Test Plan – TaskNexus web application MERN/REDUX/AWS-S3

Prepared by: Ramzi Chahbani

Date: April 14, 2025

Table of Contents

Overview

TaskNexus is a full-stack web application built using the **MERN stack (MongoDB, Express.js, React, Node.js), enhanced with Redux for state management and AWS S3 for secure media storage.** It enables users to manage their daily tasks through a secure and intuitive interface. Key features include the ability to **create, read, update, delete, and mark tasks as complete**, as well as **account deletion** and **personal information modification** such as changing the username, name, email and profile photo.

The app implements **two authentication mechanisms**:

- **JWT-based authentication**, allowing users to register and log in using an email and password.
- **Google OAuth 2.0**, enabling seamless third-party sign-in.

This test plan outlines the quality assurance strategy for validating the functional and non-functional requirements of the TaskNexus application. It includes **test objectives, scope, environments, roles and responsibilities, schedule, tools, risks, and deliverables.**

Testing efforts are conducted by a **solo Dev-Test engineer**, focusing on both **manual and automated testing practices**. Special attention is given to:

- Verifying authentication flows (JWT and Google OAuth)
- Validating task management operations
- Ensuring API response consistency
- Testing UI behavior and responsiveness
- Checking input validation and error handling
- Preserving data integrity and security

The goal of this testing process is to ensure the application delivers a **stable, user-friendly, and production-ready experience** with minimal defects and meeting quality standards suitable for a production-deployable product.

This plan serves as the foundation for all QA-related activities and aligns with the application's objective to deliver a reliable and scalable user experience.

Objectives

The primary objectives of testing the **TaskNexus** application are to ensure that all core features function as intended, the user experience is consistent and secure, and the system performs reliably under normal conditions.

Functional

- Verify the functionality of user authentication using **JWT** and **Google OAuth**.
- Ensure users can **create, read, update, delete, and mark tasks as complete**.
- Confirm that **account management features** (updating profile information and deleting accounts) work as expected.
- Validate **API responses**, including appropriate **status codes**, response structures, and **error messages**.
- Confirm that **protected routes and sensitive operations** are accessible only to authenticated users.

Non-Functional

- Assess **UI responsiveness** and compatibility across various devices and screen sizes.
- Validate **input handling and error feedback** to prevent invalid, malicious, or incomplete data submission.
- Ensure the application maintains **consistent performance and acceptable load times** under normal usage.
- Verify **secure media handling**, including file upload, and storage via **AWS S3**.
- Confirm the overall user experience is **stable, intuitive, accessible**, and free from major usability issues.

Scope

Features to be tested

Module name	Description
User Registration/Login (JWT)	Test the standard registration and login flow using email and password. Validate input, authentication tokens, error handling, and form validation.
Google OAuth Authentication	Verify third-party login and registration integration via Google OAuth 2.0. Ensure correct handling of user sessions, account creation, and error cases.
Task Management (CRUD)	Test creation, retrieval, editing, and deletion of tasks. Verify correct data persistence, error messages, and task status handling.
Mark Task as Complete	Validate that users can toggle a task's completion status. Ensure UI and data reflect the correct state consistently.
Update User Profile	Test the ability to update personal information: username, full name, email, and profile photo. Verify validation and persistence of changes.
Delete User Account	Ensure users can delete their account and all associated data is removed securely. Confirm confirmation prompts and irreversible deletion.
API Response Validation	Validate correct HTTP status codes, messages, and error responses for both success and failure scenarios.
UI Responsiveness	Test the layout and usability of the app across multiple devices and screen sizes. Ensure the UI adapts properly to different viewports.
Route Protection	Verify that unauthorized users cannot access

	protected routes and that redirection or error messages behave as expected.

Inclusions

1. Manual testing of all major features
    1.1. Login with JWT/Google Oauth
    1.2. Signup
    1.3. Logout
    1.4. Task Creation, retrieval, update and deletion
    1.5. Account Deletion
    1.6. Personal information update (username, name, email and profile photo)
2. Automated testing of authentication and task CRUD API endpoints
3. UI testing
4. Cross-browser testing for Chrome and Firefox
5. Responsive testing on desktop and mobile breakpoints
6. Positive and negative test cases
7. Performance testing
8. Load testing
9. Stress testing

Exclusions

1. Legacy browser support

Test Environments

System Configuration (Infrastructure)

Component	Details
Frontend	Not deployed yet – currently tested on http://localhost:5173 (using vite)
Backend API	Not deployed yet – currently running on http://localhost:4000/api (if 4000 is the port specified in the .env file)
Database	MongoDB (local instance or Atlas test cluster)
Auth Services	JWT + Google OAuth 2.0
Testing Tools	Postman, Jest (unit/integration tests), Playwright (E2E tests) and Chrome DevTools

Execution Environment

Platform Type	Device/OS	Browser	Purpose
Desktop	Windows 11	Chrome (latest), Firefox and Edge	Primary development & testing platform
Desktop	macOS Catalina	Chrome (latest), Firefox	Secondary development & testing platform
Mobile	Android 14	Chrome Mobile	Mobile responsiveness testing

Test Strategy

Type	Tool	Notes
Unit testing	Jest	Backend controllers
Integration testing	Jest/Supertest	API & DB
End to end testing	Playwright	Full flow testing
Manual	Chrome DevTools and Postman	Mobile view, bugs and API responses

CI/CD Integration

A GitHub Actions CI pipeline will be configured to run on every pull and push request to the main branch. The pipeline includes:

- **Unit tests** using **Jest**
- **Integration tests** using **Jest** and **Supertest**
- **End-to-end testing** using **Playwright** in headless mode

Test Reporting

Test execution reports will be generated using **Allure** to provide rich, interactive dashboards for stakeholders and developers. The Allure report will include:

- Test case execution results (pass/fail)
- Attached screenshots and logs for failed cases
- Step-by-step breakdowns of automated test flows
- Metrics such as duration and status trends

Allure will be integrated into the CI pipeline and triggered after each automated test suite execution.

Defect Reporting Procedure

Field	Description
ID	Unique identifier that identifies the defect
Summary	A short, descriptive title of the defect
Steps to Reproduce	Detailed steps to reliably reproduce the defect
Expected Result	What should happen
Actual Result	What happened
Severity	Level of impact (Critical/Major/Minor)
Priority	Urgency of fixing the defect (High/Medium/Low)
Reported By	Name or ID of the reporter
Date reported	Date when the defect was reported
Attachments	Screenshots, logs, or screen recordings as evidence
Comments/Notes	Any extra observations or clarifications

Test Schedule

Task	Date range
Unit + Integration	Apr 15–18
E2E Testing	Apr 19–21
Manual QA	Apr 22
Fix + Retest	Apr 23
Final Review	Apr 24

Test Deliverables

- Test case document (manual + automated)
- Test execution summary report
- Bug/defect tracker
- Final QA sign-off report

Risk & Mitigation

Risk	Impact	Mitigation
Mobile responsiveness issues	Medium	Test on emulators and real devices, use responsive design best practices
JWT token tampering or expiry issues	High	Validate JWT securely and use refresh token strategy
Missing or unclear error messages	Low	QA to validate all user-facing messages for clarity and helpfulness
Uncaught edge cases (e.g., duplicate emails)	Medium	Write negative test cases and validation checks on both frontend/backend

Rate-limiting not enforced	Medium	Implement backend throttling and test with high request volumes

Exit Criteria

- All major/critical bugs resolved
- 100% of test cases passed
- All core features covered by tests
- Performance acceptable under expected load
- QA sign-off completed