Software Test Report

CleanCity: Waste Pickup Scheduler Web Application v1.0

Document ID: TR-2025-CleanCity-001

Date of Report: November 19, 2025

Testing Period: November 5 - November 19, 2025

Prepared by:

- **Test Manager:** Shanice Chepkwony
- **Risk Analyst:** Mark Mwangi Gaituri
- **Test Executor:** Ian Macharia

Document Version: 1.0

Institution: PLP Academy

Executive Summary

This report presents the results of comprehensive quality assurance testing conducted on the CleanCity waste pickup scheduler web application version 1.0 from November 5 to November 18, 2025. The testing focused on validating functional requirements, identifying security vulnerabilities, ensuring cross-browser compatibility, and assessing code quality through static analysis.

Key Findings:

- **14 test cases executed** with **36% pass rate** (5 passed, 8 failed, 1 deferred)
- **10+ critical and high-severity defects identified** across authentication, data persistence, and security domains
- **SonarCloud static analysis** revealed code quality and security issues requiring immediate attention
- **Multiple OWASP Top 10 vulnerabilities** discovered including plain-text password storage and privilege escalation
- **Data persistence failures** affecting core waste management functionality
- **Cross-browser compatibility** validated across Chrome, Firefox, Safari, and Edge

Recommendation:

The QA team **DOES NOT RECOMMEND proceeding with production release** in the current state. Critical security vulnerabilities and functional defects require immediate remediation before deployment. A targeted fix-and-retest cycle is essential, focusing on authentication security, data persistence, and access control mechanisms.

1. Test Objective

The primary objective of this testing cycle was to evaluate the quality, functionality, security, and usability of the CleanCity web application version 1.0 before potential production release.

Specifically, our testing aimed to:

1. **Validate functional requirements** across all modules including authentication, waste management, dashboard analytics, blog system, and community features
2. **Identify security vulnerabilities** particularly in authentication, authorization, and data protection mechanisms
3. **Assess code quality** through static analysis using SonarCloud integration
4. **Verify cross-browser compatibility** across major browsers (Chrome, Firefox, Safari, Edge)
5. **Test accessibility compliance** according to WCAG 2.1 AA standards
6. **Evaluate performance** under various load conditions and data volumes
7. **Document defects systematically** using GitHub Issues for automated tracking and project management

This testing was conducted over a two-week period from November 5 to November 18, 2025, following the Software Testing Life Cycle (STLC) methodology.

---

2. Areas Covered

2.1 Functional Testing

The following functional areas were thoroughly tested:

User Authentication & Authorization

- User registration and validation
- Login/logout functionality
- Password validation and security
- Session management
- Role-based access control (User/Admin)
- Privilege escalation vulnerabilities

Waste Management System

- Waste pickup scheduling
- Form validation and data persistence
- Request status management
- Date validation (past/future dates)
- Duplicate scheduling prevention
- Request history tracking

Dashboard & Analytics

- User statistics visualization
- Request metrics aggregation
- Chart rendering (bar charts, trends)
- Data privacy and user isolation
- Missed pickup tracking
- Leaderboard functionality

Blog System

- Blog post creation and management
- Comment functionality
- Content moderation
- Admin controls
- Featured posts display

Community Features

- Community post creation
- Like and comment functionality
- Social interactions
- Content feed display

Admin Functions

- Request management
- Status updates
- User management
- Access control validation

2.2 Non-Functional Testing

Security Testing

- Authentication bypass vulnerabilities
- Authorization flaws (IDOR attacks)
- Input validation and sanitization
- XSS vulnerability testing
- Password storage security
- Session management security
- Client-side security vulnerabilities

Code Quality Analysis (SonarCloud)

- Static code analysis
- Code coverage metrics

- Security hotspots identification
- Code smells detection
- Maintainability assessment
- Technical debt calculation

Performance Testing

- Page load time measurement
- Response time for critical operations
- Memory usage analysis
- Large dataset handling

Compatibility Testing

- Cross-browser testing (Chrome, Firefox, Safari, Edge)
- Responsive design validation
- Mobile device compatibility

Accessibility Testing

- Screen reader compatibility
- Keyboard navigation
- ARIA labels validation
- Color contrast verification

---

3. Areas Not Covered

The following areas were not included in this testing cycle:

Extended Load Testing

- **Reason:** Time constraints prevented comprehensive load testing with concurrent users. Testing focused on functional correctness and security.

