**Test Plan: Task Manager Web Application**

**1. Introduction**

This document outlines the test strategy and plan for the Task Manager web application, which consists of a React frontend and a Node.js backend API. The primary goal of this testing effort is to ensure the application's core functionalities, including user authentication and task management (CRUD operations), work as expected and are robust against various inputs.

**2. Scope of Testing**

The testing will cover the following key areas:

**2.1. Functional UI Automation (React Frontend)**

This involves testing the user interface interactions and end-to-end user flows directly through the browser. All key functional scenarios, including user authentication (login/logout, valid/invalid credentials), task management (creating, editing, deleting, toggling completion), and data validation (presence of expected data, display of success/error messages), are thoroughly covered.

**2.2. API Test Automation (Node.js Backend)**

This focuses on validating the backend API endpoints directly, ensuring they handle requests and responses correctly, independently of the UI. This includes comprehensive testing of the /login endpoint (positive and negative scenarios), and all CRUD operations for tasks (GET /items, POST /items, PUT /items/:id, DELETE /items/:id), with both positive and negative test cases. Authorization mechanisms (JWT token validation) are also fully covered.

**3. Tools Used and Why**

* **Selenium WebDriver (with Java):**
  + **Why:** Chosen for UI automation due to its industry-standard status, robust capabilities for simulating real user interactions, and broad community support. It effectively covers all defined UI automation scenarios.
* **Postman/Newman:**
  + **Why:** Postman is an excellent tool for manually testing APIs and organizing API requests into collections. Its scripting capabilities allow for powerful assertions and dynamic data handling. Newman, as Postman's command-line runner, enables seamless integration into CI/CD pipelines for automated API testing, covering all defined API test scenarios.
* **Node.js (Express.js):**
  + **Why:** Used for the backend API due to its popularity, efficiency, and ease of setting up RESTful services.
* **React:**
  + **Why:** Used for the frontend due to its component-based architecture, declarative UI, and strong community support, making it ideal for building single-page applications.

**4. How to Run the Tests**

**4.1. Prerequisites**

Before running any tests, ensure the following are set up:

1. **Node.js Backend:**
   * Navigate to the backend directory.
   * Install dependencies: npm install express body-parser cors jsonwebtoken
   * Start the server: node server.js (should run on http://localhost:3001)
2. **React Frontend:**
   * Navigate to the frontend directory.
   * Install dependencies: npm install (and npm install -D tailwindcss postcss autoprefixer if setting up Tailwind)
   * Start the development server: npm start (should run on http://localhost:3000)
3. **Java & Selenium:**
   * Install Java Development Kit (JDK).
   * Set up a Maven or Gradle project for your Selenium tests.
   * Add Selenium WebDriver dependencies to your pom.xml (Maven) or build.gradle (Gradle) file.
   * **ChromeDriver:** Download the appropriate ChromeDriver version for your Google Chrome browser from <https://chromedriver.chromium.org/downloads>. Place the chromedriver executable in a directory that's included in your system's PATH environment variable.
4. **Postman & Newman:**
   * Install Postman Desktop App from <https://www.postman.com/downloads/>.
   * Install Newman globally: npm install -g newman

**4.2. Running UI Automation Tests (Selenium)**

1. Navigate to your Selenium Java project directory (e.g., ui\_tests).
2. Compile and run your tests using your build tool (e.g., mvn test for Maven, or execute your test runner from your IDE).
3. Observe the browser automating interactions and test results in your console.

**4.3. Running API Automation Tests (Postman/Newman)**

1. **In Postman:**
   * Manually create the "Task Manager API Tests" collection as detailed in the API Test Automation section, including all requests, headers, bodies, and test scripts.
   * Ensure the authToken and createdTaskId environment variables are set up correctly by running the "POST /login" and "POST /items" requests manually at least once within Postman to populate them.
   * Export the collection: Click "..." next to the collection name -> "Export" -> "Collection v2.1 (recommended)" and save it as task\_manager\_api\_tests.json.
2. **Using Newman (CLI):**
   * Open your terminal and navigate to the directory where you saved task\_manager\_api\_tests.json.
   * Run the collection: newman run task\_manager\_api\_tests.json
   * Newman will execute all requests in the collection and provide a summary of passed/failed tests in the terminal.

**5. Assumptions and Limitations**

**5.1. Assumptions**

* The Node.js backend runs on http://localhost:3001.
* The React frontend runs on http://localhost:3000.
* The application's initial state (in-memory data) is clean for each test run (Selenium clears local storage, Postman tests are designed to be idempotent where possible or depend on prior steps).
* Network connectivity is stable between the test runner and the application.
* Browser (Chrome) and ChromeDriver are compatible versions.

**5.2. Limitations**

* **In-Memory Data:** The backend uses in-memory data storage, meaning data is lost when the server restarts. This simplifies setup but is not suitable for production and impacts test repeatability if the server is frequently restarted between test runs. For robust testing, a persistent database would be required.
* **Error Handling:** Basic error handling is present; more comprehensive error scenarios (e.g., network failures, server crashes) are not explicitly covered by these tests.
* **Performance Testing:** This test plan does not include performance, load, or stress testing.
* **Security Testing:** Beyond basic authentication/authorization checks, in-depth security testing (e.g., penetration testing, vulnerability scanning) is not covered.
* **Cross-Browser/Device Testing:** Selenium tests are configured for Chrome only. Comprehensive testing would require running tests across multiple browsers and device types.
* **Postman Collection Generation:** The Postman collection file (.json) is not directly generated by this document; it requires manual creation within the Postman application based on the provided request details.
* **Test Data Management:** Test data is hardcoded in some places (e.g., login credentials, task titles). For larger applications, a dedicated test data management strategy would be beneficial.