QA Testing Framework - Technical Specification & Patent Information

🏛️ Title of Invention

"Unified Self-Hosted Web Quality Assurance System with Automated Visual Regression Detection, Intelligent Difference Classification, and Integrated Multi-Module Testing Platform"

Short Title: Intelligent Web QA Testing Framework with Computer-Vision-Based Visual Regression Analysis

---

📄 Abstract

A computer-implemented system and method for automated web application quality assurance that unifies visual regression testing, broken link detection, accessibility auditing, and SEO performance analysis within a single self-hosted platform. The system employs a novel multi-stage visual comparison pipeline comprising: (1) GASAE-based spatial matching with rotation-constrained affine alignment for automatic image registration between reference designs and live screenshots; (2) Perceptual Layout Entropy Metric (PLEM) computation with adaptive color-aware multi-channel analysis; (3) a Detect-then-Filter methodology using Otsu auto-thresholding for region detection followed by severity-based filtering with configurable sensitivity; and (4) an intelligent difference classification engine that categorizes detected visual discrepancies into semantic categories—including spacing/margin issues, color/style mismatches, text/content differences, and missing/extra elements—using computer vision heuristics including aspect ratio analysis, cross-correlation-based shift detection, edge density measurement, and horizontal projection text line counting. The system further integrates baseline version management with rollback capability, one-click bug tracker integration, and CI/CD pipeline automation.

📚 Background of the Invention / Prior Art Analysis

Field of the Invention

This invention relates to the field of **software quality assurance**, specifically to computer-implemented methods and systems for automated visual regression testing, web accessibility compliance verification, and multi-dimensional web quality analysis.

The invention is classified under:

- **G06F 11/36** — Digital testing or debugging of software
- **G06T 7/00** — Image analysis and processing
- **G06F 16/958** — World Wide Web content analysis

Description of the Problem

Modern web development teams face a critical challenge: ensuring visual fidelity, functional integrity, accessibility compliance, and performance optimization across frequent deployment cycles. As web applications grow in complexity, manual quality assurance becomes impractical, error-prone, and unable to scale with continuous delivery pipelines. Visual regressions—unintended changes to the appearance of a web page—are among the most common and difficult-to-detect defects, often escaping automated unit and integration tests.

Existing Solutions and Their Limitations

Several prior art solutions exist in the domain of visual regression testing:

1. Percy (BrowserStack Visual Testing)

- **What it does:** Cloud-based visual testing service that captures screenshots and compares them using pixel diffing.
- **Limitations:**
  - ❌ **No intelligent classification** — Reports raw pixel differences without categorizing them (spacing vs. color vs. content vs. layout). Developers must manually interpret every flagged region.
  - ❌ **Cloud-only / SaaS dependency** — Requires uploading all screenshots to BrowserStack's servers, creating data privacy concerns and vendor lock-in.
  - ❌ **Single-purpose** — Only performs visual testing. Teams need separate tools for accessibility, broken links, and SEO analysis.
  - ❌ **No Figma-to-live comparison** — Cannot compare design mockups (Figma PNGs) directly against live pages; limited to baseline-vs-current screenshot diffing.
  - ❌ **Cost prohibitive** — Pricing starts at $599/month for teams, with per-screenshot billing beyond plan limits.

2. Applitools Eyes

- **What it does:** AI-powered visual testing using proprietary "Visual AI" for intelligent comparison.
- **Limitations:**
  - ❌ **Proprietary black box** — The AI classification model is closed-source and non-customizable. Users cannot adjust, inspect, or extend the classification heuristics.
  - ❌ **Cloud-only** — All images are processed on Applitools' servers. No self-hosted option exists.
  - ❌ **No unified QA platform** — Focused exclusively on visual testing. Does not integrate accessibility audits, broken link detection, or SEO performance analysis.
  - ❌ **No GASAE-based alignment** — Does not use feature-matching-based image registration; relies on viewport-level matching which fails on slight rendering offsets.
  - ❌ **Enterprise pricing** — Annual contracts starting at $10,000+ for small teams.

3. BackstopJS (Open Source)

- **What it does:** Open-source visual regression testing tool using Legacy Automation Tools for screenshot capture and pixel comparison.
- **Limitations:**
  - ❌ **No intelligent diff classification** — Only generates raw pixel-diff overlays. Cannot distinguish between spacing issues, color changes, or missing elements.
  - ❌ **No severity-based filtering** — Uses a single global mismatch threshold. Cannot apply the Detect-then-Filter methodology where all regions are found first and then filtered by severity.
  - ❌ **CLI-only** — No browser-based UI for visual review, issue management, or baseline promotion workflows.
  - ❌ **No multi-module support** — Only handles visual regression. No built-in accessibility, broken links, or SEO testing capabilities.
  - ❌ **No image alignment** — Does not perform GASAE-based or any feature-matching alignment between images, leading to false positives from minor rendering shifts.

4. Chromatic (Storybook)

- **What it does:** Visual testing tool tightly integrated with Storybook for component-level visual regression testing.

- **Limitations:**
  - ❌ **Storybook dependency** — Only works with projects using Storybook. Cannot test arbitrary URLs or full-page layouts.
  - ❌ **No full-page comparison** — Designed for isolated component snapshots, not full-page visual regression testing.
  - ❌ **No smart classification** — Reports all visual changes as a single undifferentiated category without semantic understanding.
  - ❌ **Cloud-hosted** — Requires uploading component renders to Chromatic's infrastructure.
  - ❌ **No cross-domain QA** — Does not offer accessibility, SEO, or broken link checking capabilities.

Summary of Prior Art Deficiencies

Limitation	Percy	Applitools	BackstopJS	Chromatic	**This Invention**
Self-hosted / data sovereignty	❌	❌	✅	❌	✅
Intelligent diff classification	❌	⚠️ Proprietary	❌	❌	✅ Open & customizable
Detect-then-Filter sensitivity	❌	❌	❌	❌	✅
GASAE-based image alignment	❌	❌	❌	❌	✅
Cross-correlation shift detection	❌	❌	❌	❌	✅
Figma-to-live comparison	❌	❌	❌	❌	✅
Unified multi-module QA platform	❌	❌	❌	❌	✅
Baseline versioning with rollback	⚠️ Limited	⚠️ Limited	❌	❌	✅ Full lifecycle
Cost	$599+/mo	$10K+/yr	Free	$149+/mo	Free / self-hosted

Novelty Statement

The present invention addresses all of the above deficiencies through a combination of novel technical approaches not found individually or collectively in the prior art:

1. **Intelligent Difference Classification Engine** — A deterministic, rule-based computer vision pipeline that automatically categorizes visual differences into semantic categories (spacing, color, content, layout, missing/extra elements) using aspect ratio analysis, cross-correlation-based vertical shift detection, horizontal projection text line counting, edge density measurement, and local PLEM scoring. Unlike Applitools' proprietary AI, this system is transparent, auditable, and customizable.

2. **Detect-then-Filter Sensitivity Methodology** — A two-phase approach where all candidate difference regions are first detected using Otsu auto-thresholding (ensuring consistent detection regardless of sensitivity settings), then filtered by a severity threshold derived from the user's sensitivity preference. This guarantees a monotonic relationship: higher sensitivity always produces equal or more issues than lower sensitivity — a property not achieved by single-threshold approaches used in prior art.

