QA Testing Framework

Comprehensive Documentation

Generated: 2026-02-16

1. Introduction

This document provides a comprehensive overview of the QA Testing Framework, a unified tool designed to ensure the quality, consistency, and performance of web applications. The framework consolidates visual regression testing, accessibility audits, broken link checking, and SEO performance analysis into a single, cohesive interface.

2. Architecture Overview

The application is built using a modern, lightweight architecture that separates checks into modular components while providing a unified dashboard for execution and reporting.

Component	Technology	Role
Backend	Python + Flask	API handling, orchestration, core logic processing.
Frontend	Vanilla HTML/JS/CSS	User Interface, realtime updates via polling.
Database	Local JSON / FS	Persistent storage for job history and artifacts.
Browsers	Playwright	Headless browsing for screenshots and rendering.
Vision	OpenCV + SSIM	Image alignment, structural similarity, diff detection.

3. Backend Technology Stack

3.1 Core Framework: Python & Flask

What: We use Python 3.9+ with the Flask micro-framework.

Why:

- Simplicity: Flask is lightweight and allows for rapid development of API endpoints.
- Ecosystem: Python has the strongest ecosystem for data processing, image manipulation (OpenCV), and automation.

3.2 Browser Automation: Playwright

What: Microsoft's Playwright library is used to control headless Chromium browsers.

Why:

- Reliability: It is faster and more reliable than Selenium, managing dynamic content and network waiting states ('networkidle').
- Fidelity: Renders modern web features exactly as users see them.

3.3 Computer Vision: OpenCV & scikit-image

What: OpenCV is used for image alignment (ORB features) and processing. scikit-image provides the Structural Similarity Index (SSIM).

Why:

- Pixel-Perfect Accuracy: SSIM aligns with human visual perception better than simple pixel subtraction.
- Robustness: ORB alignment handles slight shifts in rendering, ensuring we compare the correct elements.

3.4 Reporting: ReportLab

What: A library for programmatically generating PDF documents.

Why: Allows us to generate professional, sharable PDF reports with embedded images (diff overlays) and text.

4. Frontend Technology Stack

What: Pure HTML5, CSS3, and Vanilla JavaScript (ES6+).

Why:

- Performance: Zero compilation steps, instant loading, and minimal overhead.
- Simplicity: Easy for any developer to maintain without needing knowledge of complex frameworks like React or Vue.
- Direct Control: Direct DOM manipulation for features like the interactive image difference toggle and file upload drag-and-drop.

5. Detailed Feature Breakdown

5.1 Visual Testing Studio

This module compares a Figma design (PNG) against a live staged URL.

Key Features:

- Smart Alignment: Automatically aligns the Figma design with the screenshot using feature matching, correcting for minor crop differences.
- Smart Cropping: Validates specific components (via selector) against full-page designs by auto-detecting the component's location.
- Dynamic Masking: Users can provide CSS selectors (e.g., '.ads') to mask out dynamic content that would cause false positives.
- Noise Tolerance: Adjustable sensitivity (Strict/Medium/Relaxed) to ignore minor font-rendering differences.
- Output: Generates Diff Overlay, Heatmap, and Aligned Side-by-Side images.

5.2 Broken Links & Asset Crawler

This module recursively crawls a website to ensure site health.

Key Features:

- Deep Crawling: Visits all internal links to finding broken pages (404s).
- Asset Validation: Checks all images and icons to ensure they load correctly.
- Integration: Uses `requests` for speed and `BeautifulSoup` for parsing.

5.3 Accessibility Auditor

Automated checks against WCAG 2.1 standards.

Key Features:

- Checks: Missing alt text, form labels, heading hierarchy, contrast ratios, and aria-labels.
- Sitemap Scanning: Can digest an entire XML sitemap and audit hundreds of pages in parallel.
- Reporting: Categorizes issues by severity (Critical, Serious, Moderate) and provides direct fix suggestions.

5.4 SEO & Performance Monitor

Ensures pages meet search engine and user experience standards.

Key Features:

- Meta Analysis: Validates Title, Description, and Viewport tags.
- H1 Verification: Ensures a single, unique H1 exists per page.
- Performance Metrics: Measures response time and page weight.

6. Visual Baseline Versioning

The framework includes robust capabilities for managing visual history and decisions.

6.1 Version History & Baselines

What: Every test run creates a unique, immutable record stored in the file system.

> Why: Allows teams to audit UI evolution over time. If a regression occurs, you can look back to see exactly when the change was introduced.

6.2 Approve / Reject Workflow

What: An interactive review interface allows QA engineers to explicitly 'Approve' or 'Reject' a visual change.

> Why: Distinguishes between intentional design updates and accidental regressions. Approved runs serve as the new 'signed-off' state.

6.3 Storing Historical Diffs

What: All artifacts (Diff Overlays, Heatmaps, PDF Reports, and Input Images) are persisted indefinitely in unique job directories.

> Why: Ensures that evidence of failures is never lost. You can download the exact diff image from a test run 3 months ago.

6.4 Rollback Capability

What: Support for referencing previous successful runs.

> Why: If a new deployment is rejected, the team can reference the last 'Approved' run to verify what the correct state should be during a rollback.

7. Third-Party Integrations

The framework integrates directly with project management tools to streamline the feedback loop.

7.1 Jira Integration

- Allows users to create Jira tasks directly from the test result page.
- Automatically attaches the issue screenshot and description.

7.2 GitHub Integration

- Creates GitHub Issues for verified bugs.
- Tags issues with 'visual-regression' labels for easy filtering.

8. Visual Workflow & Process Map

The following diagram illustrates the end-to-end workflow for visual regression testing, from initiation to baseline promotion.

Step	Action / Decision	Outcome
1. Initiation	User enters Stage URL	System checks for active baseline
2. Baseline Check	Has Active Baseline?	Yes: Auto-load Stored Image No: Prompt for Figma Upload
3. Execution	Capture Screenshot & Compare	Generates Diff, SSIM Score, Heatmap
4. Review	Is Diff Detected?	No: Test Passed ■ Yes: Human Review Required ■■
5. Decision	User Reviews Diff	Approve: Promotes Stage Image to Baseline Reject: Marks Job as Failed
6. Maintenance	Baseline Updated	New version (v2, v3...) created & stored

9. Project Roadmap

The future development roadmap outlining the progression of the framework.

Phase / Timeline	Key Milestones	Status
Q1 2026 Foundation	• Core Framework (Flask + Playwright) • Visual Testing Engine (OpenCV) • Reporting Infrastructure	Completed ■
Q2 2026 Expansion	• Accessibility Audits (WCAG) • Broken Link Crawler • SEO Performance Checks • Jira/GitHub Integration	Completed ■
Q3 2026 Scalability	• Cloud Storage Support (S3/GCS) • Multi-user Authentication • Dashboard Analytics & Trends	Planned ■■
Q4 2026 Intelligence	• AI Root Cause Analysis • Predictive Flakiness Detection • CI/CD Plugin Ecosystem	Planned ■■