Frontend Engineering Take-Home Assignment

Pre-requisites: docker

Getting Started

1. Unzip the attachment in any directory that you desire
2. Run the following command in the terminal: `docker compose up -d`
3. Run the following command in the terminal: `docker compose exec -it api uv run alembic upgrade head`
4. Ensure that it's working by visiting http://localhost:8000/docs

If there are any issues running the backend server, do reach out to us

Smart To-Do List (Dependency-Aware)

Thank you for taking the time to complete this assignment.
The goal is to assess how you structure a frontend application, manage state, and implement dependency logic.

We care most about your approach, logic, and code quality.

The Challenge

Build a Smart To-Do List web app that models tasks and their dependencies using the backend service that was provided to you. The documentation regarding the api endpoints can be found at http://localhost:8000/docs.

Optional: The backend service also exposes a websocket for real time data. The possible events streamed are:

- task.created
- task.updated
- task.state_changed
- dependency.added
- dependency.removed

You are NOT allowed to use an LLM for code generation and code completion. You may use it ONLY for debugging, searching, etc.

Core Concepts

Dependency Rules (authoritative)

1. **Blocking vs actionable**

- A task is **blocked** if **any** of its dependencies is **not** `'done'`.
- A task is **actionable** if **all** of its dependencies are `'done'`.

2. **Automatic transitions**

- When a task becomes actionable and its current state is `'blocked'`, it must automatically change to `'todo'`.
- When a task becomes blocked (because any dependency is no longer `'done'`), it must automatically change to `'blocked'` (even if it was `'todo'` or `'in_progress'`).

3. **User-driven transitions**

- Users may move an actionable task between `'todo'`, `'in_progress'`, and `'done'`.
- Users **cannot** directly set a task to `'blocked'`; that state is derived.
- Users **cannot** change the status of a task with the `'blocked'` status.

4. **Propagation**

- Updates must propagate **recursively** through all downstream dependents.
- Propagation should occur **immediately** after any state change, until the system reaches a stable state.

5. **Multiple dependencies**

- Handle any number of dependencies per task according to the rules above.

6. **Reasonable simplification**

- If a dependent task is already `'done'` and a dependency reverts (becomes not `'done'`), the dependent must become `'blocked'`. (This keeps the model consistent and is intentional.)

Functional Requirements

Mandatory

1. Display a list of all tasks with their title and current state
2. Filter tasks by state (`'todo'`, `'in_progress'`, `'done'`, `'blocked'`)
3. Implement the dependency rules and recursive propagation described above.
4. Be able to edit task status on the UI.
5. TypeScript is required. Keep code clean, modular, and readable.

Optional Enhancements

- Minimal creation or editing UI for tasks and dependencies.

- Simple cycle detection with a friendly message.
- Error handling. The errors returned by the backend should be shown to the user in a user friendly format

Out of Scope

- Authentication, routing complexity, or pixel-perfect design.

Constraints and Expectations

- You may use a simple component state, Context, or a lightweight store. Library choice is not graded; structure and clarity are.

What We Evaluate

- **Code organization**: Separation of concerns, testable pure logic where possible, appropriate typing.
- **Maintainability**: Readability, naming, minimal coupling, comments where helpful.
- **UI/UX clarity**: It should be easy to see task states and filter them; styling can be minimal.
- **Communication**: Your README explains assumptions, trade-offs, and what you would improve with more time.

Submission

1. Share a repository (GitHub, GitLab, or Bitbucket).
2. Include a **README.md** that covers:
   - How to run the project
   - Your approach to dependency evaluation and propagation
   - Data structures and algorithms used (briefly)
   - Assumptions and trade-offs
   - Potential improvements if given more time
3. Optional: a short video (≤ 3 minutes) walking through the app and logic.