# Comprehensive Code-Smells & Anti-Patterns Cheat-Sheet

*(Full-stack: Backend / Laravel / PHP / Blade + Frontend / JS / CSS / Assets)*

Targets: Laravel 12.x, PHP 8.4+, HTML Living Standard, CSS, ECMAScript 2025 (ES2025)

Generated on Nov 6, 2025

Below is an organized list combining common code items into a large, practical cheat-sheet. For each smell there's a short definition, a quick how-to-spot, and one fix/mitigation — with a note when it's especially relevant to specific backend or frontend technologies. Use this as a pre-submission checklist for marketplaces (Envato, CodeCanyon, ThemeForest, Creative Market, TemplateMonster, Mojo Marketplace, Codester, and others).

## 1. General / Backend / Architecture smells

### 1. Duplicate Code

**What:** Same or very similar code in multiple places.

**Spot:** Repeated blocks, copy/paste patterns.

**Fix:** Extract to reusable function/service/trait. Use composer packages where sensible.

**Laravel note:** Extract repeated queries into Repositories / Eloquent scopes.

### 2. Long Method / Large Function

**What:** Functions that do many things or are very long.

**Spot:** Functions > ~50–100 lines doing multiple steps.

**Fix:** Break into smaller methods; apply SRP. Use Services, Jobs, or Form Request classes in Laravel.

### 3. Large Class / God Class / Blob

**What:** Class with too many responsibilities.

**Spot:** Class with dozens of methods, many dependencies.

**Fix:** Split responsibilities (Services, Domain objects, Repositories).

### 4. Long Parameter List

**What:** Methods with many parameters.

**Spot:** >4–5 params, repeated param groups.

**Fix:** Wrap parameters in a Value Object, DTO, or config object.

### 5. Primitive Obsession

**What:** Overuse of primitives instead of domain types.

**Spot:** Passing raw strings/ints for important concepts (email, money).

**Fix:** Create small Value Objects (Email, Money).

### 6. Data Clumps

**What:** Same group of variables passed together everywhere.

**Spot:** (host, port, user, pwd) used as separate parameters.

**Fix:** Wrap into a class/config struct.

### 7. Feature Envy

**What:** A method uses another class's data more than its own.

**Spot:** Lots of calls like `other.getX(), other.getY()`.

**Fix:** Move method to the class it envies.

### 8. Middle Man

**What:** Class that delegates everything to another class.

**Spot:** Methods that just forward calls.

**Fix:** Remove middle man, or enrich it with behavior.

### 9. Message Chains

**What:** Long chained calls `a.getB().getC().do()`.

**Spot:** Deep dot chains.

**Fix:** Encapsulate with a method or hide details behind a service.

### 10. Magic Numbers / Hard-coded Literals

**What:** Raw values sprinkled through code.

**Spot:** Numbers/strings with no explanation.

**Fix:** Extract constants/config/env variables.

### 11. Dead Code / Commented-Out Code

**What:** Unused code blocks or files.

**Spot:** Large commented sections; files not referenced.

**Fix:** Remove; rely on VCS to retrieve history.

### 12. Poor / Inconsistent Naming

**What:** Ambiguous or inconsistent identifiers.

**Spot:** `doStuff()`, `tmp`, `data1`, or inconsistent conventions.

**Fix:** Use meaningful names, follow project naming conventions.

### 13. Conditional Complexity / Deep Nesting

**What:** Many nested if/else and complex boolean logic.

**Spot:** >3 nested levels; huge `switch` blocks.

**Fix:** Use Guard Clauses, Strategy pattern, polymorphism.

## 14. Spaghetti Code

**What:** Tangled control flow, no clear structure.

**Spot:** Hard to follow flow; global state changes in many places.

**Fix:** Refactor into modules/functions with clear boundaries.

## 15. Fat Controller / Fat Model (MVC-specific)

**What:** Controllers or models containing business logic, queries and presentation glue.

**Spot:** Controllers > 200–500 lines; models doing rendering or heavy business logic.

**Fix:** Move logic into Services, Actions, Jobs, or Domain classes; use Form Requests and Resources in Laravel.

## 16. Shotgun Surgery

**What:** One change forces edits in many files.

**Spot:** Frequent PRs touching many modules for small changes.

**Fix:** Improve cohesion; reduce coupling; introduce facades/abstractions.

## 17. Speculative Generality

**What:** Code written "for future features" that never come.

**Spot:** Overly generic hooks, unused extension points.

**Fix:** Simplify, YAGNI (You Aren't Gonna Need It).

## 18. Hidden / Global Dependencies

**What:** Global state, singletons, service locators, static calls.

**Spot:** Use of globals, `window` (frontend), or static facades everywhere.

**Fix:** Use explicit dependency injection and interfaces.

## 19. Inappropriate Intimacy

**What:** Classes tightly access each other's internals.

**Spot:** Frequent use of `friend`-like access or direct field access.

**Fix:** Respect encapsulation; add methods for required behavior.

## 20. Lazy Class / Lazy Element

**What:** Small class that adds little value.

**Spot:** Classes with 1–2 trivial methods.

**Fix:** Merge or remove, unless planned future role is justified.

## 21. Comments Instead of Clean Code

**What:** Comments explain what code does because code is unclear.

**Spot:** Many “what” comments.

**Fix:** Refactor so code is self-documenting; comments should explain why.

### 22. Switch Statements over Polymorphism

**What:** Large `switch`/`case` controlling behavior by type.

**Spot:** Many case branches; adding types requires editing switch.

**Fix:** Use polymorphism or strategy objects.

### 23. Parallel Inheritance Hierarchies

**What:** Two class hierarchies that mirror each other.

**Spot:** Changes duplicated across hierarchies.

**Fix:** Merge or rethink design, composition over inheritance.

### 24. Divergent Change

**What:** Class changed for many reasons frequently.

**Spot:** High churn in one file for many features.

**Fix:** Split responsibilities.

### 25. Refused Bequest

**What:** Subclass inherits behavior it doesn’t use.

**Spot:** Empty overrides or unused inherited fields.

**Fix:** Rework inheritance (prefer composition).

## 2. Backend / Laravel & PHP specific smells & notes

### 26. Eloquent Model Bloat

**What:** Models with business logic, view logic, and heavy query building.

**Spot:** Models with many helper methods, query builders and HTML snippets.

**Fix:** Keep models for relations/attributes; move business logic to Services/Actions; use Query Scopes and Repositories.