3. **Rotation-Constrained GASAE Alignment** — A novel application of GASAE spatial matching with partial affine estimation where the rotation component is explicitly zeroed (since web screenshots are never rotated), while preserving translation and bounded scaling. This prevents perspective distortion artifacts that occur with full homography-based alignment.

4. **Unified Self-Hosted Multi-Module QA Platform** — The first self-hosted system to integrate visual regression testing, broken link/asset crawling, WCAG accessibility auditing, and SEO/performance analysis within a single application with shared history, reporting, and bug tracker integration.

⚖️ Patent Claims

The following claims define the scope of protection sought by this invention.

Independent Claims

Claim 1. A computer-implemented method for automated visual regression testing of web applications, comprising:

- (a) capturing a live screenshot of a target web page at a specified URL using a headless browser engine;
- (b) receiving a reference image representing an intended visual design of the target web page;
- (c) aligning the live screenshot to the reference image using Grid-Aware Spatial Anchor Extraction (GASAE) feature matching with rotation-constrained partial affine transformation, wherein the rotation component of the affine matrix is explicitly set to zero while preserving translation and bounded scaling parameters;
- (d) computing a Perceptual Layout Entropy Metric (PLEM) score between the aligned images;
- (e) detecting candidate difference regions using Otsu auto-thresholding on the PLEM difference map, followed by morphological dilation and contour detection;
- (f) filtering the candidate difference regions by computing a mean pixel difference intensity for each region and retaining only regions exceeding a severity threshold derived from a user-configurable sensitivity parameter; and
- (g) classifying each retained difference region into a semantic category selected from the group consisting of: section spacing issue, column spacing issue, section spacing mismatch, padding/margin difference, color/style mismatch, text/content mismatch, missing content, extra content, missing element, extra element, and layout mismatch.

Claim 6. A system for intelligent classification of visual differences in web page screenshots, comprising:

- a processor configured to execute instructions that, for each detected difference region between a reference image and a live screenshot:
  - (a) extract a cropped sub-image from both the reference and live images at the coordinates of the difference region;
  - (b) determine whether the region represents missing or extra content by comparing text line counts obtained through horizontal projection analysis with Otsu binarization;

- (c) determine whether the region represents a missing or extra element by comparing mean grayscale intensity values of the crops against a whitespace threshold;
- (d) determine whether the region represents a spacing or margin issue by computing the aspect ratio of the region and analyzing the standard deviation of grayscale values within the crop;
- (e) determine whether the region represents a section spacing mismatch by computing the normalized cross-correlation of horizontal projection profiles between the two crops, and identifying a vertical shift exceeding a minimum displacement threshold with a correlation coefficient above a similarity threshold;
- (f) determine whether the region represents a padding or margin difference by dividing the crop vertically into thirds and comparing the standard deviation of the top and bottom thirds against the middle third;
- (g) determine whether the region represents a color or style mismatch by computing a local PLEM score between the crops and identifying structural similarity above 0.90 with differing pixel values; and
- (h) determine whether the region represents a text or content mismatch by computing the edge density of the reference crop using Canny edge detection and identifying edge density above 5%.

Claim 11. A method for configurable sensitivity-based visual difference detection in web application testing, comprising:

- (a) generating a grayscale difference map between a reference image and a test image;
- (b) detecting all candidate difference regions from the difference map using Otsu auto-thresholding, Gaussian blur preprocessing, binary thresholding, morphological dilation with a configurable iteration parameter, and contour extraction with minimum area filtering;
- (c) receiving a sensitivity parameter from a user interface, the sensitivity parameter representing a value on a continuous scale;
- (d) computing a severity threshold as a monotonically decreasing function of the sensitivity parameter, such that higher sensitivity values produce lower severity thresholds;
- (e) for each candidate difference region, computing the mean pixel difference intensity within the region's bounding rectangle on the difference map;
- (f) retaining only those candidate difference regions whose mean pixel difference intensity equals or exceeds the computed severity threshold; and
- (g) guaranteeing a monotonic relationship wherein a higher sensitivity parameter always produces an equal or greater number of retained difference regions compared to any lower sensitivity parameter value.

Claim 15. A method for automatic image alignment in visual regression testing of web pages, comprising:

- (a) detecting keypoints and computing descriptors in both a reference image and a test image using an GASAE anchor detector configured with a specified maximum number of features, scale factor, and edge threshold;
- (b) matching descriptors between the two images using a brute-force Hamming distance matcher with k-nearest-neighbor matching and Lowe's ratio test filtering;
- (c) estimating a partial affine transformation matrix from the matched keypoints using RANSAC with a specified reprojection threshold;
- (d) extracting scale components from the estimated transformation matrix and validating that the scale falls within a bounded range centered on 1.0;
- (e) reconstructing the transformation matrix with the rotation component explicitly set to zero, the validated scale preserved, and translation components preserved; and

- (f) applying the reconstructed transformation to warp the test image to align with the reference image's coordinate space.

Claim 18. A self-hosted web quality assurance platform, comprising:

- a backend server providing a unified API for executing and managing multiple quality assurance testing modules, the modules including:
  - (a) a visual regression testing module that compares reference images against live web page screenshots using the method of Claim 1;
  - (b) a broken link and asset detection module that crawls web pages to identify broken hyperlinks, missing images, and unreachable resources;
  - (c) an accessibility auditing module that evaluates web pages against WCAG 2.0/2.1 compliance criteria at configurable conformance levels;
  - (d) an SEO and performance analysis module that integrates with external performance measurement APIs to evaluate page load metrics; and
  - (e) a baseline management module that maintains versioned reference images with support for promotion, rollback, and version history tracking;
- a browser-based user interface that provides unified access to all testing modules with shared navigation, history viewing, and report generation; and
- integration interfaces for creating bug reports in external issue tracking systems directly from detected issues, with auto-populated issue details and attached evidence images.

Dependent Claims

Claim 2. The method of Claim 1, wherein step (a) further comprises:

- disabling all CSS animation and transition properties on the target web page before screenshot capture;
- setting the text cursor caret-color to transparent to prevent cursor artifacts; and
- waiting a configurable delay period to allow lazy-loaded content and JavaScript execution to complete.

Claim 3. The method of Claim 1, wherein step (b) comprises receiving a reference image from one of: a user-uploaded Figma design PNG, or an automatically retrieved active baseline image associated with the target URL from a baseline version store.

Claim 4. The method of Claim 1, wherein step (c) further comprises:

- after alignment, transforming the corners of the test image through the affine matrix to determine the overlapping region in the reference image's coordinate space;
- cropping both the reference image and the aligned test image to the intersection of their valid regions; and
- thereby limiting the comparison to the overlapping area to prevent false positives from size mismatches between images of different dimensions.

Claim 5. The method of Claim 1, further comprising:

- generating a visual diff overlay image by creating a red color layer and blending it with the live screenshot using a mask derived from the difference map, where the mask is intersected with a region mask containing only the retained difference regions; and
- generating a heatmap image by applying a color map transformation to the grayscale difference map.

Claim 7. The system of Claim 6, wherein the text line counting in step (b) comprises:

- applying Otsu binarization to invert the grayscale crop such that text content appears as white pixels on a black background;
- computing a horizontal projection by summing pixel values along each row;
- identifying content rows as those where the horizontal projection sum exceeds 2% of the row width multiplied by the maximum pixel value; and
- counting the number of contiguous groups of content rows as distinct text lines.

Claim 8. The system of Claim 6, wherein the aspect ratio analysis in step (d) determines:

- a horizontal spacing strip when the width-to-height ratio exceeds 4:1, indicating a gap between vertically stacked sections; and
- a vertical spacing strip when the width-to-height ratio is less than 1:4, indicating a gap between horizontally adjacent elements.

