■ QA Testing Framework — Complete Feature Documentation

Version: 2.0 | **Last Updated:** February 2026

Author: QA Automation Team

Stack: Python (Flask) + Playwright + OpenCV + SSIM + Google PageSpeed API

■ Table of Contents

Overview

The **QA Testing Framework** is a self-hosted, all-in-one quality assurance platform built for web development teams. It provides automated visual regression testing, broken link detection, accessibility auditing, and SEO/performance analysis — all from a single browser-based UI running on `localhost:7860`.

Key Capabilities

Capability	Description
Visual Regression Testing	Pixel-by-pixel comparison of Figma designs against live staging pages
Broken Links Detection	Crawls pages for broken images, icons, and hyperlinks
Accessibility Auditing	WCAG 2.0/2.1 compliance checks (AA and AAA levels)
SEO & Performance	Google PageSpeed Insights integration for Core Web Vitals

Capability	Description
Baseline Management	Version-controlled baseline images with rollback support
Bug Tracking Integration	One-click issue creation in Jira and GitHub
CI/CD Pipeline	GitHub Actions workflow for automated testing on every push

---

Module 1: Visual Testing Studio

The heart of the framework. Compares a **reference image** (Figma design / baseline) against a **live screenshot** captured from a staging URL.

1.1 Core Comparison Engine

Feature	Description
SSIM Scoring	Structural Similarity Index between reference and live page (0.0–1.0). Higher = more similar.
Pixel Diff Detection	Identifies regions where pixels differ between the two images.
Diff Overlay	Generates a visual overlay highlighting all detected differences in red.
Diff Heatmap	Creates a color-mapped heatmap showing intensity of differences across the page.
Automatic Alignment	Uses ORB feature matching to align the live screenshot to the reference, handling small offsets.
Smart Cropping	Limits comparison to the overlapping area between reference and live images, preventing false positives from size mismatches.

How it works:

1. A screenshot of the staging URL is captured using Playwright (headless Chromium).
2. The screenshot is aligned to the reference image using ORB-based feature matching.
3. SSIM is computed to get a similarity score.
4. Pixel difference regions are detected, classified, and highlighted.
5. A PDF report is generated with all findings.

1.2 Smart Issue Classification

Each detected difference is automatically classified into a specific category. This helps developers quickly understand **what kind of issue** it is.

Category	Badge	How It's Detected	Example
Section Spacing / Margin Issue	■ Spacing	Wide, short diff region with uniform fill (gap between stacked sections)	`Section Spacing / Margin Issue (height: 24px)`
Column Spacing / Gap Issue	■ Spacing	Tall, narrow diff region (gap between side-by-side elements)	`Column Spacing / Gap Issue (width: 12px)`
Section Spacing Mismatch	■ Spacing	Cross-correlation detects same content shifted vertically	`Section Spacing Mismatch (~15px shift)`
Padding/Margin Difference	■ Spacing	Top or bottom of region is uniform but middle has content	`Padding/Margin Difference (top)`
Color/Style Mismatch	■ Style	SSIM > 0.90 between crops (structure same, colors different)	`Color/Style Mismatch`
Text/Content Mismatch	■ Content	Edge density > 5% (text-heavy region differs)	`Text/Content Mismatch`
Missing Content (Text/List)	■ Content	Reference has more text lines than live	`Missing Content (Text/List)`
Extra Content (Text/List)	■ Content	Live has more text lines than reference	`Extra Content (Text/List)`
Missing Element	■■ Element	Reference has content, live is blank/white	`Missing Element`
Extra Element	■■ Element	Reference is blank, live has content	`Extra Element`
Layout Mismatch	■ Layout	General structural difference (fallback)	`Layout Mismatch`

Each issue card in the UI shows:

- **Category badge** (color-coded)
- **Issue label** with location and dimensions
- **Cropped screenshot** of the affected region
- **Coordinates** (X, Y) and **size** (WxH pixels)
- **One-click actions**: Download, Send to Jira, Send to GitHub

1.3 Validation Type Filters

You can choose which types of differences to check. Unchecked types are excluded from results.

Filter	What it controls
■ **Layout & Structure**	Catches spacing, padding, missing/extra elements, layout mismatches
■ **Text & Content**	Catches text differences, missing/extra content
■ **Colors & Styles**	Catches color/gradient/style changes

Use case: If only layout matters (e.g., you're redesigning colors intentionally), uncheck "Colors & Styles" to avoid noise.

