Test Strategy Document

Project: UI_Api_Automation_Framework_Swati

Prepared By: Swati Malviya

Date: 28 July 2025

1. Objective: -

To verify the functional correctness of a full-stack web application built with a React frontend and a Node.js backend API.

2. Scope of Testing: -

This strategy covers the following areas:

- **UI Functional Testing** using Selenium WebDriver:

    - Login with valid/invalid credentials

    - Create, edit, and delete a Todo item

    - Validation of UI messages and behavior after each action

- **API Testing** using REST-assured:

    - Login endpoint authentication

    - Full CRUD operations on Todo items via `/todos` endpoint

    - Positive and negative test cases

3. Tools and Frameworks: -

Selenium WebDriver:
Used for automating UI functional tests on the React frontend. It provides cross-browser support and integration with TestNG for structured test execution.

REST-assured:
Selected for API automation of the Node.js backend. It simplifies HTTP requests and JSON validation, supporting both positive and negative test cases.

TestNG
Employed as the test framework for structuring tests, enabling annotations, grouping, parallel execution, and generating test reports.

Maven
Chosen as the build tool for managing project dependencies, plugins, and test execution lifecycle.

Chrome Driver (Selenium)
Used to execute browser-based tests in Chrome; supports both headed and headless modes.

GitHub Actions (CI/CD)
Configured to run UI and API tests automatically on push and pull requests, ensuring consistent test coverage in CI environments.

Custom Utilities

- DriverFactory: Manages WebDriver instance creation with config-driven browser and headless options.
- ConfigReader & TestConfigReader: Loads environment-specific settings and test parameters from. properties files.
- ScreenshotUtil: Captures screenshots automatically on test failure for debugging.

4. Test Coverage: -

UI Tests (Selenium)

The automated tests cover the following areas:

UI Functional Tests (Selenium)

- **Login**
    - Valid login with correct credentials
    - Invalid login with incorrect credentials
- **Todo Management**
    - Create a new Todo item and verify its presence
    - Edit an existing Todo item and validate the update
    - Delete a Todo item and confirm removal
- **Validation**
    - Ensure UI error/success messages are displayed correctly after each action

- Verify the consistency of data displayed in the UI with backend updates

API Tests (REST-assured)

Authentication

- Positive test: valid login returns token
- Negative test: invalid credentials result in 401 Unauthorized

Todo CRUD Operations

- **GET /todos**: Validate authorized fetch and unauthorized access
- **POST /todos**: Create a Todo with valid payload; reject invalid payload
- **PUT /todos/:id**: Update with valid ID; verify failure with invalid ID
- **DELETE /todos/:id**: Delete with valid ID; verify failure with invalid ID

Data Validation

- Confirm that created/updated/deleted items reflect correctly in subsequent GET requests

Non-Functional Aspects

- Basic validation of API response times to ensure performance thresholds
- Headless browser execution supported for faster CI runs

5. How to Run the Tests: -

Prerequisites:

- Chrome + Chrome Driver installed
- MongoDB running or using MongoDB Atlas
- Application (React + Node.js) running locally on:
- Frontend: `http://localhost:3000`
- Backend: `http://localhost:5000/api`

Execute Tests:

UI Tests:

- Command: mvn test -DsuiteXmlFile=src/test/resources/testng.xml
- Runs the Selenium WebDriver suite against the React frontend.
- Validates login (valid/invalid), create, edit, and delete flows.
- Captures screenshots automatically for failed test cases.

API Tests (run with TestNG suite):

- Command: mvn test
- Runs the REST-assured test suite against the Node.js backend API.
- Covers login, and full CRUD operations (GET /todos, POST /todos, PUT /todos/:id, DELETE /todos/:id).
- Includes both positive and negative test scenarios.

Reports

- TestNG HTML reports are generated under ./test-output.
- Screenshots of failed UI tests are saved in ./test-output/screenshots.
- In CI, reports and logs can be exported as artifacts for review.

6. Assumptions

- The backend must be running and seeded with a valid test user (`test@example.com / password123`)
- The UI locators used (IDs like `login-button`, `new-todo`, etc.) are stable
- Chrome is the default browser under test
- Auth tokens are assumed to be valid for API access during test execution

7. Limitations

- Cross-browser testing not covered (limited to Chrome)
- No performance, accessibility, or visual regression testing included
- No mocking/stubbing of backend APIs or DB
- No test data cleanup in DB (CRUD tests rely on order of execution)
- Test user credentials are hardcoded in config files

8. CI Integration

- A GitHub Actions workflow (`.github/workflows/ci.yml`) is included to automate test execution on every push or pull request to the `main` branch.

Pipeline Steps:

- Checkout code
- Set up JDK 11 and Chrome
- Cache Maven dependencies

Run the Maven TestNG test suite:

-
  ```
  mvn test -DsuiteXmlFile=src/test/resources/testng.xml
  ```