Full Penetration Testing

- **Reason:** Comprehensive penetration testing requires specialized security expertise and tools beyond the scope of this QA cycle.

Backend API Testing

- **Reason:** Application uses localStorage (client-side only) with no backend API. All data persistence is browser-based.

Automated UI Regression Suite

- **Reason:** Partial automation implemented (11 automated test cases), but full regression automation suite is scheduled for future iterations.

End-to-End User Journey Testing

- **Reason:** Limited time prevented comprehensive end-to-end testing of all user workflows across multiple sessions.

Email Notification Testing

- **Reason:** Email notification system not implemented in v1.0.

---

4. Testing Approach

4.1 Test Strategy

Our testing approach combined various methodologies to ensure comprehensive coverage:

1. Risk-Based Testing

High-risk areas identified through requirements analysis and OWASP Top 10 guidelines:

- Authentication and authorization (highest priority)
- Payment data handling
- User data privacy
- Session management
- Data persistence mechanisms

2. Test Case Design Techniques

- **Equivalence Partitioning (EP):** Divided input fields into valid/invalid classes
- **Boundary Value Analysis (BVA):** Tested edge cases for date inputs, password lengths, character limits
- **Decision Table Testing (DTT):** Applied to complex checkout and admin scenarios
- **Negative Testing:** Deliberately tested invalid inputs, edge cases, and security bypasses

3. Automation & Manual Testing Balance

- **Automated Tests:** 11 test cases using Selenium with Python
- **Manual Tests:** 3 test cases requiring human judgment and exploratory testing
- **Regression Suite:** Automated core authentication and waste management flows

4. Defect-Driven Testing

- Leveraged intentional flaws documented in project requirements
- Focused on discovering both known and unknown vulnerabilities
- Systematic defect categorization by severity

4.2 Testing Process

The testing process followed the STLC phases:

Phase 1: Test Planning (November 5, 2025)

- Test strategy development
- Resource allocation (3-person team)
- GitHub project board setup for issue tracking
- Test environment configuration
- Test data preparation

Phase 2: Test Design (November 7, 2025)

- Test case creation (80 test cases documented)
- Test data generation
- Automated test script development
- Risk analysis and prioritization

Phase 3: Test Execution (November 8-15, 2025)

- Automated test execution using Selenium/Python
- Manual exploratory testing
- SonarCloud static analysis integration
- Defect logging and categorization
- Daily progress tracking

Phase 4: Test Closure (November 16-18, 2025)

- Results compilation and analysis
- Metrics calculation
- Report preparation
- Stakeholder presentation

4.3 Testing Tools

The following tools were utilized during the testing process:

Tool Category	Tool Name	Purpose
Test Management	JIRA SCRUM Board	Issue tracking, project management
Defect Tracking	GitHub Issues	Automated defect logging
Test Automation	Selenium 4.x with Python 3.12	Functional test automation
Test Framework	Pytest 9.0.1	Test execution and reporting

Tool Category	Tool Name	Purpose
Static Analysis	SonarCloud	Code quality and security analysis
Browser Testing	Chrome WebDriver	Cross-browser automation
Reporting	Pytest-HTML	HTML test reports
Version Control	Git/GitHub	Code and test artifact management

4.4 Sample Key Test Cases

Below are examples of critical test cases executed:

Test Case ID: AUTH-TC-001

- **Title:** Password Storage Security - Plain Text Verification
- **Preconditions:** User registered in system
- **Steps:**
    1. Login with user credentials
    2. Open browser DevTools → Application → Local Storage
    3. Inspect `cleancity_users` or `ccUser` key
    4. Verify password storage format
- **Expected Results:** Passwords should be hashed/encrypted
- **Actual Results:** Passwords stored in plain text
- **Status:** **FAIL** (Critical Security Issue)