1.4 Custom Pixel Sensitivity

Fine-tune how strictly the tool detects differences using the **Custom Pixel Validation Sensitivity** slider.

Preset	Sensitivity	Effect
Loose (25%)	Low	Only catches major, obvious differences. Ignores subtle pixel-level noise.
Medium (50%)	Standard	Balanced — catches noticeable differences without flagging anti-aliasing.
Strict (75%)	High	Catches most differences including small color shifts.
Pixel-perfect (95%)	Maximum	Catches even the tiniest 1-2px differences. Use for final sign-off.

How it works (Detect-then-Filter approach):

1. **Detection**: All potential difference regions are found using consistent Otsu auto-thresholding.
2. **Severity Filtering**: For each region, the **mean pixel difference intensity** is measured. Only regions above a sensitivity-derived threshold are kept.
3. **Result**: Higher sensitivity → lower severity threshold → more issues reported.

This ensures: **Pixel-perfect > Strict > Medium > Loose** in issue count, always.

1.5 Capture Size Control

Controls the height and scope of the captured screenshot to focus comparison on what matters.

Option	Description
Full Page (Entire Scroll)	Captures the full scrollable height of the page. Default behavior.
Viewport Only (Above the Fold)	Captures only the visible viewport area. Great for hero section comparisons.
Custom Max Height	Captures full page, then crops to a specified max pixel height (e.g., 3000px).

Additional control:

Option	Description
Remove Sections	CSS selectors for elements to completely **remove from the DOM** before capture (e.g., `footer, .sidebar, .newsletter-section`). Unlike "Ignore Elements" which just hides them visually, this collapses the space and actually reduces the page height.
Ignore Elements	CSS selectors for elements to **visually hide** (`visibility: hidden`). The space is preserved but the content becomes invisible. Use for dynamic content like timestamps, ads, or cookie banners.

Key difference:

- `Ignore Elements` → `visibility: hidden` → space preserved, element invisible
- `Remove Sections` → `display: none` → space collapses, page shrinks

1.6 Component Selector

Capture and compare **only a specific component** instead of the full page.

- **Input:** A CSS selector (e.g., `#header`, `.hero-section`, `[data-testid="navbar"]`)
- **Behavior:** Playwright locates the element and takes a screenshot of just that element
- **Use case:** Compare individual components like headers, footers, or cards without full-page noise

---

1.7 Viewport Options

Test across different screen sizes:

Preset	Resolution
Desktop	1366×768
Mobile	390×844

Preset	Resolution
1280×800	1280×800
1440×900	1440×900 (default)
1920×1080	1920×1080

1.8 Noise Tolerance

Controls the general sensitivity independently of the pixel threshold:

Level	Description	Best for
Strict	Sensitive to small changes	Final QA sign-off
Medium	Standard sensitivity	Day-to-day testing
Relaxed	Ignores minor noise	Early development stages

1.9 Pass Threshold (SSIM)

Sets the minimum SSIM score for a comparison to be marked as **"Passed"**.

- **Default:** 0.85
- **Range:** 0.50 – 0.99
- A comparison **passes** if: `SSIM ≥ threshold AND change_ratio ≤ 5%`

1.10 Wait Time

Configurable delay (in milliseconds) before taking the screenshot. Allows:

- JavaScript animations to complete
- Lazy-loaded images to render
- Dynamic content to populate
- **Default:** 1000ms

1.11 Animation & Noise Reduction

Automatically applied before every screenshot capture:

- All CSS `animation` properties are disabled
- All CSS `transition` properties are disabled
- Text cursor (`caret-color`) is made transparent
- This prevents flickering content from creating false positives

---

Module 2: Broken Links Checker

Crawls a webpage to find broken assets and links.

