**Test Plan Strategy for UI & API Automation**

**1. Overview**

This document presents the strategy and scope for automated testing of the web application with React frontend and Node.js backend API. The focus is on functional UI tests and REST API tests to ensure core features work as expected, including authentication and CRUD operations on items.

**2. What is Being Tested**

**UI Tests**

* User authentication (login) with valid and invalid credentials.
* Item management functionality:
  + Creating a new item
  + Editing an existing item
  + Deleting an item
* Verification of expected data presence after each action.
* Visual feedback via screenshots captured on test failure.

**API Tests**

* Authentication API (POST /login) with positive and negative scenarios.
* CRUD APIs for items:
  + Retrieve items (GET /items)
  + Create items (POST /items)
  + Update items (PUT /items/:id)
  + Delete items (DELETE /items/:id)
* Error handling for invalid requests, missing authentication, and invalid data.

**3. Test Coverage Areas**

| **Area** | **Positive Tests** | **Negative Tests** |
| --- | --- | --- |
| **UI - Login** | Login with valid username/password | Login with invalid credentials, missing fields |
| **UI - Items** | Create, edit, delete items, assert UI updates | Attempt actions with invalid data or no login |
| **API - Authentication** | Valid login returns token | Invalid login, missing fields, invalid token |
| **API - Items** | CRUD operations with valid token | CRUD with invalid IDs, missing fields, no token |

**4. Types of Testing**

* **Functional Testing:** Validates core user flows and API endpoints behave correctly.
* **Positive Testing:** Ensures system works as expected with valid input.
* **Negative Testing:** Verifies system robustness against invalid or missing data.
* **Security Testing:** Basic auth token validation for protected endpoints.
* **Regression Testing:** Automated suites can be run on each build to prevent regressions.
* **UI Automation:** End-to-end tests simulate user interaction on the frontend.
* **API Automation:** Direct backend API testing via Postman collection.

**5. Tools Used and Why**

| **Tool** | **Purpose** | **Reason for Choice** |
| --- | --- | --- |
| **Selenium WebDriver (Java)** | UI Automation | Industry-standard for browser automation; Java provides strong typing and ecosystem support. |
| **TestNG** | Test Framework & Assertions | Enables organized test cases, grouping, and reporting. |
| **Maven** | Build and Dependency Management | Simplifies build lifecycle and dependency resolution. |
| **Postman** | API Testing & Automation | User-friendly interface to create, run, and automate API tests; supports scripting. |
| **Newman** | Command-line Runner for Postman | Enables running Postman collections in CI pipelines or terminal. |
| **Git/GitHub** | Source Control | Version control and collaboration. |
| **Jenkins / GitHub Actions (optional)** | CI/CD Pipeline | Automate test execution on commits/pull requests. |

**6. How to Run the Tests**

**UI Tests**

1. Ensure Java (JDK 8+) and Maven installed.
2. Clone the repository.
3. Run tests via command line: mvn clean test
4. Test reports and screenshots on failure will be saved in the target folder.

**API Tests**

1. Import the provided Postman collection JSON into Postman.
2. Set environment variables baseUrl (e.g., http://localhost:3000/api) and others as needed.
3. Run collection or individual requests.
4. Alternatively, run with Newman CLI:

newman run ui-app-api-tests.postman\_collection.json -e environment.json

**7. Assumptions and Limitations**

* The backend API is stable and follows REST conventions.
* Authentication token returned on login is valid and required for protected routes.
* UI elements have consistent IDs or selectors for Selenium to interact with.
* Tests assume the database starts in a known state or supports test data cleanup.
* Screenshots are captured only on UI test failures.
* Tests do not cover performance, load, or security beyond token auth.
* The Postman collection requires environment variables to be set manually or via CI.
* Tests require a running instance of the app backend and frontend during execution.