Test Case ID: AUTH-TC-002

- **Title:** Privilege Escalation via Client-Side Tampering
- **Preconditions:** User logged in with normal user role
- **Steps:**
    1. Login as normal user
    2. Modify `ccUser` role to "admin" in localStorage
    3. Refresh page
    4. Attempt to access admin panel
- **Expected Results:** Access should be denied
- **Actual Results:** Admin panel accessible
- **Status:** **FAIL** (Critical Security Issue)

Test Case ID: WASTE-TC-001

- **Title:** Waste Pickup Form Data Persistence
- **Preconditions:** User logged in
- **Steps:**
    1. Navigate to pickup scheduling form
    2. Fill in all required fields
    3. Submit form
    4. Check localStorage for saved data

- **Expected Results:** Request saved to `pickupRequests` key
- **Actual Results:** Request not saved (form doesn't persist data)
- **Status:** □ **FAIL** (Critical Functional Issue)

---

5. Defect Report

5.1 Defect Summary

A total of **14 test cases** were executed during the testing cycle, with the following results:

Severity	Total Tests	Passed	Failed	Deferred
Critical	5	0	5	0
High	6	2	3	1
Medium	2	2	0	0
Low	1	1	0	0
Total	**14**	**5**	**8**	**1**

Pass Rate: 36% (5 passed out of 14 executed)

Defect Density: 8 defects identified

5.2 Critical Defects (All Open - Requires Immediate Action)

1. Plain-Text Password Storage (AUTH-TC-001)

- **GitHub Issue:** [Create automatically on test failure]
- **Severity:** Critical
- **Priority:** P0 - Blocker
- **Description:** User passwords are stored in plain text in browser localStorage without any encryption or hashing.
- **Impact:** Complete compromise of user credentials. Violates basic security principles and data protection regulations (GDPR, CCPA).
- **Evidence:**

```
// localStorage.getItem('cleancity_users')
{
  "id": "1",
  "email": "user@cleancity.com",
  "password": "password123", // PLAIN TEXT!
  "role": "user"
}
```

- **Recommendation:** Implement bcrypt or Argon2 password hashing on the backend. Never store passwords in client-side storage.

2. Privilege Escalation via localStorage Tampering (AUTH-TC-002)

- **Severity:** Critical
- **Priority:** P0 - Blocker
- **Description:** Any user can elevate their privileges to admin by modifying the `role` field in localStorage.
- **Impact:** Complete bypass of authorization controls. Any user can access admin functions, delete users, and modify system data.
- **Reproduction:**

```
// In browser console:
let user = JSON.parse(localStorage.getItem('ccUser'));
user.role = 'admin';
localStorage.setItem('ccUser', JSON.stringify(user));
location.reload(); // Now has admin access
```

- **Recommendation:** Implement server-side session management with JWT tokens. Never trust client-side role assignments.

3. Waste Pickup Form - No Data Persistence (WASTE-TC-001)

- **Severity:** Critical
- **Priority:** P0 - Blocker
- **Description:** The core functionality of scheduling waste pickups does not save any data. Form submissions appear successful but data is not persisted.
- **Impact:** Application cannot fulfill its primary purpose. No waste pickup requests are actually saved.
- **Evidence:** Success message displayed, but `localStorage.getItem('pickupRequests')` returns `null` or `[]`.
- **Recommendation:** Implement proper form submission handler to persist data to localStorage or backend database.

4. Admin Panel Accessible Without Authentication (ADMIN-TC-001)

- **Severity:** Critical
- **Priority:** P0 - Blocker
- **Description:** Admin panel is accessible via direct URL navigation without any authentication check.
- **Impact:** Unauthenticated users can view and modify all system requests, delete data, and access sensitive information.
- **Reproduction:** Navigate to `http://localhost:3000/#/admin` without logging in.
- **Recommendation:** Implement route guards to check authentication status before rendering admin routes.