### 27. Controller Action Bloat

**What:** Controllers that handle validation, business logic, rendering, and responses.

**Spot:** Controller methods doing DB work, validation, and formatting.

**Fix:** Use Form Requests, Services, Responses (API Resources), Jobs.

### 28. Blade Templates Doing Business Logic

**What:** Heavy PHP logic inside Blade views.

**Spot:** Loops and conditionals that compute complex data in templates.

**Fix:** Compute in controller/service; use View Components, ViewModels/Presenters.

### 29. Overuse of Facades / Static Calls

**What:** Everywhere `Cache::get()` or static helpers, making testing harder.

**Spot:** Static calls in business logic or models.

**Fix:** Inject contracts/clients where possible; use dependency injection.

### 30. Query in Loop (N+1)

**What:** Running DB queries inside a loop resulting in many queries.

**Spot:** Looping through models and calling relations without eager loading.

**Fix:** Eager load relationships (`with()`), use bulk queries.

### 31. Tight Coupling to Framework

**What:** Code that relies heavily on framework globals and facades, making reuse/test hard.

**Spot:** Business code directly calls `request()`, `auth()` etc.

**Fix:** Isolate framework-level code to controllers/adapters.

### 32. No Configurable Environments / Secrets in Code

**What:** Secrets or environment-specific values hardcoded.

**Spot:** API keys, passwords in code.

**Fix:** Use `.env`, config files, and secrets management.

### 33. No Input Validation / Weak Validation

**What:** Inadequate validation before processing input.

**Spot:** Missing FormRequests or ad-hoc validation.

**Fix:** Use Laravel FormRequest with rules and authorize methods.

### 34. Improper Error Handling / Swallowing Exceptions

**What:** Catching exceptions and ignoring them.

**Spot:** `try { } catch (\Throwable $e) {}` with empty catch.

**Fix:** Log and handle errors, return appropriate responses, and bubble up when needed.

### 35. No Tests / Poor Test Coverage

**What:** Little or no automated tests.

**Spot:** No `phpunit`/Pest suites or CI checks.

**Fix:** Add unit, feature tests and CI checks; use test doubles, factories.

### 36. Heavy Migrations / No Migration Strategy

**What:** Large migration files with destructive operations in production paths.

**Spot:** Massive migrations that alter millions of rows.

**Fix:** Break migrations into smaller steps; use rolling changes, background scripts.

## 3. Frontend / UI / Browser-side smells

### 37. Bundle Size Bloat / Too Many Third-Party Libs

**What:** Large JS/CSS because of many heavy libraries.

**Spot:** Big `dist` size; slow initial load.

**Fix:** Tree-shake, replace heavy libs with lightweight alternatives, lazy load.

### 38. No Lazy Loading / Large Initial Payload

**What:** All assets loading at startup.

**Spot:** Slow Time-to-Interactive.

**Fix:** Code split, lazy load routes/components, defer non-critical assets.

### 39. Unoptimized Assets (images, fonts)

**What:** Large images, uncompressed assets, many fonts.

**Spot:** Slow page loads, large network transfers.

**Fix:** Compress, use responsive images, webp, font subsets.

### 40. Inline Styles / Logic in Templates

**What:** Putting JS/CSS or logic directly in HTML/Blade.

**Spot:** Event handlers and large style attributes in template.

**Fix:** Use external JS/CSS and component props; keep templates declarative.

### 41. Mutable Shared State / Side Effects in Components

**What:** Components that mutate shared objects causing unpredictable UI.

**Spot:** Many components reading/writing a global object.

**Fix:** Use immutable patterns or state managers with clear rules (Vuex/Redux/pinia).