2.1 Check Types

Type	What it crawls	Issues detected
Broken Links	All `<a>` tags + `<img>` / `<link>` tags	HTTP 404/500 errors, unreachable URLs, missing images
Overlapping & Breaking	DOM element positions using Selenium	Overlapping elements, elements breaking out of containers, z-index conflicts
All (Comprehensive)	Combines both crawlers	Full audit of links, images, and visual overlap issues

2.2 Image & Icon Crawler

- Finds all `<img>`, `<source>`, `<link rel="icon">` tags
- Validates each URL with an HTTP HEAD request
- Reports: URL, HTTP status code, parent page, element type

2.3 Link Crawler

- Finds all `<a>` tags on the page
- Validates link destinations
- Detects redirect chains, timeouts, and SSL errors

2.4 Overlapping & Breaking Crawler

- Uses Selenium WebDriver to get computed element positions
- Detects elements that overlap other elements unexpectedly
- Finds elements that overflow their parent containers

2.5 Report Generation

- Results exported as **Excel (.xlsx)** with columns: URL, Status, Type, Parent Page
- Downloadable directly from the UI

Module 3: Accessibility Checker

Audits web pages for WCAG 2.0/2.1 compliance issues.

3.1 Checks Performed

#	Check	Severity	WCAG Criterion
1	Images without `alt` text	Violation (Critical)	1.1.1 Non-text Content
2	Form inputs without labels	Violation (Critical)	1.3.1 Info and Relationships
3	Missing `<title>` tag	Violation (Serious)	2.4.2 Page Titled
4	Missing `lang` attribute on `<html>`	Violation (Serious)	3.1.1 Language of Page
5	Missing H1 heading	Warning (Moderate)	1.3.1 Info and Relationships
6	Multiple H1 headings	Warning (Moderate)	1.3.1 Info and Relationships
7	Skipped heading levels	Warning (Moderate)	1.3.1 Info and Relationships
8	Non-descriptive link text ("click here", "read more")	Warning (Moderate)	2.4.4 Link Purpose
9	Buttons without accessible names	Violation (Critical)	4.1.2 Name, Role, Value
10	Missing viewport meta tag	Warning (Moderate)	1.4.10 Reflow

3.2 Input Options

Option	Description
Single URL	Audit one page
Sitemap URL	Provide a sitemap XML — all URLs are audited in batch
WCAG Level	Choose WCAG2A, WCAG2AA (default), or WCAG2AAA

3.3 Report Output

- **In-app results table** with issue count, severity breakdown (violations/warnings/notices)
- **PDF report** with detailed findings, element references, and recommended fixes
- **History** with paginated past results

Module 4: SEO & Performance Analyzer

Integrates with **Google PageSpeed Insights API** for real-world performance metrics.

4.1 Scores Provided

Score	Source	Range
SEO Score	PageSpeed Insights	0–100
Performance Score	Lighthouse	0–100
Accessibility Score	Lighthouse	0–100
Best Practices Score	Lighthouse	0–100

4.2 Performance Metrics (Core Web Vitals)

Metric	What it measures
Largest Contentful Paint (LCP)	Loading — when main content becomes visible
Cumulative Layout Shift (CLS)	Visual stability — how much the page shifts during load
First Contentful Paint (FCP)	When the first content pixel renders
Time to First Byte (TTFB)	Server response time
Speed Index	How quickly content is visually populated
Total Blocking Time (TBT)	Time spent blocked by long JavaScript tasks

4.3 Input Options

Option	Description
Single URL	Analyze one page
Sitemap URL	Provide a sitemap XML — all URLs analyzed in batch with parallel processing
CSV/Excel Upload	Upload a file with URLs for batch analysis
Test Type	SEO Only, Performance Only, or Both
Device	Desktop or Mobile

4.4 Report Output

- **In-app dashboard** with scores, bar charts, and pie charts

- **PDF report** with visual analysis (charts), critical issues, recommendations, and detailed metrics
- **Batch PDF report** for sitemap/CSV analysis with per-URL breakdown
- **History** with paginated past results

Module 5: Baseline Management

Baselines are the **reference images** used for visual comparison. The framework provides a full baseline lifecycle.

5.1 Features

Feature	Description
Upload Baseline	Upload a PNG reference image for any URL
Auto-Detect Baseline	If no image is uploaded, the framework automatically uses the active baseline for that URL
Promote to Baseline	After a successful comparison, promote the reference image as the new active baseline
Version History	Each baseline is versioned with timestamps and source job IDs
Rollback	Roll back to any previous baseline version
Delete Baseline	Remove baselines that are no longer needed

5.2 Baseline Workflow

```
1. Upload Figma PNG → Run comparison → Review results
2. If results are acceptable → Click "Promote to Baseline"
3. Future comparisons auto-use this baseline (no upload needed)
4. If design changes → Upload new PNG or promote new screenshot
5. Made a mistake? → Rollback to any previous version
```

Module 6: Jira Integration

Create Jira issues directly from detected visual differences — no context switching.