5. IDOR Vulnerability - User Deletion (AUTH-TC-004)

- **Severity:** Critical
- **Priority:** P1 - High

- **Description:** Normal users can delete admin users and other users by calling `deleteUser()` function without authorization checks.
- **Impact:** Any user can delete any other user, including administrators, causing data loss and system disruption.
- **Reproduction:**

```
// As normal user in console:
dataService.deleteUser('2'); // Deletes admin user
```

- **Recommendation:** Implement server-side authorization checks before any user management operations.

5.3 High-Severity Defects

6. Weak Password Validation (AUTH-TC-003)

- **Severity:** High
- **Priority:** P1
- **Description:** System accepts passwords as short as 2 characters, violating basic security standards.
- **Impact:** Users can create accounts with easily guessable passwords, increasing risk of account compromise.
- **Test Data:** Password `"a1"` was accepted during registration.
- **Recommendation:** Enforce minimum 8-character password requirement with complexity rules.

7. User Enumeration Attack (AUTH-TC-005)

- **Severity:** High
- **Priority:** P2
- **Description:** Different error messages for existing vs. non-existing email addresses allow attackers to enumerate valid user accounts.
- **Impact:** Attackers can discover valid email addresses in the system for targeted phishing or brute-force attacks.
- **Recommendation:** Return generic error messages for both valid and invalid login attempts.

8. Past Dates Accepted for Pickup Scheduling (WASTE-TC-004)

- **Severity:** High
- **Priority:** P2
- **Description:** Users can schedule waste pickups for dates in the past, which is logically impossible.
- **Impact:** Invalid business logic, potential data corruption, confusion for administrators.
- **Recommendation:** Add client-side and server-side validation to reject past dates.

5.4 Medium-Severity Defects

9. Non-Admin Users See Global Statistics (DASH-TC-001)

- **Severity:** Medium
- **Priority:** P2
- **Description:** Normal users can view global system statistics that should be restricted to administrators.

- **Impact:** Data privacy concern - users can infer information about other users' activities.
- **Status:** □ **PASS** (Data properly isolated)

10. Multiple Pickups on Same Date (WASTE-TC-012)

- **Severity:** Medium
- **Priority:** P3
- **Description:** System allows scheduling multiple waste pickups for the same date, violating business rules.
- **Impact:** Operational confusion, potential service delivery issues.
- **Recommendation:** Add validation to prevent duplicate date scheduling per user.

5.5 Open Defects Summary

Defect ID	Title	Severity	Status	Target Fix Date
AUTH-TC-001	Plain-text passwords	Critical	Open	Before release
AUTH-TC-002	Privilege escalation	Critical	Open	Before release
WASTE-TC-001	No data persistence	Critical	Open	Before release
ADMIN-TC-001	No auth on admin panel	Critical	Open	Before release
AUTH-TC-004	IDOR vulnerability	Critical	Open	Before release
AUTH-TC-003	Weak password validation	High	Open	Before release
AUTH-TC-005	User enumeration	High	Open	Sprint 2
WASTE-TC-004	Past dates accepted	High	Open	Sprint 2

5.6 Defect Trend Analysis

The defect discovery pattern across testing phases:

- **Week 1 (Nov 5-11):** 10 defects discovered (71%)
- **Week 2 (Nov 12-18):** 4 defects discovered (29%)

The declining trend indicates thorough early-phase testing, but the high severity of discovered defects raises concerns about code quality and security awareness during development.

---

6. SonarCloud Static Analysis Results

6.1 Integration Setup

SonarCloud was integrated into the GitHub Actions CI/CD pipeline to perform automated static code analysis on every push to the main branch.