### 42. Callback Hell / Nested Subscriptions

**What:** Deeply nested callbacks making flow hard to follow.

**Spot:** Many nested `then/catch` or nested event handlers.

**Fix:** Use async/await, observables, or flatten with Promises utilities.

## 43. Big/Monolithic Components

**What:** UI components that handle too many concerns.

**Spot:** Components with hundreds of lines of logic/markup.

**Fix:** Break into smaller presentational/container components or subcomponents.

## 44. Deep Props Drilling / Excessive Passing

**What:** Passing props through many intermediate components.

**Spot:** Components passing props they don't use.

**Fix:** Use context/provide/inject or a state manager.

## 45. Tight Coupling UI↔Data Layer

**What:** UI tightly tied to data fetching/shape.

**Spot:** Component assumes exact API payloads and transforms heavily within template.

**Fix:** Use adapters or presentational layers; normalize data.

## 46. Untracked List Rendering (key problems)

**What:** Lists rendered without stable keys causing re-render issues.

**Spot:** UI flicker, wrong DOM reuse.

**Fix:** Use stable unique keys (IDs).

## 47. Dead UI Code / Orphan CSS

**What:** Styles or components no longer used.

**Spot:** Unused files, selectors unreferenced.

**Fix:** Remove unused code; run coverage for CSS (PurgeCSS).

## 48. Global Variables / Window Pollution

**What:** Putting app state on `window` or global objects.

**Spot:** `window.app = {...}` usage.

**Fix:** Use module scope, state managers, or DI.

## 49. Inline Event Handlers Everywhere

**What:** Many `onclick`/`onchange` inline handlers.

**Spot:** Inline handlers in HTML attributes.

**Fix:** Use unobtrusive event listeners or framework event binding.

## 50. Accessibility (a11y) Neglect

**What:** Missing alt tags, poor keyboard/ARIA support.

**Spot:** Failing accessibility audits.

**Fix:** Follow WCAG, use semantic HTML and ARIA where needed.

## 4. CSS / Styling smells

### 51. CSS Specificity War / !important Abuse

**What:** Overusing `!important` and overly specific selectors.

**Spot:** Large, complicated selectors and `!important` everywhere.

**Fix:** Adopt BEM/utility classes, simplify selector hierarchy.

### 52. Monolithic Stylesheet (no modularity)

**What:** Huge global stylesheet with everything.

**Spot:** One massive CSS file > thousands lines.

**Fix:** Modularize: CSS modules, scoped styles, component styles.

### 53. Repeated Styles (no variables or tokens)

**What:** Same colors, spacings duplicated.

**Spot:** Many literal color codes and sizes.

**Fix:** Use CSS variables, design tokens, SASS variables.

## 5. Performance & Deployment smells

### 54. Synchronous Blocking Calls / Long Requests

**What:** Blocking operations on the main thread/server code.

**Spot:** Slow responses, UI jank.

**Fix:** Use queues (Laravel Jobs), background processing, optimize queries.

### 55. No Caching or Wrong Caching Strategy

**What:** No caching or caching at wrong layer.

**Spot:** Re-run expensive queries on every request.

**Fix:** Cache views, queries, and computed values appropriately.

### 56. Memory Leaks (server or browser)

**What:** Unreleased resources leading to growing memory.

**Spot:** Increasing memory usage over time.

**Fix:** Close DB connections, remove listeners, avoid retaining large closures.

### 57. Overloaded Third-Party Integrations

**What:** Blocking on slow external API calls.

**Spot:** App waits on third-party responses synchronously.

**Fix:** Use async workers, retries, circuit breakers, timeouts.

## 6. Testing, CI, Documentation smells

### 58. No Continuous Integration / No Static Analysis

**What:** No automated checks (lint, tests, static analyzers).

**Spot:** PRs merged with failing tests or lint errors.