Claim 9. The system of Claim 6, wherein the cross-correlation analysis in step (e) comprises:

- computing row-wise mean grayscale values to produce a horizontal projection profile for each crop;
- normalizing each projection profile by subtracting its mean and dividing by its L2 norm;
- computing the full cross-correlation of the two normalized profiles;
- identifying the shift corresponding to the maximum cross-correlation coefficient; and
- classifying the region as a section spacing mismatch when the maximum correlation coefficient exceeds 0.6 and the absolute shift exceeds 2 pixels.

Claim 10. The system of Claim 6, wherein the classification steps are applied in a priority order such that: missing/extra content detection takes precedence over missing/extra element detection, which takes precedence over spacing detection, which takes precedence over color/style detection, which takes precedence over text/content detection, with layout mismatch serving as a fallback classification.

Claim 12. The method of Claim 11, wherein the severity threshold is computed as: `severity_min = max(3, floor(130 × (1 − sensitivity / 100)))` , where sensitivity is expressed as a percentage from 0 to 100.

Claim 13. The method of Claim 11, wherein the morphological dilation iteration parameter in step (b) is selected based on the sensitivity parameter such that:

- sensitivity ≥ 80%: 1 iteration, producing more separate, granular regions;
- 50% ≤ sensitivity < 80%: 2 iterations, producing standard region merging; and
- sensitivity < 50%: 3 iterations, producing aggressive merging into fewer, larger regions.

Claim 14. The method of Claim 11, further comprising:

- receiving one or more validation type filter selections from the user, the filter types including layout and structure, text and content, and colors and styles;
- after classification of each retained region, excluding regions whose classification category does not match any of the selected validation type filters; and
- thereby allowing the user to focus the analysis on specific categories of visual differences.

Claim 16. The method of Claim 15, wherein the bounded range in step (d) constrains the scale to between 0.9 and 1.1, and alignment is abandoned in favor of simple image resizing when the scale falls outside this range.

Claim 17. The method of Claim 15, wherein the GASAE anchor detector is configured with 5000 maximum features, a scale factor of 1.2, an edge threshold of 15, and a patch size of 31, and wherein the ratio test in step (b) uses a threshold of 0.75 and requires a minimum of 8 good matches to proceed with transformation estimation.

Claim 19. The platform of Claim 18, wherein the broken link and asset detection module further comprises:

- an image and icon crawler that validates HTTP status codes of all image, source, and icon link elements on a web page; and
- an overlapping element detector that uses a browser automation engine to compute element bounding rectangles and identify elements that overlap other elements or overflow their parent containers.

Claim 20. The platform of Claim 18, further comprising:

- a one-click code push module that displays current git repository status, accepts a custom commit message, stages all changes, commits, and pushes to a specified remote branch; and
- a CI/CD trigger mechanism wherein the push automatically initiates a continuous integration pipeline that executes visual regression tests and generates reports.

🔬 Detailed Description of Preferred Embodiments (Algorithm Specifications)

The following section provides formal algorithmic descriptions of the novel methods claimed in this invention.

Algorithm 1: Rotation-Constrained GASAE Alignment

Purpose: Automatically align a live web page screenshot to a reference design image, correcting for minor positional offsets while preventing rotation and perspective distortion artifacts.

Pseudocode:

```
FUNCTION AlignImages(reference, test_image):
    INPUT:  reference    — BGR image (H₁ × W₁ × 3)
            test_image   — BGR image (H₂ × W₂ × 3)
    OUTPUT: aligned_image — BGR image (H₁ × W₁ × 3)
            transform_M  — 2×3 affine matrix or NULL

    1.  gray_ref  ← ConvertToGrayscale(reference)
    2.  gray_test ← ConvertToGrayscale(test_image)

    3.  gasae ← GASAE_Initialize(nfeatures=5000, scaleFactor=1.2,
                        edgeThreshold=15, patchSize=31)

    4.  kp₁, desc₁ ← gasae.DetectAndCompute(gray_ref)
    5.  kp₂, desc₂ ← gasae.DetectAndCompute(gray_test)

    6.  IF desc₁ = NULL OR desc₂ = NULL OR |kp₁| < 10 OR |kp₂| < 10:
            RETURN Resize(test_image, W₁, H₁), NULL

    7.  matcher ← BFMatcher(NORM_HAMMING, crossCheck=False)
    8.  raw_matches ← matcher.KnnMatch(desc₁, desc₂, k=2)

    9.  good_matches ← ∅
    10. FOR EACH (m, n) IN raw_matches:
            IF m.distance < 0.75 × n.distance:        // Lowe's ratio test
```

```
                good_matches ← good_matches ∪ {m}

    11. IF |good_matches| < 8:
            RETURN Resize(test_image, W₁, H₁), NULL

    12. src_pts ← [kp₁[m.queryIdx].pt FOR m IN good_matches]
    13. dst_pts ← [kp₂[m.trainIdx].pt FOR m IN good_matches]

    14. M, inliers ← EstimateAffinePartial2D(dst_pts, src_pts,
                                             method=RANSAC,
                                             threshold=5.0)

    15. IF M = NULL:
            RETURN Resize(test_image, W₁, H₁), NULL

    // ── Rotation Suppression (Novel Step) ──
    // M = [[α, β, t_x], [−β, α, t_y]] where α = s·cos(θ), β = s·sin(θ)

    16. s_x ← √(M[0,0]² + M[0,1]²)
    17. s_y ← √(M[1,0]² + M[1,1]²)
    18. scale ← (s_x + s_y) / 2

    19. IF scale < 0.9 OR scale > 1.1:                // Scale sanity check
            RETURN Resize(test_image, W₁, H₁), NULL

    20. M[0,0] ← scale    // Replace α with scale (remove rotation)
    21. M[0,1] ← 0         // Set β to 0 (zero rotation)
    22. M[1,0] ← 0         // Set −β to 0
    23. M[1,1] ← scale     // Replace α with scale
    // Translation M[0,2] (t_x) and M[1,2] (t_y) are preserved

    24. aligned ← WarpAffine(test_image, M, (W₁, H₁), INTER_LINEAR)
    25. RETURN aligned, M
```

Mathematical Foundation:

The partial affine transformation matrix is:

```
M = | s·cos(θ)   s·sin(θ)   t_x |
    | −s·sin(θ)  s·cos(θ)   t_y |
```

After rotation suppression (θ = 0):

```
M' = | s    0    t_x |
     | 0    s    t_y |
```

Where:

- `s` = scale factor, constrained to [0.9, 1.1]
- `t_x`, `t_y` = horizontal and vertical translation (preserved from RANSAC estimation)
- `θ` = rotation angle (forced to 0, since web screenshots have no rotation)

Algorithm 2: Detect-then-Filter Region Detection

Purpose: Detect all candidate visual difference regions using a consistent auto-threshold method, then apply severity-based filtering to achieve configurable sensitivity with guaranteed monotonic behavior.

