" or only "done", etc.)
- Sort tasks by at least one field (created date, priority, etc.)
- Update task status without refreshing the page
- When you create or delete something, the list updates immediately

---

Bonus Features (Optional)

If you have extra time, you can add **one** of these features:

Option A - Analytics: Add a section on the dashboard showing:

- Total number of tasks across all boards
- How many tasks are in each status
- Percentage of completed tasks

Option B - Real-Time Updates: If you open the app in two browser windows, changes in one window appear in the other window automatically (use WebSockets or polling)

Option C - Export Data: Add a button to download all board data as a JSON or CSV file

Note: These are optional. Only add one if you finish the required features and have time remaining.

Pages and Routes

Your application should have these routes:

Homepage / Dashboard: `/` → Shows all boards

Board Detail: `/board/[id]` → Shows all tasks for one board

API Routes (if using API Routes instead of Server Actions):

- `/api/boards` → Board operations
- `/api/tasks` → Task operations

What You Must Submit

1. Code Repository

Submit one of these:

- GitHub repository (public or give us access)
- ZIP file with all code
- Deployed application URL + GitHub link

Must include:

- All source code
- README.md file
- package.json with all dependencies
- Database schema or migration files
- .env.example file (if you use environment variables)

2. README File

Your README.md must explain:

How to install:

```
None
1. What software is needed (Node.js version, etc.)

2. How to install dependencies (npm install, etc.)

3. How to set up the database

4. How to run the application

5. What URL to open in the browser
```

Tech stack: List all major technologies and libraries you used

Example:

```
None
# Task Board System
```

```
## Requirements
- Node.js 18 or higher
- PostgreSQL (or SQLite for local development)


## Setup
1. Install dependencies: `npm install`
2. Set up database: `npx prisma migrate dev`
3. Run the app: `npm run dev`
4. Open: http://localhost:3000


## Tech Stack
- Next.js 14 with App Router
- TypeScript
- Prisma ORM
- PostgreSQL
- Tailwind CSS
```

3. AI Workflow Document

Time limit: Spend only 15 minutes writing this document

File name: `AI_WORKFLOW.md`

What to write:

1. What AI tool did you use?

- Which tool: Cursor, Claude Code, or AntiGravity?
- How did you use it?

2. Example prompts:

- Write 2-3 actual prompts you used
- Explain why you wrote the prompt this way
- Did the AI give you a good answer?

3. Your process:

- What did you use AI for? (scaffolding, debugging, etc.)
- What did you code manually?
- Where did AI-generated code not work?
- How did you fix problems?

4. Time management:

- What did you build first?
- What did you skip?
- If you had more time, what would you add?

Example:

None
```
# AI Workflow



## Tool Used

I used Cursor with Claude integration.



## Example Prompts

1. "Create a Prisma schema with Board and Task models where Task
has a foreign key to Board"

   - This worked well and generated the correct schema
```

```
2. "Create a Next.js API route to get all boards with their task
counts"

   - Had to fix the Prisma query to include task count


## My Process

- Used AI to generate initial Prisma schema and API routes

- Manually coded the frontend components and state management

- AI-generated code for task filtering needed debugging

- Used AI to help with TypeScript types


## Time Management

- First 30 min: Set up project, database, and schema

- Next 45 min: Built API routes and tested with Postman

- Next 30 min: Built frontend pages and components

- Last 15 min: Added filtering and fixed bugs

- Skipped: Styling, bonus features
```

4. Architecture Decisions Document

File name: `ARCHITECTURE.md`

What to write:

1. Why did you choose your technologies?

- Why App Router or Pages Router?

- Why this database?
- Why this styling approach?

2. How did you structure your data?

- Why did you design the database tables this way?
- What happens when you delete a board?

3. How did you design your API?

- What endpoints did you create?
- Why did you structure them this way?

4. How did you organize your frontend?

- Where did you put state management?
- How did you structure components?
- How did you handle routing?

5. What would you change?

- What would you improve with more time?
- What problems exist in your code?
- What would be different in a real production app?

Minimum Requirements to Pass

Your submission must have all of these:

- [ ] Dashboard page showing all boards
- [ ] Can create a new board
- [ ] Can click a board to see its tasks
- [ ] Board detail page showing all tasks
- [ ] Can create a new task
- [ ] Can change task status
- [ ] Data saves to database (works after restart)
- [ ] Basic error handling works
- [ ] README with setup instructions
- [ ] AI workflow document
- [ ] Architecture decisions document

Important Notes

Time Management:

- You have exactly 2 hours
- Submit what you have when time is up
- Partial work is acceptable
- Quality is more important than finishing everything

What We Are Testing:

- Can you use AI tools effectively?
- Can you build a full-stack application quickly?
- Can you make good technical decisions?
- Can you write clean, working code?
- Can you explain your thinking?

Environment:

- Your code should run on Mac, Windows, or Linux
- We will test it on Node.js 18 or higher
- If you deploy it, give us the URL

Questions

If you need to ask questions about the requirements, ask before you start the assessment.

Keywords: developer-assessment, react, nextjs, typescript, prisma, ai-assisted-development, cursor, claude-code, antigravity, full-stack, task-management, 2-hour-test