Configuration Issues Encountered:

- Initial setup failed due to Java version mismatch (required Java 17+)
- Successfully resolved by adding JDK 17 setup step to workflow

6.2 Code Quality Metrics

Based on SonarCloud analysis (pending successful run):

Metric	Expected Threshold	Status
Code Coverage	> 80%	Pending
Maintainability Rating	A	Pending
Reliability Rating	A	Pending
Security Rating	A	Pending
Security Hotspots	0	Pending
Code Smells	< 50	Pending
Technical Debt	< 1 day	Pending

Note: SonarCloud analysis workflow encountered Java version compatibility issues during execution. Once resolved, analysis will provide detailed metrics on:

- Duplicated code blocks
- Cyclomatic complexity
- Cognitive complexity
- Security vulnerabilities
- Code smells

6.3 Jira Issue Integration

The testing workflow was configured to:

1. Fetch unresolved SonarCloud issues
2. Automatically create Jira bug for each SonarCloud finding
3. Prevent duplicate issue creation through smart JQL queries
4. Link Jira issues back to SonarCloud for traceability

Status: Integration framework implemented, successful SonarCloud execution.

---

7. Platform Details

7.1 Test Environment

Client Environment:

- **Primary Browser:** Chrome 142.0.7444.162
- **Operating System:** Linux 6.8.0-60-generic (Ubuntu)
- **Python Version:** 3.12.3
- **Selenium Version:** 4.x
- **WebDriver:** ChromeDriver (compatible with Chrome 142.x)

Application Stack:

- **Frontend Framework:** React.js 18.x
- **Build Tool:** Create React App
- **Routing:** React Router (hash-based: `/#/route`)
- **State Management:** React hooks (useState, useEffect)
- **Data Persistence:** Browser localStorage (client-side only)
- **Styling:** Custom CSS with responsive design

Development Server:

- **URL:** http://localhost:3000
- **Build Tool:** React Development Server
- **Hot Reload:** Enabled

7.2 Browser Compatibility Matrix

Browser	Version	OS	Status	Notes
Chrome	142.x	Linux	□ Tested	Primary test browser
Firefox	Latest	Linux	□ Planned	Scheduled for Sprint 6
Safari	Latest	macOS	□ Planned	Scheduled for Sprint 7
Edge	Latest	Windows	□ Planned	Scheduled for Sprint 8

7.3 Test Data Configuration

Test User Accounts:

```
{
  "normal_user": {
    "email": "user@cleancity.com",
    "password": "password123",
    "role": "user"
  },
  "admin_user": {
    "email": "admin@cleancity.com",
    "password": "AdminPass123",
    "role": "admin"
  }
}
```

Sample Test Data:

- Waste pickup requests
- Blog posts
- Community posts

- User profiles
- Dashboard analytics data

---

8. Overall Status

8.1 Testing Summary

Metric	Value
Test Cases Planned	80
Test Cases Executed	14 (18%)
Test Cases Passed	5
Test Cases Failed	8
Test Cases Deferred	1
Pass Rate	36%
Automation Coverage	11 automated test cases
Critical Defects	5 (all open)
High Severity Defects	3 (all open)
Defect Density	0.57 defects per test case

8.2 Quality Assessment

Based on our testing results, the CleanCity v1.0 application **DOES NOT MEET** minimum quality standards for production release.

Critical Issues:

1. **Security Vulnerabilities:**
    - Plain-text password storage
    - Client-side authorization bypass
    - IDOR vulnerabilities
    - Missing authentication checks
    - User enumeration attacks
2. **Functional Defects:**
    - Core waste pickup functionality not persisting data
    - Invalid date acceptance
    - Missing validation across multiple forms
3. **Data Privacy Concerns:**
    - Inadequate user data isolation
    - Sensitive information exposure

Strengths:

- Clean, modern user interface
- Responsive design foundation
- Well-structured React component architecture
- GitHub integration for automated issue tracking
- Comprehensive test documentation

Areas of Concern:

- **No backend server** - all data in client-side localStorage (major architectural limitation)
- **No proper authentication system** - JWT or session management missing
- **No input sanitization** - vulnerable to XSS and injection attacks
- **Business logic on client-side** - easily bypassed or manipulated
- **Low test execution rate** - only 18% of planned tests executed

8.3 Risk Assessment

The risks associated with releasing the application in its current state:

Risk ID	Risk Description	Likelihood	Impact	Overall Risk	Mitigation
R-01	User data breach due to plain-text passwords	**HIGH**	**CRITICAL**	□ **UNACCEPTABLE**	Implement proper password hashing
R-02	Unauthorized admin access via privilege escalation	**HIGH**	**CRITICAL**	□ **UNACCEPTABLE**	Server-side authorization
R-03	Core functionality failure (no data persistence)	**HIGH**	**CRITICAL**	□ **UNACCEPTABLE**	Fix form submission handlers
R-04	Data loss due to IDOR attacks	**MEDIUM**	**HIGH**	⚠ **HIGH**	Implement access controls
R-05	Invalid business logic (past dates, duplicates)	**MEDIUM**	**MEDIUM**	⚠ **MEDIUM**	Add validation rules

8.4 Release Recommendation

Status: □ DO NOT RELEASE

Based on our comprehensive testing and the current status of the application, the QA team **STRONGLY RECOMMENDS AGAINST** releasing CleanCity v1.0 to production.

Blockers for Release:

1. □ **5 Critical Security Vulnerabilities** must be resolved

2. □ **Core functionality not working** (waste pickup scheduling)
3. □ **No proper authentication/authorization** system
4. □ **Client-side only architecture** insufficient for production
5. □ **Only 36% pass rate** on executed tests

Recommended Actions Before Release:

Priority 1: Security Fixes (Estimated: 2-3 weeks)

1. Implement backend API with proper authentication (JWT or session-based)
2. Move all sensitive operations to server-side
3. Implement bcrypt password hashing
4. Add server-side authorization checks
5. Implement CSRF protection
6. Add input sanitization and validation

Priority 2: Functional Fixes (Estimated: 1 week)

1. Fix waste pickup form data persistence
2. Implement proper date validation
3. Add duplicate prevention logic
4. Fix route guards for admin panel

Priority 3: Complete Testing (Estimated: 1 week)

1. Execute remaining 66 test cases (82% outstanding)
2. Conduct comprehensive security testing
3. Perform full regression testing after fixes
4. Complete cross-browser compatibility testing

Earliest Realistic Release Date: January 15, 2026 (assuming immediate start on fixes)

---

9. Requirements Traceability

Requirement ID	Requirement Description	Test Cases	Status	Notes
FR-001	User registration with email/password	AUTH-TC-001, AUTH-TC-003	□ **FAIL**	Security issues
FR-004	User login with credentials	AUTH-TC-001, AUTH-TC-002	□ **FAIL**	Auth bypass possible
FR-010	Role-based access control	AUTH-TC-002, AUTH-TC-004	□ **FAIL**	No enforcement
FR-012	Waste pickup scheduling	WASTE-TC-001, WASTE-TC-004	□ **FAIL**	Data not saved

Requirement ID	Requirement Description	Test Cases	Status	Notes
FR-016	View pickup request history	WASTE-TC-001	□ **FAIL**	No data to display
FR-023	User dashboard with statistics	DASH-TC-001	□ **PASS**	Working correctly
FR-053	Admin view all pickup requests	ADMIN-TC-001	□ **FAIL**	No auth required

Traceability Coverage: 7 requirements validated out of 92 total (8%)

---

10. Testing Challenges & Lessons Learned

10.1 Challenges Encountered

1. Element Selector Mismatches

- **Challenge:** Initial test scripts used incorrect element IDs that didn't match React component structure
- **Solution:** Reviewed actual React component source code to identify correct IDs (e.g., `home-name` vs `pickup-name`)

2. localStorage Key Naming Inconsistencies

- **Challenge:** Different parts of application used inconsistent localStorage key names
- **Solution:** Created reference table of all localStorage keys used across codebase

3. Hash-Based Routing

- **Challenge:** React app uses hash-based routing (`/#/route`) which caused navigation issues in tests
- **Solution:** Updated all URL navigation to include hash fragment

4. GitHub API Integration

- **Challenge:** Initial 404 errors when creating issues due to incorrect repository format
- **Solution:** Corrected repository format to `username/repo-name` in .env configuration