Pseudocode:

```
FUNCTION DetectAndFilter(diff_map, sensitivity):
    INPUT:  diff_map    — Grayscale difference image (H × W), values [0, 255]
            sensitivity — User parameter [0, 100]
    OUTPUT: filtered_regions — List of (x, y, w, h) bounding rectangles

    // ── Phase 1: Consistent Detection (independent of sensitivity) ──

    1.  severity_min ← max(3, ⌊130 × (1 − sensitivity / 100)⌋)
    2.  min_area     ← max(20, ⌊600 × (1 − sensitivity / 100)⌋)

    3.  IF sensitivity ≥ 80: dilate_iter ← 1
        ELIF sensitivity ≥ 50: dilate_iter ← 2
        ELSE: dilate_iter ← 3

    4.  blurred ← GaussianBlur(diff_map, kernel=(3,3), σ=0)

    5.  _, binary ← OtsuThreshold(blurred)
        // Otsu automatically selects the optimal threshold T that
        // minimizes intra-class variance: σ²_w(T) = w₁(T)·σ²₁(T) + w₂(T)·σ²₂(T)

    6.  kernel ← Ones(3, 3)
    7.  dilated ← MorphDilate(binary, kernel, iterations=dilate_iter)

    8.  contours ← FindContours(dilated, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE)

    9.  candidates ← ∅
    10. FOR EACH contour IN contours:
            (x, y, w, h) ← BoundingRect(contour)
            IF w × h ≥ min_area:
                candidates ← candidates ∪ {(x, y, w, h)}

    // ── Phase 2: Severity Filtering (sensitivity-dependent) ──

    11. filtered_regions ← ∅
    12. FOR EACH (x, y, w, h) IN candidates:
            region_pixels ← diff_map[y : y+h, x : x+w]
            mean_severity ← Mean(region_pixels)
            IF mean_severity ≥ severity_min:
                filtered_regions ← filtered_regions ∪ {(x, y, w, h)}

    13. RETURN filtered_regions
```

Monotonic Guarantee Proof:

Let $S_1 < S_2$ be two sensitivity values.

- `severity_min(S₁) = max(3, ⌊130 × (1 − S₁/100)⌋)`
- `severity_min(S₂) = max(3, ⌊130 × (1 − S₂/100)⌋)`

Since $S_1 < S_2$:

- `(1 − S₁/100) > (1 − S₂/100)`
- `severity_min(S₁) ≥ severity_min(S₂)`

Therefore, every region that passes the filter at sensitivity S_1 also passes at sensitivity S_2, proving: **$|\text{filtered\_regions}(S_2)| \geq |\text{filtered\_regions}(S_1)| \; \forall \; S_1 < S_2$ ∎**

Algorithm 3: Intelligent Difference Classification Engine

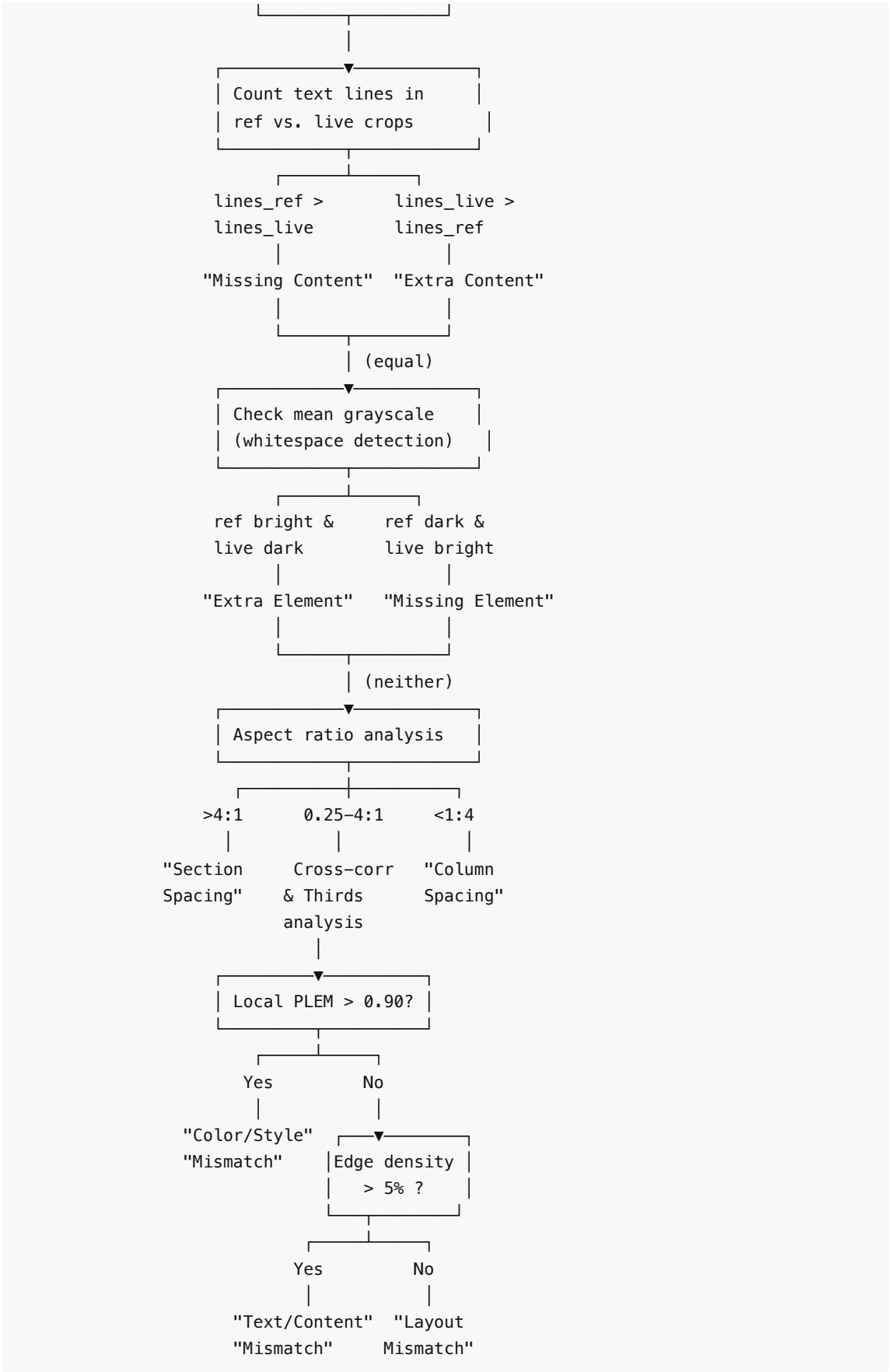
Purpose: Automatically categorize each detected visual difference into a human-understandable semantic category using deterministic computer vision heuristics.

