CLEAN CITY TEST PLAN PROJECT

PROJECT OVERVIEW

The purpose of this document is to define the test plan for the CleanCity Waste Pickup Scheduler web application. The system enables citizens to schedule waste pickup requests, track their status, and provide feedback, while administrators can manage requests and user data. This test plan follows the IEEE 829 standard and defines objectives, scope, approach, resources, and schedules for all testing activities.

TEST OBJECTIVES

To validate that users can successfully register, log in, schedule waste pickups, and submit feedback.
To confirm that pickup requests are stored, retrieved, and updated accurately in LocalStorage.
To ensure that all client-side form validations (registration, pickup, feedback) function correctly under both valid and invalid inputs.
To verify that the Admin Panel correctly supports administrative operations, including:

- Viewing, filtering, and updating pickup request statuses.
- Managing registered users (add, edit, delete) securely.
- Reviewing and analyzing feedback data.
- Generating and viewing operational statistics (e.g., total, pending, and completed pickups).

To assess access control by verifying that non-admin users cannot access admin functionalities.
To evaluate overall system performance, usability, and error handling during various workflows.
To ensure data persistence and correct synchronization between the user and admin modules.

SCOPE

In Scope

The following features and components are included in the scope of testing for the CleanCity Waste Management System:
User Portal

a) User Registration & Authentication
   - ✓ Account creation, login, logout, and session validation.

b) Pickup Request Management
   - ✓ Creating, viewing, editing, and deleting waste pickup requests.
   - ✓ Validation of input fields (date, location, waste type).

c) Feedback Submission
   - ✓ Submitting service feedback and verifying data storage.

d) Dashboard Functionality
   - ✓ Display of request history and real-time status updates.

e) Client-Side Validation & Error Handling
   - ✓ Field validation, required field checks, invalid input alerts, and confirmation messages.

f) LocalStorage / Data Persistence Testing
   - ✓ Ensuring pickup requests, user sessions, and feedback data are retained after page reloads.

g) Cross-Browser Compatibility (Functional UI)

✓ Chrome, Firefox, and Edge.

h) Mobile Responsiveness Testing
   ✓ Layout and usability verification on smaller screen sizes.

Admin Panel

a) Authentication & Access Control
   ✓ Only authorized admin users can log in and access admin features.
b) Request Management
   ✓ Viewing, filtering, and updating pickup request statuses (Pending → Scheduled → Completed → Missed).
c) User Management
   ✓ Adding, editing, and deleting user accounts securely.
d) Feedback Review and Analytics
   ✓ Viewing all feedback submissions and analyzing satisfaction levels.
e) Data Visualization and Reports
   ✓ Testing generation and export of summary statistics (e.g., number of completed pickups, performance metrics).
f) Admin Interface Usability
   ✓ Ensuring admin navigation, sorting, and filtering work intuitively and without errors.
g) Functional & Regression Testing using Selenium
   ✓ Verifying that critical admin features remain functional after updates or fixes.

Out of Scope

The following areas are excluded from the current test plan and will not be tested as part of this cycle:

a) Back-End Server or Database Integration
   ✓ The current system uses LocalStorage; no real database or API server integration is tested.
b) Payment Gateways or Billing Systems
   ✓ There are no financial transactions or integrations to be validated.
c) Email or SMS Notifications
   ✓ Notification mechanisms are not implemented in this version.
d) Third-Party API Integrations (e.g., Maps, GIS)
   ✓ Any integrations beyond the app's internal data are excluded.
e) Advanced Security or Penetration Testing
   ✓ Testing for SQL injection, XSS, and similar exploits is out of this project's current scope.
f) Performance Load and Stress Testing
   ✓ High-load or concurrency testing is not required for this version.
g) Accessibility Compliance (WCAG / ADA)
   ✓ Accessibility features are not formally tested in this iteration.
h) Offline / PWA Capabilities
   ✓ The app is not tested for offline functionality or service worker performance.

<u>TEST ITEMS</u>

The primary components to be tested include:

- Frontend: HTML, CSS, and JavaScript user interface.
- Business logic: dataService.js i.e. handles localStorage CRUD operations.

- Integration: form submissions, navigation, and feedback system.
- Admin module: user management and request status updates.

TESTING APPROACH

The testing strategy combines manual and automated methods:

Manual Testing:
-Exploratory testing for UI layout, navigation, and form validation.
- Usability and error message consistency checks.