6.1 Setup

- **Configuration Modal** accessible from the header ("Jira Config" button)
- Enter: Jira domain, email, API token, and default project key
- Credentials stored locally in `jira_config.json`

6.2 Features

Feature	Description
One-Click Issue Creation	Click "To Jira" on any issue card to create a Jira ticket
Auto-Populated Fields	Issue title, description, and labels are auto-filled from the visual diff
Image Attachment	The cropped issue screenshot is referenced in the ticket
Available From All Tabs	Jira integration works from Visual Testing, Broken Links, Accessibility, and SEO tabs

---

Module 7: GitHub Integration & One-Click Push

7.1 GitHub Issues

Create GitHub issues directly from detected visual differences.

Feature	Description
Configuration Modal	Enter GitHub owner, repository, and personal access token
One-Click Issue Creation	"To GitHub" button on each issue card
Auto-Populates	Issue title, description, and labels from the diff context

7.2 One-Click Push to GitHub

Push all code/baseline changes to GitHub without touching the terminal.

Feature	Description
Git Status Preview	Shows current branch, number of changed files, and last commit
Custom Commit Message	Editable commit message with emoji support
Branch Selection	Push to current branch or specify a different one
Progress Tracking	Real-time progress: staging → committed → pushed
Auto-Trigger CI	Push triggers GitHub Actions workflow for automated visual tests
Error Handling	Detailed feedback if any git step fails

7.3 GitHub Actions CI/CD

Automated visual testing pipeline (`.github/workflows/visual_tests.yml`):

```
Push to GitHub
    ↓
GitHub Actions Triggered
    ↓
Install Dependencies
    ↓
Run Visual Regression Tests
    ↓
Generate Reports
    ↓
Post Results to PR/Commit
```

Module 8: History & Reporting

8.1 Visual Testing History

Feature	Description
Paginated History	Browse all past visual test runs
Filter by Status	All / Pending / Approved / Rejected
Approve / Reject	Review workflow — approve or reject each comparison
Delete	Remove old or irrelevant results
Re-Download	Access all outputs (overlay, heatmap, report) from past runs

8.2 PDF Reports

Every visual comparison generates a **downloadable PDF report** containing:

- Summary: Job ID, date, URL, SSIM score, pass/fail status
- Reference image and diff overlay screenshots
- Itemized issue list with thumbnails, categories, and locations
- Recommendations

8.3 Module-Specific History

Each module maintains its own history:

- **Broken Links History** — Past crawl results with Excel reports
- **Accessibility History** — Past audits with PDF reports
- **SEO & Performance History** — Past analyses with PDF reports

---

Technical Architecture

Folder Structure

```
Testing framework/
■■■ app.py                      # Flask server — all API endpoints (~1950 lines)
■■■ static/
■   ■■■ index.html              # Visual Testing UI
■   ■■■ broken_links.html       # Broken Links UI
■   ■■■ accessibility.html      # Accessibility UI
■   ■■■ seo_performance.html    # SEO &amp;amp; Performance UI
■   ■■■ app.js                  # Visual Testing logic
■   ■■■ broken_links.js         # Broken Links logic
■   ■■■ accessibility.js        # Accessibility logic
■   ■■■ seo_performance.js      # SEO logic
■   ■■■ git_push.js             # One-Click Push logic
■   ■■■ jira_integration.js     # Jira integration
■   ■■■ github_integration.js   # GitHub integration
■   ■■■ styles.css              # Core styles
■   ■■■ header.css              # Navigation header
■   ■■■ loader.css              # Loading animations
■   ■■■ issues.css              # Issue card styles
■   ■■■ toggle.css              # Toggle switch styles
■   ■■■ file_upload.css         # Drag-and-drop upload styles
■■■ utils/
■   ■■■ screenshot.py           # Playwright screenshot capture
■   ■■■ image_compare.py        # SSIM + OpenCV comparison engine
■   ■■■ report.py               # PDF report generator (visual testing)
■   ■■■ baseline_store.py       # Baseline version management
■   ■■■ store.py                # Job/run data persistence
■   ■■■ crawler.py              # Image, icon, and link crawlers
■   ■■■ overlapping_crawler.py  # DOM overlap detection (Selenium)
■   ■■■ pagespeed.py            # Google PageSpeed API client
■   ■■■ sitemap_parser.py       # Sitemap XML parser
■   ■■■ jira_client.py          # Jira REST API client
■   ■■■ github_client.py        # GitHub REST API client
■   ■■■ accessibility_report.py    # Accessibility PDF report
■   ■■■ seo_performance_report.py  # SEO PDF report
■   ■■■ background_validator.py    # Background color/image validation
■■■ runs/                       # All job outputs (screenshots, reports, diffs)
■■■ .github/workflows/          # GitHub Actions CI/CD config
■■■ requirements.txt            # Python dependencies
■■■ Dockerfile                  # Container support
■■■ *.md                        # Documentation files
```