**Fix:** Add CI with linters (PHPCS, ESLint), PHPStan/Psalm, test suites.

### 59. Fragile Tests (brittle)

**What:** Tests that break on small UI/data changes.

**Spot:** Frequent test rewrites for unrelated changes.

**Fix:** Use stable selectors, avoid implementation detail assertions.

### 60. Poor / Missing Documentation

**What:** No README, missing installation or usage docs.

**Spot:** Reviewer confusion, many “how to” questions.

**Fix:** Provide README, usage examples, config notes, upgrade guides.

## 7. Security & Compliance smells

### 61. Unsanitized Input / XSS / SQL Injection Risks

**What:** Not sanitizing or escaping user input.

**Spot:** Direct echoing of input in Blade, raw queries concatenating strings.

**Fix:** Use parameterized queries, Eloquent/Query Builder, `{{ }}` for escaping in Blade, `escape()` where needed.

### 62. Secrets in Repo / Weak Access Controls

**What:** API keys, credentials committed to VCS.

**Spot:** Keys in `.env` checked into repo.

**Fix:** Remove, rotate keys, use secret management.

### 63. Missing CSRF / Broken Auth Checks

**What:** Forms/APIs lacking CSRF tokens or auth enforcement.

**Spot:** State changing endpoints without auth/CSRF protection.

**Fix:** Use Laravel's CSRF middleware, properly guard routes.

## 8. Maintainability & UX smells

### 64. No Versioning / Upgrade Path

**What:** No clear upgrade or breaking change policy.

**Spot:** Package updates break apps; no changelog.

**Fix:** Semantic versioning, CHANGELOG, upgrade docs.

### 65. Monorepo or File Layout Confusion

**What:** Files scattered with inconsistent structure.

**Spot:** No predictable folder structure; test files scattered.

**Fix:** Adopt standard Laravel layout, group by feature where helpful.

### 66. Poor Logging / No Observability

**What:** Lack of logs or too noisy logs.

**Spot:** No meaningful error messages or metrics.

**Fix:** Structured logs, proper levels, integrate monitoring.

## 9. Anti-patterns (common named anti-patterns)

### 67. Shotgun Surgery (covered)

When a change requires many edits across modules.

### 68. Lava Flow

Dead code left in because developer afraid to remove.

**Fix:** Delete with confidence; use VCS.

### 69. Golden Hammer

Using one familiar tool for everything (e.g., solving everything with controllers).

**Fix:** Learn patterns; use the right tool for the job.

### 70. Cargo Cult Programming

Copying patterns without understanding.

**Fix:** Understand before applying; document why patterns exist.

## 10. Marketplace (Envato/ThemeForest/CodeCanyon) specific checklist — quick pass

- Structure: Project follows a clear folder structure and README with installation steps.
- Slim Controllers / Clean Models: No monolithic controllers/models.
- Minimal logic in views: Blade templates are presentation-only; heavy logic moved out.
- Sanitized output & security: XSS/CSRF/SQL injection safeguards present.
- No secrets/keys: `.env` not committed; sample `.env.example` provided.
- Assets optimized: images compressed, JS/CSS minified, bundle small.
- No commented large blocks / dead files: Clean repo.
- Consistent naming & coding standards: Linting and style applied (PSR-12 / ESLint).
- Composer & NPM usage clear: Dependency list and install commands documented.
- Test/smoke tests & CI: At least basic tests or manual test steps documented.
- License/commercial dependencies identified: No forbidden proprietary libs bundled.
- Upgrade notes and compatibility: Which Laravel/PHP versions supported clearly stated.
- Error handling / user feedback: Nice error pages/messages, logs.
- Demo content & seeders: Include seeders/fake data for demo, but not production secrets.
- Accessibility & Responsiveness: Basic responsiveness and accessibility checks.