Manual test cases:

TESTID	TEST CASE	INPUT	EXPECTED
mT001	Register new user	Enter valid details	Registration successful
mT002	Register duplicate user	Existing email	Error displayed
mT003	Login valid user	Correct credentials	Dashboard loads
mT004	Login invalid user	Wrong password	Error message
mT005	Submit pickup request	All fields valid	Request saved
mT006	Submit incomplete request	Missing field	Validation error
mT007	Filter requests	Filter: Completed	Filtered correctly
mT008	Feedback submission valid	All fields correct	Feedback stored
mT009	Feedback missing comment	Leave blank	Error displayed
mT010	Admin update status	Change Pending to Completed	Status updated
mT011	Unauthorized admin access	User opens admin.html	Redirected to login
mT012	Persistent storage	Reload page	Data retained
mT013			

Automated Testing:
- Unit Testing and integration testing: Performed using Jest and jsdom for dataService.js functions.
- Integration & E2E Testing: Executed using Cypress to simulate user workflows (workflow automated testing)
- Cross-Browser Functional Testing: Conducted via Selenium WebDriver on Chrome and Firefox.
Positive and negative test cases will be included for each test type.

- Using ChromeDevTools: for Performance analysis and localStorage monitoring

- Github issues: To track bugs and reporting.

Automated test cases:

<u>Jest & jsdom Unit & Integration Tests</u>

- Test initializeData() - creates default datasets if empty.
- Test addUser() - successfully adds and prevents duplicates.
- Test getUserByEmail() - retrieves correct user or returns undefined.
- Test addPickupRequest() - generates valid IDs and default status.

- Test updateRequestStatus() - correctly changes status.
- Negative tests - missing fields, invalid data, and duplicate IDs.

Cypress E2E Tests

Cypress automates full browser flows to ensure end-user features work correctly.
Scenarios:

- Register - Login - Logout flow.
- Create and display a pickup request.
- Filter requests by location and status.
- Submit feedback successfully.
- Admin updates request status.
- Verify error handling for invalid inputs.

Selenium WebDriver Tests
Selenium validates that CleanCity functions consistently across browsers (Chrome, Firefox, Edge).
Automated Scenarios:

- Cross-browser registration form submission.
- Login and dashboard navigation check.
- Verify visibility of feedback confirmation messages.
- Check filter dropdown functionality in multiple browsers.
- UI validation for element presence and colours.

PASS & FAIL CRITERIA

A test passes if the actual outcome matches the expected result defined in the test case. Failure occurs when discrepancies exist, including missing feedback, incorrect status updates, or persistence errors.

SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

Testing will be suspended if critical blocking defects (e.g., data not saving or app not loading) prevent further testing. Testing will resume once the issue has been resolved and verified by regression testing.

TEST DELIVERABLES

- Test Plan (this document)
- Test Cases and Scripts (Jest, Cypress, Selenium)
- Test Data Sets
- Test Execution Reports
- Defect Reports (via GitHub Issues)
- Final Test Summary Report

TESTING TASKS

1. Prepare test environment and data.
2. Execute Jest unit tests for dataService.js.

3. Perform Cypress E2E tests for registration, login, requests, and feedback.
4. Conduct Selenium cross-browser tests.
5. Run manual UI and usability tests.
6. Log defects and retest fixes.

ENVIRONMENT WHERE THE TESTS WILL BE CARRIED OUT

Hardware: Windows 10+
Software:
- Node.js v18+
- npm
- Browsers: Chrome, Firefox
- Test Frameworks: Jest, Cypress, Selenium WebDriver
- Local web server (e.g., serve, live-server)

RISKS AND CONTIGENCIES

Risks:
- LocalStorage limitations affecting test repeatability.
- Browser version incompatibility.
- Test data corruption between runs.

Contingencies:
- Use mock data for repeatable unit tests.
- Isolate Cypress environment per test.
- Maintain browser version control in CI/CD.

DELIVERABLES

1. Jest test coverage report
2. Cypress E2E automation scripts
3. Selenium cross-browser test suite
4. Manual test case execution sheet
5. Bug report and summary for GitHub Issues

APPROVALS:

PREPARED BY: KENEDY AMBILA – TEST MANAGER

REVIEVED BY: DANIEL MUSEMBI – RISK ANALYST

REVIEWED BY: JULIET BONARERI – TEST EXECUTOR