Technology Stack

Layer	Technology
Backend	Python 3.9+ / Flask
Screenshot Engine	Playwright (headless Chromium)
Image Comparison	OpenCV + scikit-image SSIM
Image Processing	NumPy, Pillow
PDF Generation	ReportLab

Layer	Technology
Web Crawling	Requests + BeautifulSoup
DOM Analysis	Selenium WebDriver
SEO/Performance	Google PageSpeed Insights API
Frontend	Vanilla HTML/CSS/JavaScript
CI/CD	GitHub Actions
Containerization	Docker

API Reference

Visual Testing

Method	Endpoint	Description
`POST`	`/api/compare`	Start a visual comparison job
`GET`	`/api/status/<job_id>`	Check job progress and results
`GET`	`/api/history`	List past visual testing jobs
`DELETE`	`/api/history`	Delete specific visual testing jobs
`POST`	`/api/job/<job_id>/approve`	Approve a comparison result
`POST`	`/api/job/<job_id>/reject`	Reject a comparison result

Baselines

Method	Endpoint	Description
`GET`	`/api/baselines`	List all baselines
`POST`	`/api/baselines/upload`	Upload a new baseline image
`POST`	`/api/baselines/promote/<job_id>`	Promote a job's reference to baseline
`POST`	`/api/baselines/rollback`	Rollback to a previous baseline version
`POST`	`/api/baselines/delete`	Delete a baseline

Broken Links

Method	Endpoint	Description
`POST`	`/api/broken-links`	Start a broken links crawl
`GET`	`/api/broken-links/history`	List past crawl results
`DELETE`	`/api/broken-links/history`	Delete specific crawl results

Accessibility

Method	Endpoint	Description
`POST`	`/api/accessibility`	Start an accessibility audit
`GET`	`/api/accessibility/history`	List past audit results
`DELETE`	`/api/accessibility/history`	Delete specific audit results

SEO & Performance

Method	Endpoint	Description
`POST`	`/api/seo-performance`	Start an SEO/performance analysis
`GET`	`/api/seo-performance/history`	List past analysis results
`DELETE`	`/api/seo-performance/history`	Delete specific analysis results

Jira

Method	Endpoint	Description
`POST`	`/api/jira/config`	Save Jira configuration
`GET`	`/api/jira/config`	Get current Jira configuration
`POST`	`/api/jira/issue`	Create a Jira issue

GitHub

Method	Endpoint	Description
`POST`	`/api/github/config`	Save GitHub configuration

Method	Endpoint	Description
`GET`	`/api/github/config`	Get current GitHub configuration
`POST`	`/api/github/issue`	Create a GitHub issue
`POST`	`/api/git/push`	Push changes to GitHub
`GET`	`/api/git/status`	Get current git repository status

Utility

Method	Endpoint	Description
`GET`	`/download/<job_id>/<filename>`	Download any output file from a job

Quick Start

```
# 1. Install dependencies
pip install -r requirements.txt
playwright install chromium

# 2. Start the server
python app.py

# 3. Open the UI
# Navigate to http://localhost:7860
```

■ **Note:** This document covers all features as of the latest version. For setup-specific guides, see `QUICK_START.md`, `GITHUB_ACTIONS_SETUP.md`, and `PAGESPEED_SETUP.md`.