Pseudocode:

```
FUNCTION ClassifyDifference(ref_crop, live_crop):
    INPUT:  ref_crop  — BGR sub-image from reference (h × w × 3)
            live_crop — BGR sub-image from live screenshot (h × w × 3)
    OUTPUT: category  — String label identifying the type of visual difference

    1.  gray_ref  ← ConvertToGrayscale(ref_crop)
    2.  gray_live ← ConvertToGrayscale(live_crop)
    3.  (h, w) ← Dimensions(ref_crop)

    // ── Priority 1: Missing/Extra Content (Text Line Count) ──

    4.  lines_ref  ← CountTextLines(gray_ref)      // See Algorithm 4
    5.  lines_live ← CountTextLines(gray_live)
    6.  IF lines_ref > lines_live: RETURN "Missing Content (Text/List)"
    7.  IF lines_live > lines_ref: RETURN "Extra Content (Text/List)"

    // ── Priority 2: Missing/Extra Element (Whitespace Check) ──

    8.  mean_ref  ← Mean(gray_ref)
    9.  mean_live ← Mean(gray_live)
    10. IF mean_ref > 250 AND mean_live < 245: RETURN "Extra Element"
    11. IF mean_ref < 245 AND mean_live > 250: RETURN "Missing Element"

    // ── Priority 3: Spacing & Gap Detection ──

    12. aspect_ratio ← w / h

    13. IF aspect_ratio > 4:                        // Wide & short → horizontal gap
            σ_ref ← StdDev(gray_ref)
            σ_live ← StdDev(gray_live)
            IF σ_ref < 20 OR σ_live < 20:
                RETURN "Section Spacing / Margin Issue (height: {h}px)"
            RETURN "Spacing/Padding Issue"

    14. IF aspect_ratio < 0.25:                     // Tall & narrow → vertical gap
```

```
            σ_ref ← StdDev(gray_ref)
            σ_live ← StdDev(gray_live)
            IF σ_ref < 20 OR σ_live < 20:
                RETURN "Column Spacing / Gap Issue (width: {w}px)"
            RETURN "Spacing/Padding Issue"

    // ── Priority 3b: Cross-Correlation Shift Detection ──

    15. IF h > 20 AND w > 20:
            proj_ref  ← RowMeans(gray_ref)          // Horizontal projection profile
            proj_live ← RowMeans(gray_live)

            p₁ ← (proj_ref − Mean(proj_ref)) / ‖proj_ref − Mean(proj_ref)‖₂
            p₂ ← (proj_live − Mean(proj_live)) / ‖proj_live − Mean(proj_live)‖₂

            correlation ← CrossCorrelate(p₁, p₂, mode='full')
            best_shift ← ArgMax(correlation) − (len(p₁) − 1)
            peak_corr  ← Max(correlation)

            IF peak_corr > 0.6 AND |best_shift| > 2:
                RETURN "Section Spacing Mismatch (~{|best_shift|}px shift)"

    // ── Priority 3c: Padding/Margin (Thirds Analysis) ──

    16. third_h ← max(1, ⌊h / 3⌋)
        σ_top ← StdDev(gray_ref[0 : third_h, :])
        σ_bot ← StdDev(gray_ref[h−third_h : h, :])
        σ_mid ← StdDev(gray_ref[third_h : h−third_h, :])

    17. IF (σ_top < 10 OR σ_bot < 10) AND σ_mid > 25:
            uniform_area ← "top" IF σ_top < σ_bot ELSE "bottom"
            RETURN "Padding/Margin Difference ({uniform_area})"

    // ── Priority 4: Color/Style Mismatch (Local PLEM) ──

    18. local_ssim ← PLEM(gray_ref, gray_live)
    19. IF local_ssim > 0.90: RETURN "Color/Style Mismatch"

    // ── Priority 5: Text/Content Mismatch (Edge Density) ──

    20. edges ← CannyEdgeDetector(gray_ref, low=50, high=150)
    21. density ← CountNonZero(edges) / TotalPixels(edges)
    22. IF density > 0.05: RETURN "Text/Content Mismatch"

    // ── Fallback ──

    23. RETURN "Layout Mismatch"
```

Classification Decision Tree:


Count text lines in ref vs. live crops
lines_ref > lines_live
lines_live > lines_ref
"Missing Content"
"Extra Content"
(equal)
Check mean grayscale (whitespace detection)
ref bright & live dark
ref dark & live bright
"Extra Element"
"Missing Element"
(neither)
Aspect ratio analysis
>4:1
0.25–4:1
<1:4
"Section Spacing"
Cross-corr & Thirds analysis
"Column Spacing"
Local PLEM > 0.90?
Yes
No
"Color/Style" "Mismatch"
Edge density > 5% ?
Yes
No
"Text/Content" "Mismatch"
"Layout Mismatch"

Algorithm 4: Horizontal Projection Text Line Counter

Purpose: Count the number of distinct text lines in a grayscale image crop using horizontal projection analysis.

Pseudocode:

```
FUNCTION CountTextLines(gray_image):
    INPUT:  gray_image — Grayscale image (h × w)
    OUTPUT: line_count — Integer count of distinct text lines

    1.  _, binary ← OtsuThreshold(gray_image, mode=BINARY_INVERSE)
        // Inverts so text = white (255), background = black (0)

    2.  h_proj ← RowSum(binary)                 // Sum pixels along each row
        // h_proj[i] = Σⱼ binary[i][j]

    3.  IF Max(h_proj) = 0: RETURN 0

    4.  width ← Columns(gray_image)
    5.  content_threshold ← width × 255 × 0.02  // 2% of max possible row sum

    6.  has_content[i] ← (h_proj[i] > content_threshold) FOR EACH row i

    7.  line_count ← 0
    8.  in_line ← False

    9.  FOR i ← 0 TO Length(has_content) − 1:
            IF has_content[i] AND NOT in_line:
                line_count ← line_count + 1
                in_line ← True
            ELIF NOT has_content[i] AND in_line:
                in_line ← False

    10. RETURN line_count
```

Algorithm 5: Smart Crop — Overlap-Based Comparison Region Extraction

Purpose: After alignment, determine the valid overlapping region between the reference and aligned test images, and crop both images to this region to prevent false positives from non-overlapping areas.

Pseudocode:

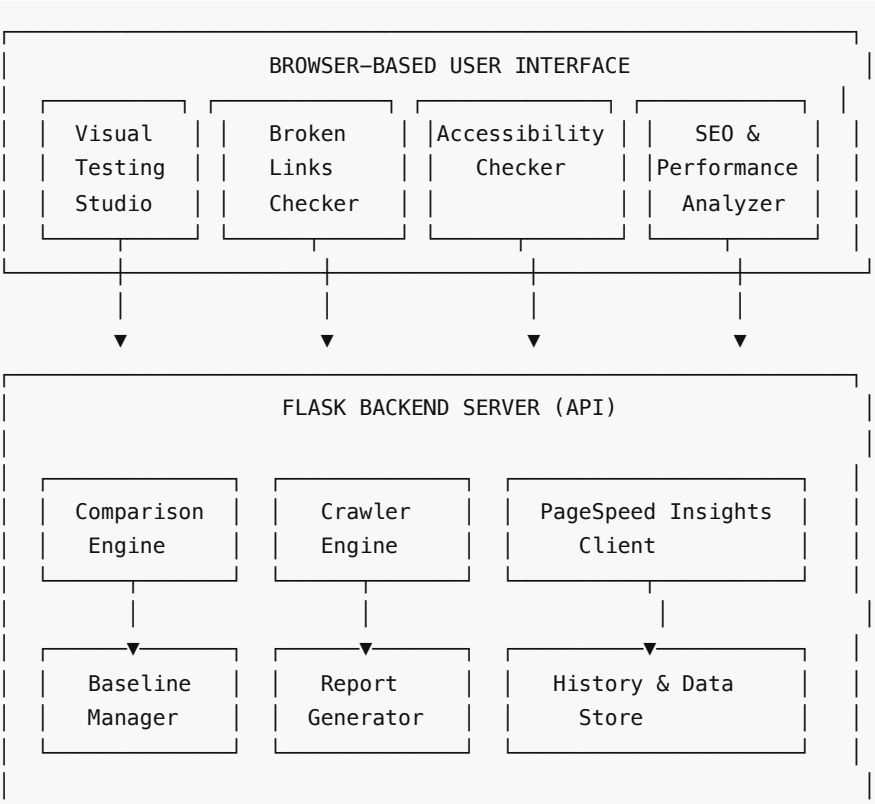
```
FUNCTION SmartCrop(reference, aligned_test, transform_M, original_test):
    INPUT:  reference      — BGR image (H₁ × W₁ × 3)
            aligned_test   — BGR image (H₁ × W₁ × 3)
            transform_M    — 2×3 affine matrix
            original_test  — BGR image (H₂ × W₂ × 3)
    OUTPUT: ref_cropped    — BGR image (H' × W' × 3)
            test_cropped   — BGR image (H' × W' × 3)

    1.  (H₂, W₂) ← Dimensions(original_test)
```

```
    2.  (H₁, W₁) ← Dimensions(reference)

    3.  corners ← [[0, 0], [W₂, 0], [W₂, H₂], [0, H₂]]

    4.  transformed_corners ← AffineTransform(corners, transform_M)
        // Maps test image corners into reference coordinate space

    5.  x_min ← max(0, ⌊Min(transformed_corners[:, 0])⌋)
    6.  y_min ← max(0, ⌊Min(transformed_corners[:, 1])⌋)
    7.  x_max ← min(W₁, ⌈Max(transformed_corners[:, 0])⌉)
    8.  y_max ← min(H₁, ⌈Max(transformed_corners[:, 1])⌉)

    9.  IF x_max > x_min AND y_max > y_min:
            ref_cropped  ← reference[y_min:y_max, x_min:x_max]
            test_cropped ← aligned_test[y_min:y_max, x_min:x_max]
        ELSE:
            ref_cropped  ← reference
            test_cropped ← aligned_test

    10. RETURN ref_cropped, test_cropped
```

📐 Description of Figures

The following figures illustrate the system architecture and key processes of the invention. In a formal patent filing, these would be rendered as professional technical drawings.

Figure 1 — High-Level System Architecture



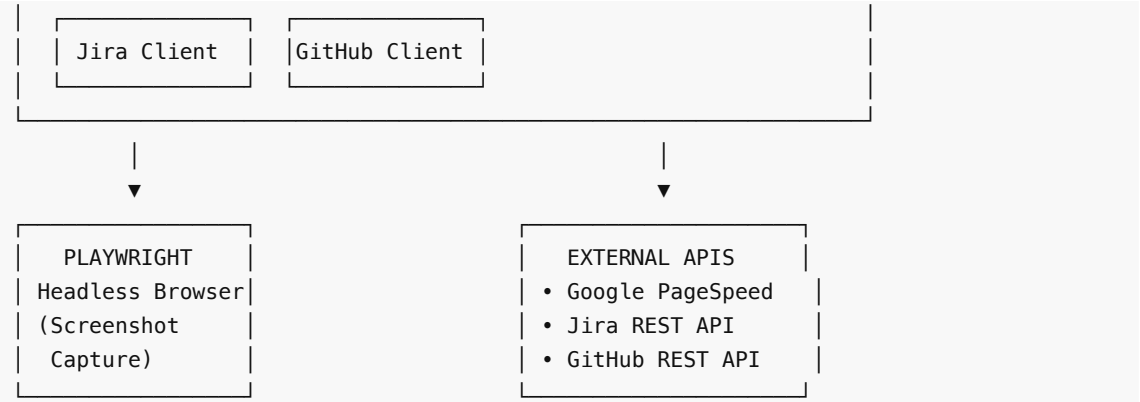
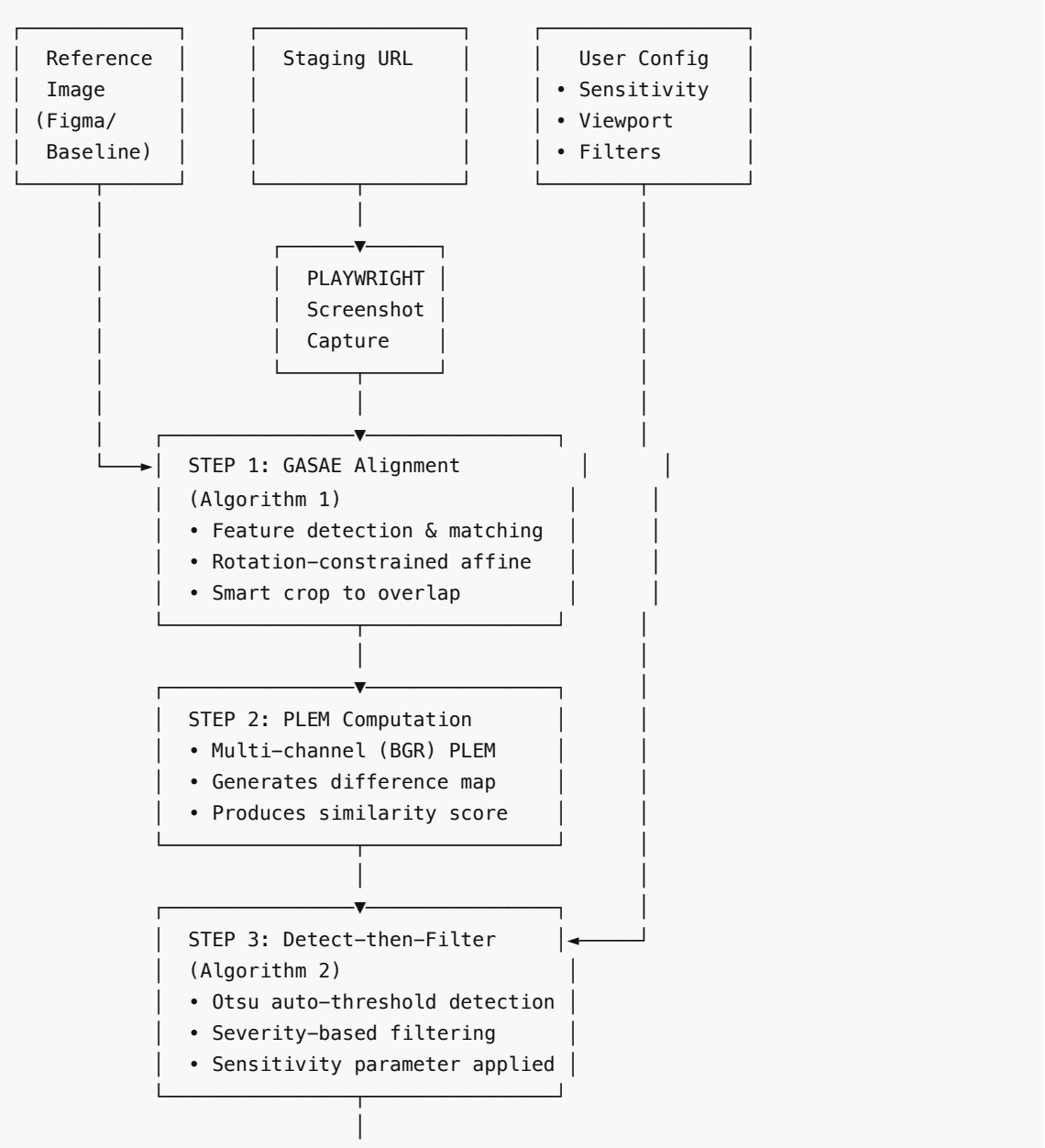


Figure 2 — Visual Comparison Pipeline (End-to-End Flow)



