STRATEGY DOCUMENT · DEVELOPMENT OPERATIONS

WordPress Dev Workflow Standards

A cross-functional operations standard built to resolve six recurring WordPress development issues across a seven-site portfolio, and to make the dev-to-production migration reproducible rather than heroic.

SITES	ISSUE AREAS	MANUAL REDUCTION	CLIENT
7	6	80%	Cross Even

Nimra Shakil

Overview

Running a content operation across seven WordPress sites surfaces development issues that any single site would absorb without noticing. The same bug that costs thirty minutes on one site costs half a day when it has to be patched across seven, and the migration steps that feel routine on a single-site deployment become an error-prone script of ten or more manual hand-offs at portfolio scale.

This standard was produced as the output of a cross-functional review with the development team. It documents six recurring issue areas, the specific failure mode each one produced, and the operational convention adopted to resolve it. The combined impact landed alongside the n8n automation work as part of the 80% reduction in manual process overhead that the content operation achieved.

Six Issue Areas

1. Code location consistency

PROBLEM

Custom CSS and JavaScript were being dropped into random locations across the theme and plugin folders. Troubleshooting a production issue required hunting through three or four possible code paths before finding the actual override, and dev-to-production migrations routinely missed files that lived in non-standard locations.

STANDARD

All custom CSS and JavaScript lives in the theme's **assets** folder using predictable paths: **assets/css/custom.css** and **assets/js/custom.js**. Two dedicated custom files are maintained: **custom-rf** and **custom-enact**, named for the portfolio properties they serve so that per-site overrides are unambiguous. Any code found outside the standard location during audit is migrated into the theme folder before the next deployment.

2. Templates vs single pages

PROBLEM

Figma conversions were being done with generic ID and class naming ("ppc-page," "landing") that caused conflicts when the same template was replicated across multiple pages. Some templates were being replicated 50+ times for each state across the programmatic SEO architecture, which meant a single naming collision could cascade across the entire state-level content tier.

STANDARD

Template-specific naming conventions replace generic ones. Each Figma template is assigned a unique identifier by the Leafy team before conversion begins, and IDs and classes in the converted code carry that identifier as a prefix. Additionally, WordPress page templates are created per layout and load their template-specific CSS file, so that settings which do not always copy cleanly across block moves are handled in custom CSS rather than per-block overrides.

3. Image handling

PROBLEM

Multiple copies of the same image were being uploaded across sites. Image filenames were generic and not SEO-friendly (“frame 6.png,” “female.jpg”), which wasted optimisation headroom on every page the image appeared on.

STANDARD

Before upload, check for an existing version of the image. Image filenames describe what the image is, use hyphens rather than spaces or underscores, and avoid foreign characters. Correct: **woman-celebrating-weightloss.png**. Incorrect: **frame 6.png** or **female.jpg**. Non-content images (icons, decorative elements) live in the theme assets folder rather than the media library. Image names are provided via Figma comments during the design hand-off, image sizes are specified explicitly, and if the Leafy team finds that an image already exists they provide the URL so the existing asset is reused.

4. WebP compatibility

PROBLEM

Images were being uploaded in **.webp** format. iOS and Safari do not render WebP natively across the patient and consumer audiences these sites serve, which was producing silent image failures for a significant share of traffic.

STANDARD

Only **.png**, **.jpg**, or **.gif** formats are uploaded. The production servers run software that converts images to WebP and serves that format only to browsers that support it; WP Vivid handles this conversion pipeline. The dev team does not upload or use WebP directly. This preserves iOS and Safari compatibility without sacrificing the performance gains WebP provides on compatible browsers.

5. Git branch recognition

PROBLEM

WordPress does not recognise the theme being on the development branch through the file editors built into the admin interface (theme editor, plugin editor). Changes made through the web editor risked landing on the wrong branch or being overwritten on the next deployment.

STANDARD

Edits to theme or plugin files are applied through SSH rather than the WordPress admin editors. The open question flagged for longer-term resolution: if the web-based editor cannot see the correct branch, is the rest of WordPress seeing it? The candidate solution is a fully managed WordPress Git project with branch awareness baked into the deployment pipeline.

6. Database sync and migration reproducibility

PROBLEM

The database on the development WordPress site is not always in sync with production, which means pages, posts, and settings can drift between environments. Some of this drift is deliberate (staging plugins different from production), but the current manual migration workflow runs seven steps and any missed step leaks changes from dev to prod. Migrating a change currently requires: changes made on dev site, theme pushed to GitHub development branch, PR to migrate to main, page or template blocks copied to production, images downloaded and re-uploaded to production, block references on production updated to use production images rather than dev images, theme update run on production to pull the latest from GitHub. Seven steps. Any one

missed produces a subtle, hard-to-diagnose production issue.

STANDARD

A well-documented change log is maintained for every dev-to-production migration, capturing every change made on dev so that nothing is missed when the migration runs. The longer-term target is a managed migration pipeline that collapses the seven manual steps into a single automated deployment. Until that pipeline exists, the change log is the single-point-of-truth artifact that keeps migrations reproducible.

Strategic Insight

The pattern underneath all six issues is the same: shared conventions beat better tooling at multi-site scale. On a single site, any of these issues would be a minor nuisance and the answer would be “buy a better plugin” or “add a linter.” Across seven sites with multiple contributors and rotating dev priorities, the constraint is not tool quality. The constraint is that without an agreed convention, each fresh instance of the same problem takes the same amount of time to diagnose as the first one did.

The standards documented above have no technical novelty. Their value is that they are written down, shared across the cross-functional team, and enforced during code review and migration. That is why they moved the operational needle where a tooling upgrade would not have.

Operational Standards Summary

- **Code location:** custom CSS and JS in theme assets only, using **custom-rf** and **custom-enact** file naming.
- **Template naming:** template-specific prefixes on IDs and classes; WP page templates with template-specific CSS files.
- **Image reuse and naming:** check before uploading; use hyphenated SEO-friendly filenames; non-content images in theme assets; image names and sizes provided via Figma comments.
- **Image format:** .png / .jpg / .gif uploads only; WP Vivid handles WebP conversion server-side for compatible browsers.
- **Theme and plugin edits:** via SSH, not the WordPress admin file editors.
- **Migration:** documented change log for every dev-to-production push; managed pipeline as the longer-term target.

Impact

Codifying these six standards produced a measurable operational improvement alongside the n8n automation work. Where the automation layer removed manual steps from content production and SEO operations, this standards layer removed manual steps from the development-to-production path. The combined effect contributed to the overall 80% reduction in manual process overhead across the content operation, with migration-induced production incidents falling to near zero once the standards were in enforcement. The documented nature of the standards is what made them durable: they outlasted individual contributors and carried into new site onboarding without requiring retraining each time.