## 11. How to detect smells quickly (tools & heuristics)

- Static analyzers: PHPStan, Psalm, PHPCS for PHP; ESLint, TypeScript compiler for frontend.
- Linters & formatters: Enforce naming/consistency rules.
- Bundle analyzers: Webpack / Vite bundle analyzers for JS size.
- Profiler: Xdebug/profiler for PHP, Chrome devtools for frontend.
- Code review checklist: Enforce small PRs, mandatory reviews.
- Automated tests: Run tests + mutation testing to assess quality.

## 12. Quick fixes / small refactor recipes

- Duplicate code → Extract method / Trait / Service.
- Long method → Extract function & name the pieces.
- Fat controller → Move business logic to Service or Action class.
- Blade logic → Move to View Component / Presenter / Controller.
- N+1 queries → Eager load or batch queries.

- Large bundle → Split code, lazy load routes/components.
- Hardcoded values → Move to config or constants.
- Global state → Introduce DI or state manager.
- Unclear names → Rename using meaningful domain terms.

## 13. Added items for 2025+ stack (new)

### 71. Missing Content Security Policy (CSP) & Subresource Integrity (SRI)

**What:** Lack of CSP/SRI increases XSS/supply-chain risk.

**Spot:** Security headers absent; scripts/styles loaded from CDNs without integrity attributes.

**Fix:** Define a strict default-src and script-src policy; add SRI hashes to external assets; test in Report-Only before enforcing. Laravel: send headers via middleware.

### 72. CORS Misconfiguration

**What:** Overly permissive origins/headers allow data exfiltration.

**Spot:** Access-Control-Allow-Origin: with credentials; wildcards on private endpoints.

**Fix:** Lock CORS to known origins and methods; do not allow credentials with wildcards. Use Laravel's cors.php config.

### 73. Unbounded Logging and PII in Logs

**What:** Logs grow indefinitely or include sensitive data.

**Spot:** No log rotation; tokens/emails/password hints present in logs.

**Fix:** Scrub PII at log boundaries; rotate and retain with limits; use Monolog processors.

### 74. Missing Rate Limiting / Throttling

**What:** Abuse of auth-sensitive endpoints possible.

**Spot:** No throttling on login/password reset/search endpoints.

**Fix:** Apply Laravel rate limiters (per IP/user); add CAPTCHA or WebAuthn/passkeys for abuse-prone flows.

### 75. No Dependency Pinning / Supply-Chain Risk

**What:** Unpinned composer/npm versions and unscanned artifacts.

**Spot:** Caret ranges everywhere; no advisory scanning.

**Fix:** Pin or lock versions; enable Dependabot/Snyk; verify hashes; use npm/yarn/Composer audits.

### 76. Inefficient Database Indexing

**What:** Slow queries due to missing/incorrect indexes.

**Spot:** Full table scans in slow query logs; composite indexes absent.

**Fix:** Add proper indexes and cover common WHERE/ORDER BY; verify with EXPLAIN and Laravel Scout/Scout Engines as needed.

### 77. No Transaction Boundaries

**What:** Multi-step writes without transactions cause partial state.

**Spot:** Inconsistent data after errors/crashes.

**Fix:** Wrap critical sequences in DB::transaction(); prefer outbox/queued patterns for side effects.

### 78. Inadequate Internationalization (i18n)

**What:** Hard-coded strings and formats hinder localization.

**Spot:** Strings, dates, and currency literals in views/controllers.

**Fix:** Move strings to lang files; use Carbon locales/Intl; format currency via Money VO.

### 79. Improper Timezone & Clock Assumptions

**What:** Mixing server local time with UTC; daylight saving bugs.

**Spot:** Inconsistent timestamps; cron vs. user display mismatch.

**Fix:** Store UTC in DB; convert per-user locale; test DST with Carbon and queues.

### 80. Debug Mode / Verbose Errors in Production

**What:** Sensitive stack traces leaked to users.

**Spot:** APP_DEBUG=true in production; verbose error pages.

**Fix:** Ensure APP_ENV=production, APP_DEBUG=false; provide friendly error pages and Sentry-style reporting.

## References (specs & docs)


- Laravel 12.x Docs — Releases, Blade, CSRF, Eloquent, Starter Kits, