ReturnZero_

20 August 2025
COS301 - Capstone
University of Pretoria

BushBuddy

BushBuddy Testing Policy

Version 1

Team members
Ruan Esterhuizen (u23532387)
Ruben Hannes Gadd (u23633353)
Raphael Rato (u22887581)
Tom Schulz (u05039364)
Jean Steyn (u22537229)

Contact
g24capstone@gmail.com

1. Purpose and Scope

The purpose of this Testing Policy is to define the testing approach, methodologies, and processes for ensuring the quality, reliability, and maintainability of the software. This policy applies to all parts of the system, including frontend, backend, integrations, deployment pipelines, and non-functional characteristics such as performance, reliability, and accessibility.

2. Testing Objectives

- Detect defects early in the development cycle.
- Ensure individual units of code behave correctly and predictably.
- Verify that integrated modules interact correctly.
- Validate that the entire application functions as intended for end-users.
- Prevent regressions when new features or changes are introduced.
- Assess non-functional qualities such as performance, reliability, and accessibility.
- Maintain a high level of confidence in automated deployments.

3. Testing Levels

3.1 Unit Testing

Scope: Backend services and frontend components.
Tools: Jest.
Methodology:

- Backend divided into layers (controllers, services, repositories).
- Each layer tested in isolation using mocks/stubs.
- Mock data used to validate expected behaviour.
- Tests cover normal cases, edge cases, and failure handling.
- Coverage Goal: ≥80% coverage of core logic.

3.2 Integration Testing

Scope: Interactions between backend layers and modules.

Tools: Jest.

Methodology:

- Verify correct data flow between layers (service ↔ repository ↔ controller).
- Ensure error handling and data transformations work across boundaries.
- In-memory or mocked databases used for repeatability.
- Coverage Goal: All critical workflows tested at least once.

3.3 End-to-End (E2E) Testing

Scope: Complete workflows, from frontend UI through backend and database.

Tools: Cypress.

Methodology:

- Test critical user journeys (authentication, data submission, retrieval).
- Simulate real browser interactions.
- Regression tests included for updated features.
- Coverage Goal: 100% coverage of critical systems.

4. Non-Functional Testing

4.1 Reliability Testing

Scope: AI image detection model.

Methodology:

- Tested model accuracy using benchmark datasets.
- Verified detection consistency across multiple test runs.
- Measured false positives/negatives to assess real-world reliability.

4.2 Performance Testing

Scope: Frontend and backend performance under normal and peak usage.

Tools: Google Lighthouse, PageSpeed Insights (pagespeed.web.dev).

Methodology:

- Measured page load times, rendering speed, and server response times.
- Identified performance bottlenecks (e.g., unoptimized assets, blocking scripts).
- Ensured acceptable performance metrics under expected usage conditions.

4.3 Accessibility Testing

Tools: Google Lighthouse.

Methodology:

- Evaluated compliance with accessibility standards (e.g., contrast, labels, navigation).
- Ensured the application is usable with assistive technologies.
- Addressed major accessibility issues before deployment.

4.4 Usability Testing

Tools: Google Form **Methodology**:

- sent out link and form to multiple different users
- asked users to complete form based on their experience using the application
- used feedback to address any issues found during testing

5. Automation

5.1 Automated Testing

Toolchain: GitHub Actions.

Process:

- Runs on pushes and pull requests to main and dev branches.

Selective execution:

- only relevant tests run depending on which codebase area was modified.
- Builds only pass if all associated tests succeed.

5.2 Automated Deployment

Environments: Render

Development: dev branch → staging deployment.

Production: main branch → production deployment.

Process:

- Triggered after successful automated tests.
- Rollback available for failed deployments.