Technical Task for Junior Frontend React TypeScript Engineer

Task Overview

You are required to develop a feature-rich "Task Management Dashboard" for a React application using TypeScript. This task is designed to assess your skills in React, TypeScript, state management, API integration, and advanced UI/UX design.

Feature Description

Create a "Task Management Dashboard" application with the following functionalities:

- Add Task: Users can add a new task with details such as title, description, due date, priority, and tags.
- Edit Task: Users can edit an existing task's details.
- Delete Task: Users can delete a task.
- Toggle Task Completion: Users can mark a task as completed or incomplete.
- Filter and Search Tasks: Users can filter tasks based on their status (All, Completed, Incomplete), priority, and tags. They can also search tasks by title.
- Sort Tasks: Users can sort tasks by due date, priority, or creation date.
- Task Details Modal: Clicking on a task opens a modal with detailed information about the task.
- Drag-and-Drop Reordering: Users can reorder tasks via drag-and-drop.

Technical Requirements

1. State Management

- Use a state management library such as Redux toolkit.
- Create a context or Redux store for managing the dashboard state.

2. UI Components

Create the following components:

- TaskList: Displays the list of tasks.
- TaskItem: Represents a single task with options to edit, delete, and toggle completion.
- TaskForm: A form to add a new task or edit an existing task.
- FilterButtons: Buttons to filter tasks based on their completion status, priority, and tags.
- SearchBar: An input field to search tasks by title.
- SortOptions: Options to sort tasks.
- TaskDetailsModal: A modal to display detailed information about a task.
- DragAndDropContainer: Container to enable drag-and-drop reordering of tasks.

3. API Integration

- Call an API using a service like JSONPlaceholder, Example:
  - https://jsonplaceholder.typicode.com/todos
- Fetch initial tasks from the API on component mount.
- Persist new, updated, and deleted tasks to the mock API.

4. TypeScript

- Define appropriate types for tasks, components, and state.
- Use TypeScript's type checking to ensure type safety across the application.

5. Error & loading state Handling

- Implement robust error handling for API requests.
- Display user-friendly error messages and fallback UI.
- Display user-friendly loading UI.

Bonus Requirements

- Implement a calendar view to visualize tasks by their due dates.
- Use localStorage to persist tasks and user preferences (e.g., theme) between page reloads.
- Add animations for task transitions (e.g., adding, deleting, reordering).
- Implement real-time updates using WebSockets or a similar technology.

Instructions

- Create a Repository: Create a GitHub repository to push your code to it.
- Development: Make meaningful commits as you progress.
- Documentation: Add a README file with instructions on how to set up and run the project. Include any assumptions made and any additional features implemented.
- Submission: Once completed, provide a link to the repository.

Evaluation Criteria

- Code Quality: Clean, readable, and well-documented code.
- TypeScript Usage: Proper usage of TypeScript for type safety.
- State Management: Efficient and scalable state management solution.
- UI/UX: User-friendly, responsive, and visually appealing interface.
- Error Handling: Robust handling of errors and edge cases.
- Testing: Comprehensive test coverage with meaningful tests.
- Bonus Features: Implementation of bonus requirements will be considered.

Resources

- React Documentation
- TypeScript Documentation
- Redux Documentation
- JSONPlaceholder

Additional Information

- Deadline: You have one week from the date you receive this task to complete it.
- Support: If you have any questions or need clarification, feel free to reach out.

Good luck! We are looking forward to seeing your solution.