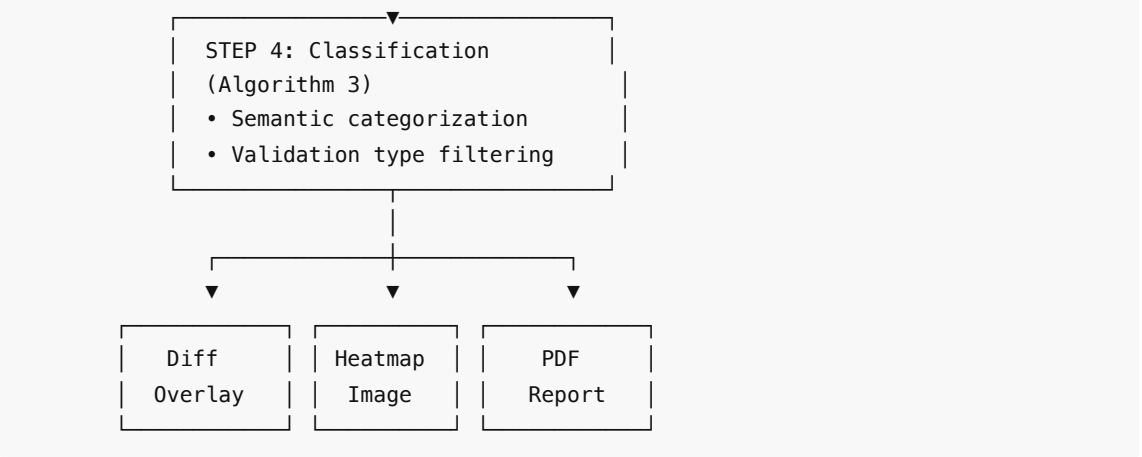


Figure 3 — Classification Decision Tree

(See Algorithm 3 above for the complete decision tree diagram)

