QA Analysis for Tanstack Query & Feature Implementation

Report for Quality Assurance

Executive Summary

This pull request introduces a major architectural overhaul by integrating Tanstack Query for server state management and Redux Toolkit for client state. It also delivers significant new features, including a task creation form, a task listing view, and a CSV data export function. The UI has been refactored from standard CSS to SCSS, with numerous new components being added. Due to the breadth and depth of these changes, a thorough, multi-layered testing approach is critical to ensure application stability, data integrity, and a seamless user experience.

Testing Strategy Overview

The testing strategy must focus on three core areas: the new data-fetching layer (Tanstack Query), the state management logic (Redux), and the new task management user flow (Add/List/Export). We will employ a combination of component-level testing for the new UI elements, integration testing to validate the interaction between the frontend and backend APIs, and end-to-end testing for the critical user paths. Special attention must be paid to caching behavior, loading states, and error handling introduced by Tanstack Query.

Functional Testing Requirements

Functional testing will verify that all new features operate according to specifications.

- Task Creation Form (`AddForm`): Validate all input fields, including required fields (title, start time), date/time constraints (start before end), and successful submission logic. Verify correct payload construction and API calls.
- Task Listing (`HomePageContent`): Test the data fetching lifecycle: loading state, successful data rendering, and error state. Ensure tasks are correctly grouped and

displayed. Verify duration calculation and formatting.

- CSV Download: Confirm that clicking the download button triggers a GET request to the correct endpoint and initiates a file download. The downloaded file should be named correctly and contain valid data.
- UI Component Validation: Each new component (`BaseInput`, `Button`, `TaskTable`, `TaskItem`) must be tested in isolation to ensure it renders correctly and handles user interactions as expected.
- Navigation: Verify that navigation between the home page and the new `/add` page functions correctly.

Non-functional Testing Considerations

Beyond core functionality, we must assess performance, security, and usability.

- Performance Testing: Analyze the caching strategy of Tanstack Query to ensure it effectively reduces redundant API calls. Measure the initial load time of the task list page and the responsiveness of the task creation form.
- Security Testing: The hardcoded `user_id: 2` in both task creation and CSV download is a critical vulnerability (Insecure Direct Object Reference). This must be flagged and tested to ensure users cannot access or create data for other users. All form inputs must be validated on the backend to prevent injection attacks.
- Usability & UX: Assess the clarity and helpfulness of validation messages, loading indicators, and error toasts. Ensure the application provides clear feedback to the user during asynchronous operations.
- Cross-Browser Compatibility: Test the new UI components and layouts on major browsers (Chrome, Firefox, Safari, Edge) to ensure consistent rendering and functionality.

Test Environment Setup Needs

A stable test environment is required to execute the test plan effectively.

- Backend API: A deployed instance of the backend service (`http://localhost:3000/api/v1/...`) is required. The database should be seeded with a diverse set of task data for multiple users to test fetching, grouping, and security.
- Network Simulation: Use browser developer tools to simulate various network conditions (e.g., slow 3G, offline) to test Tanstack Query's caching, retries, and offline behavior.

- Developer Tools: Require testers to have access to React Developer Tools and Redux DevTools to inspect component state, props, and the Redux store during test execution.

Risk-based Testing Prioritization

Testing efforts should be prioritized based on the risk and impact of potential failures.

- High Priority: The entire task creation flow, including Formik validation and Tanstack Query mutation. The CSV download functionality, focusing on security and data integrity. Data fetching and caching logic for the task list.
- Medium Priority: Correct rendering and grouping of tasks on the home page. Responsiveness and styling of all new components across different screen sizes.
- Low Priority: Static content rendering and basic component appearance. The removal of old CSS files.

Recommended Test Scenarios

- Verify submitting the 'Add Task' form with an empty 'Title' field displays the 'Title is required' error message.
- Verify submitting the 'Add Task' form with a 'Start Time' that is later than the 'End Time' displays the 'Start time have to be before end time' error.
- Successfully fill out and submit the 'Add Task' form and verify that the form fields are cleared and a POST request is sent to the `/api/v1/tasks` endpoint with the correct payload.
- Simulate a 500 server error on task submission and confirm that a user-friendly 'Error' toast message is displayed.
- On the home page, verify a 'Loading...' message is displayed while the task data is being fetched.
- After tasks are fetched successfully, verify they are grouped correctly by date (or other key) and that the task title and formatted duration (e.g., '1h 30 min') are displayed.
- Simulate a network error when fetching the task list and verify that an 'Error' message is shown to the user.
- Click the 'Download CSV' button and confirm it initiates a file download with the name format '{userId}-evidence.csv'.

- Test the 'Add Task' form by entering a description containing special characters (`