Junior Frontend Engineer Assessment

Technical Challenge & Evaluation

Project: Build a Personal Task Manager web application with React, Tailwind CSS, and API integration

---

📋 Project Overview

Create a modern, responsive Personal Task Manager that allows users to efficiently manage their daily tasks. This project will assess your React development skills, Tailwind CSS proficiency, API integration capabilities, code organization, and UI/UX design capabilities.

⚡ Core Requirements

Functional Requirements

- **Task Management:** Add, edit, delete, and toggle completion status of tasks via API integration
- **API Integration:** Connect with DummyJSON API for task CRUD operations
- **Task Organization:** Filter by status (All, Active, Completed), search functionality, and sorting options
- **Drag & Drop:** Reorder tasks by dragging them to new positions (mandatory feature)
- **Categories:** Create and assign color-coded categories to tasks
- **Theme Toggle:** Switch between dark and light modes with proper theme persistence
- **Data Persistence:** Use localStorage for categories and theme preferences (tasks managed via API)
- **Error Handling:** Graceful handling of API errors, network issues, and empty states

Technical Stack Requirements

- React with functional components and hooks
- Tailwind CSS for styling
- **API Integration:** DummyJSON API for task data management
- **Drag and Drop:** Task reordering functionality
- Dark/light theme implementation
- Task categories with color coding system
- TypeScript (bonus points)
- Modern JavaScript (ES6+) with async/await

🌐 API Integration Requirements

Your application must integrate with the **DummyJSON API** for task management operations:

Base URL: https://dummyjson.com

Required Endpoints: • GET /todos - Fetch all tasks • GET /todos/{id} - Fetch specific task • POST /todos - Create new task • PUT /todos/{id} - Update existing task • DELETE /todos/{id} - Delete task

API Features: • No authentication required • Realistic mock data responses • Standard REST API patterns • JSON request/response format

Implementation Requirements

- Create a dedicated API service layer for all HTTP operations
- Implement proper error handling for network failures and API errors
- Show loading states during API calls
- Handle edge cases (empty responses, malformed data)
- Use async/await patterns for clean asynchronous code

🏗️ Project Structure

- Follow best practices

✨ UI/UX Design Requirements

Visual Design Standards

- Clean, modern aesthetic with proper typography hierarchy
- Consistent color palette and spacing system
- Dark and light theme implementations with smooth transitions
- Color-coded task categories with intuitive visual hierarchy
- Smooth animations and micro-interactions
- Visual feedback for drag and drop operations
- Loading states and skeleton screens during API calls
- Clear visual feedback for user actions and API responses

Responsive Breakpoints

- **Mobile:** 320px and up (mobile-first approach)
- **Tablet:** 768px and up
- **Desktop:** 1024px and up

Accessibility Requirements

- Proper color contrast ratios (WCAG AA compliance)
- Keyboard navigation support
- Screen reader friendly markup
- Focus indicators for interactive elements

🌟 Bonus Features (Optional)

- Due dates with calendar integration
- Keyboard shortcuts for power users
- Export tasks to JSON/CSV functionality
- Task completion statistics and analytics

⏱️ Time Allocation

Recommended: 2 days | **Maximum:** 3 days

The additional features require more time - focus on core functionality first, then implement drag & drop, themes, and categories

---

📦 Deliverables

- **GitHub Repository:** Public repo with clean commit history and meaningful messages
- **Live Demo:** Deployed application (Vercel, Netlify, or similar)
- **Documentation:** Comprehensive README with setup instructions and features overview
- **Screenshots:** Visual demonstration of the application in action

---

📝 README Requirements

- Project description and key features
- Installation and setup instructions
- Technology stack and dependencies
- Screenshots or GIF demonstrations
- Known limitations and future enhancements
- Personal reflection on development process

👥 User Stories

As a user, I want to:

- Quickly add new tasks via API integration and organize them into color-coded categories
- View all tasks fetched from the API in a clean, scannable list with drag and drop reordering
- Mark tasks complete with satisfying interactions that sync with the API
- Find specific tasks using search and category filtering
- See loading states when the app is communicating with the API

- Switch between dark and light themes based on my preference
- Use the app seamlessly on mobile and desktop with responsive design
- Have my categories and theme preference persist locally while tasks sync with the API

🏆 Success Criteria

A successful submission demonstrates:

- **Technical Excellence:** Clean, maintainable React code with proper hooks usage and API integration
- **API Proficiency:** Robust error handling, loading states, and asynchronous operations
- **Design Sensibility:** Modern, professional interface that feels polished
- **Interactive Features:** Smooth drag & drop functionality with visual feedback
- **User-Centric Thinking:** Intuitive interactions and thoughtful user experience
- **Problem-Solving:** Elegant handling of network errors, edge cases, and loading states
- **Attention to Detail:** Consistent styling, smooth animations, and accessibility

---

📤 Submission Process

1. Create a public GitHub repository
2. Deploy your application to a free hosting platform
3. Ensure cross-browser compatibility (Chrome, Firefox, Safari)
4. Write comprehensive setup instructions
5. Submit both repository URL and live demo link