Figure 4 — Baseline Management Lifecycle

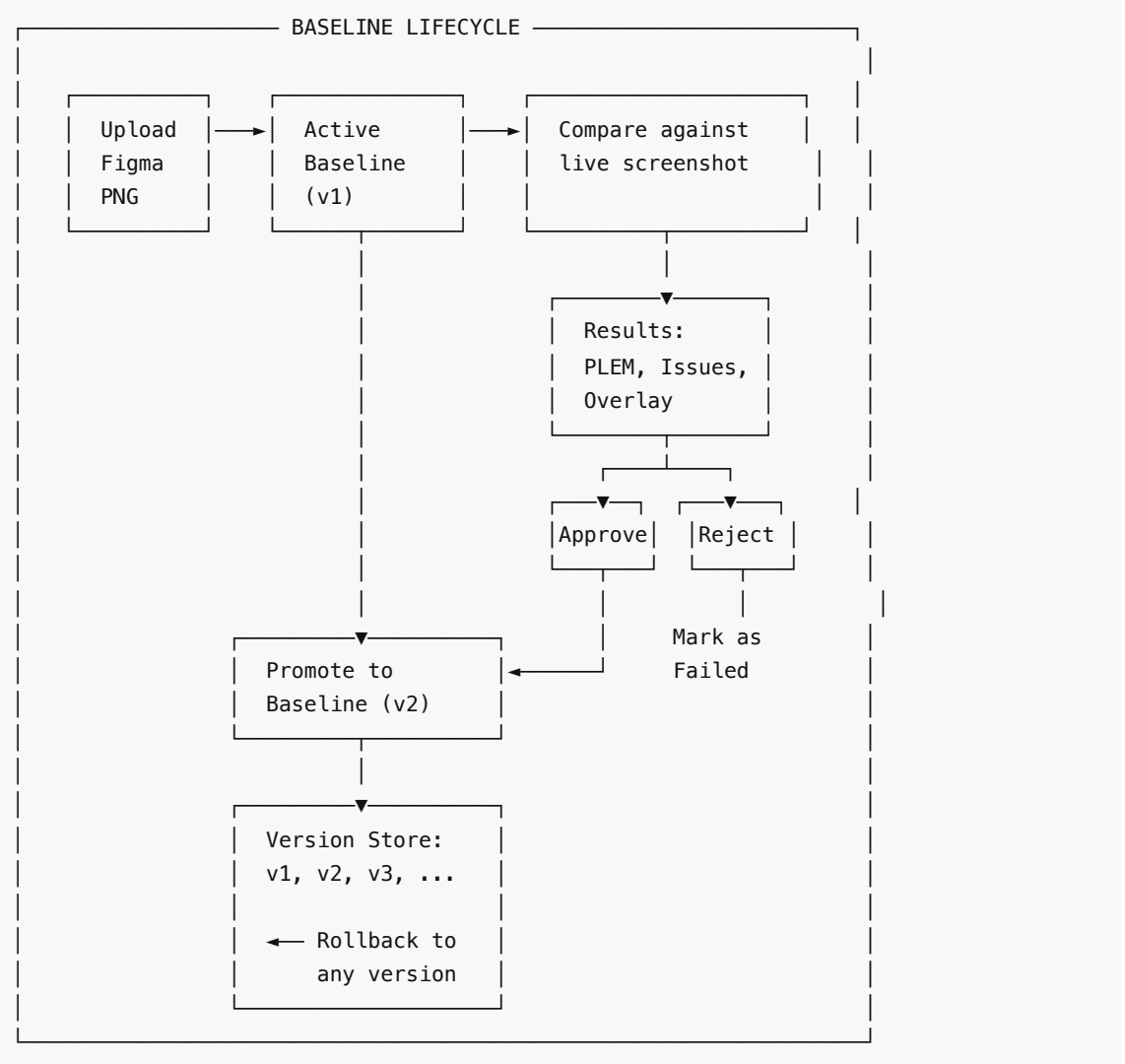


Figure 5 — Sensitivity Parameter Effect on Detection

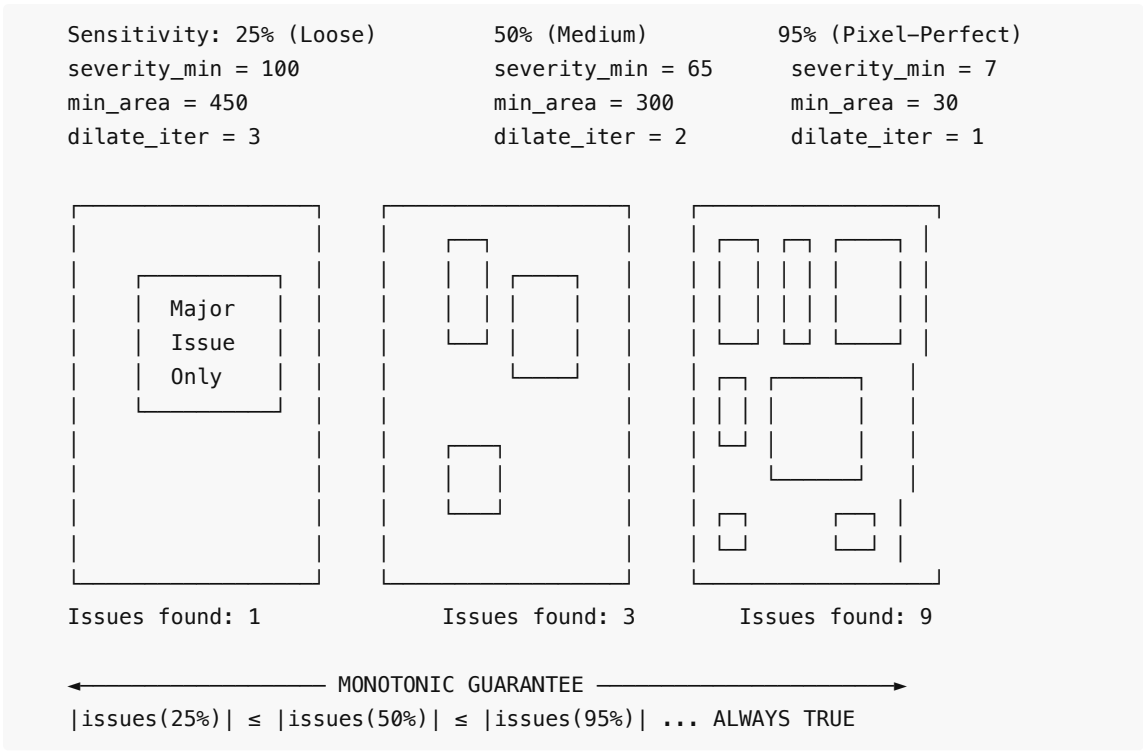
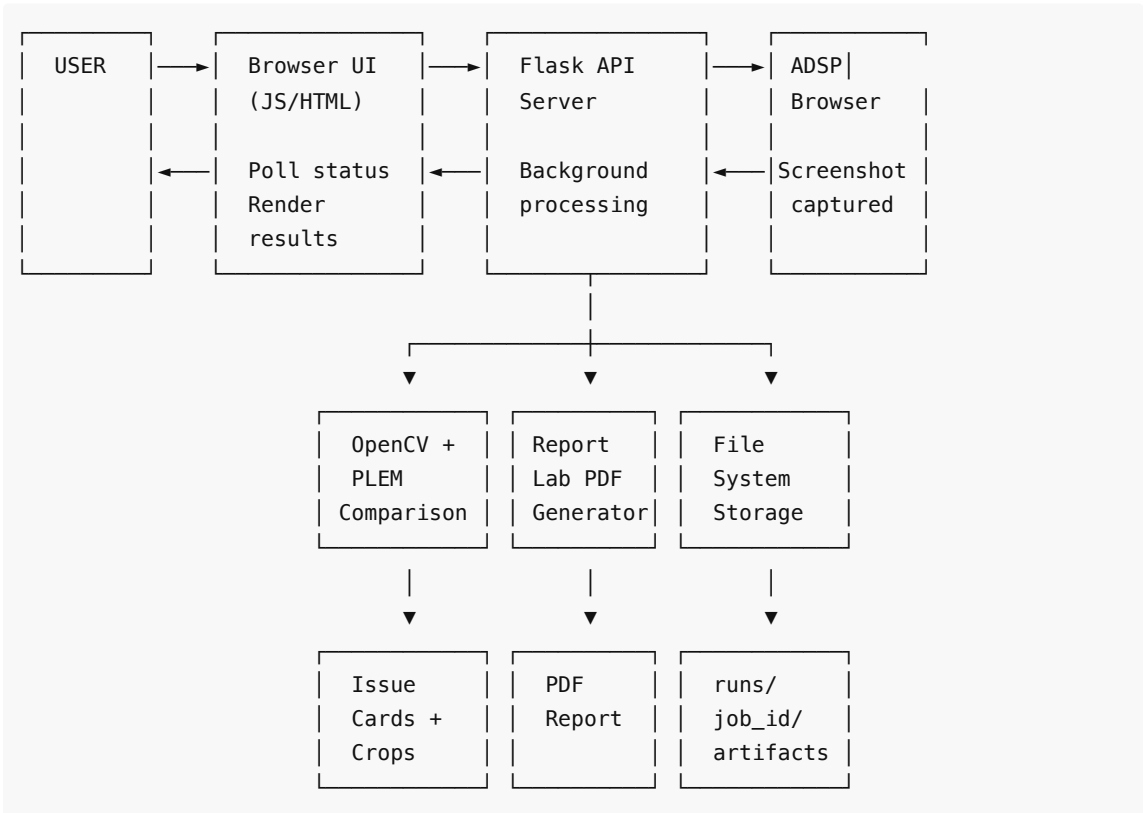


Figure 6 — Data Flow Diagram



🏭 Industrial Applicability

Target Markets and Users

This invention has direct industrial applicability across multiple sectors of the software development industry:

Market Segment	Use Case	Estimated Market Size
Web Development Agencies	Visual QA for client websites before handoff	$5.2B (Global Web Dev Market)
Enterprise QA Teams	Automated regression testing in CI/CD pipelines	$40B+ (Software Testing Market)
E-commerce Platforms	Ensuring visual consistency across product pages	$6.3T (Global E-commerce)
Design-to-Code Teams	Figma-to-live comparison during development	$16.6B (Design Tools Market)
Digital Accessibility Compliance	WCAG audit automation for legal compliance	$800M+ (Accessibility Tools Market)
SEO & Marketing Teams	Performance monitoring and SEO audit automation	$80B (Digital Marketing Industry)

Deployment Models

The self-hosted architecture of this invention enables deployment in environments where cloud-based alternatives are prohibited or impractical:

1. **On-Premises Enterprise Deployment** — Organizations in regulated industries (healthcare, finance, government) require data sovereignty. This invention operates entirely on local infrastructure with no external data transmission for visual testing.
2. **CI/CD Pipeline Integration** — The Docker containerization and GitHub Actions workflow enable seamless integration into existing development pipelines, reducing manual QA effort by an estimated 60-80%.
3. **Standalone QA Workstation** — Individual QA engineers can run the entire platform on a laptop, enabling visual testing without infrastructure dependencies.

Cost Advantage Over Existing Solutions

Solution	Annual Cost (10-Person Team)	Data Sovereignty	Multi-Module
Percy (BrowserStack)	$7,188 – $14,388	❌ Cloud only	❌ Visual only
Applitools Eyes	$10,000 – $50,000	❌ Cloud only	❌ Visual only
Chromatic	$1,788 – $7,188	❌ Cloud only	❌ Visual only
This Invention	**$0 (self-hosted)**	✅ **Full**	✅ **4 modules**

Commercial Viability

The invention can be commercialized through:

1. **Open-core model** — Core engine open-source, premium features (cloud storage, multi-user auth, AI root cause analysis) as paid add-ons
2. **SaaS offering** — Hosted version with per-user pricing
3. **Enterprise licensing** — On-premises deployment with support contracts
4. **Integration partnerships** — Bundling with CI/CD platforms, design tools, and project management systems

---

📋 QA Testing Framework — Complete Feature Documentation

> ***Version:*** *2.0* | ***Last Updated:*** *February 2026*
> ***Author:*** *QA Automation Team*
> ***Stack:*** *Python (Flask) + ADSP + OpenCV + PLEM + 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
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
PLEM Scoring	Perceptual Layout Entropy Metric 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 GASAE spatial 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 Asynchronous DOM-State Projector (ADSP).
2. The screenshot is aligned to the reference image using GASAE-based spatial matching.
3. PLEM 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	PLEM > 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:** ADSP 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
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 (PLEM)

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

- **Default:** 0.85
- **Range:** 0.50 – 0.99
- A comparison **passes** if: `PLEM ≥ 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, PLEM 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 & 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           # ADSP screenshot capture
│   ├── image_compare.py        # PLEM + 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	Asynchronous DOM-State Projector (ADSP)
Image Comparison	OpenCV + scikit-image PLEM
Image Processing	NumPy, Pillow
PDF Generation	ReportLab
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
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
adsp install renderer

# 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`.