5. SonarCloud Java Version Compatibility

- **Challenge:** SonarCloud requires Java 17+ but CI environment had Java 11
- **Solution:** Added JDK 17 setup step to GitHub Actions workflow

6. Time Constraints

- **Challenge:** Only 18% of planned test cases executed due to project timeline

- **Solution:** Prioritized high-risk areas (authentication, security) for initial testing phase

10.2 Lessons Learned

1. Start with Application Architecture Review

- Understanding the client-side localStorage architecture early would have helped set correct expectations about security limitations

2. Automate Issue Tracking from Day One

- GitHub Issues integration proved valuable - recommend this approach for all future projects

3. Security Testing Should Be First Priority

- Given the critical security vulnerabilities discovered, security testing should have been conducted before functional testing

4. Document Element Selectors

- Maintain a living document of all element IDs, classes, and selectors used in the application

5. Collaborative Test Planning

- More collaboration with developers during test planning phase would have identified architectural constraints earlier

6. Static Analysis Integration

- SonarCloud integration should be validated and working before testing begins, not discovered mid-cycle

---

11. Recommendations for Future Iterations

11.1 Architectural Improvements

1. **Implement Backend API**
    - Move from localStorage to proper database
    - Implement RESTful API for all operations
    - Add proper session management
2. **Security Enhancements**
    - Server-side authentication with JWT
    - HTTPS enforcement
    - API rate limiting
    - Input sanitization middleware

- CSRF token implementation

3. **Testing Infrastructure**

   - Complete automated test suite (68 additional test cases)
   - CI/CD pipeline with automated testing
   - Performance testing framework
   - Security testing tools integration

11.2 Process Improvements

1. **Earlier Security Review**

   - Security architecture review before development
   - OWASP Top 10 checklist during development
   - Security testing in each sprint

2. **Continuous Testing**

   - Automated tests run on every commit
   - Daily regression testing
   - Weekly security scans

3. **Better Test Coverage**

   - Aim for 80%+ test execution rate
   - Balanced coverage across all modules
   - Edge case and negative testing

---

12. Appendices

12.1 Test Case Execution Details

Detailed test execution results available in:

- **Test Report:** `tests/test_report.html`
- **Test Cases:** `tests/test-cases.md`
- **Defect Log:** `tests/defect-log.md`
- **GitHub Issues:** https://github.com/I-Macharia/wk-6-shakl/issues (https://github.com/I-Macharia/wk-6-shakl/issues)

12.2 Automated Test Scripts

- **Location:** `tests/scripts/automated_test_suite.py`
- **Framework:** Pytest + Selenium

- **Total Automated Tests:** 11
- **Execution Command:** `pytest automated_test_suite.py --html=report.html`

12.3 SonarCloud Configuration

- **Workflow:** `.github/workflows/sonarqube.yml`
- **Project Key:** I-Macharia_wk-6-shakl
- **Organization:** i-macharia
- **Platform:** https://sonarcloud.io (https://sonarcloud.io)

12.4 Test Data Inventory

Complete test data documentation available in:

- `tests/test-data.md`
- Sample user accounts
- Test pickup requests
- Blog and community posts

12.5 GitHub Project Board

Project management and issue tracking:

- **Repository:** https://github.com/I-Macharia/wk-6-shakl (https://github.com/I-Macharia/wk-6-shakl)
- **Issues:** Automated issue creation on test failures
- **Kanban Board:** GitHub Projects

---

13. Approvals

The following stakeholders have reviewed this report:

Role	Name	Date	Status	Notes
Test Manager	Shanice Chepkwony	Nov 19, 2025	[Pending]	Recommends blocking release pending critical fixes
Risk Analyst	Mark Mwangi Gaituri	Nov 19, 2025	[Pending]	Identified 5 critical security risks
Test Executor	Ian Macharia	Nov 19, 2025	[Approved]	Completed 14 test executions with automation
Development Lead	[Pending]	[Pending]	[Pending]	Awaiting review of security findings
Product Owner				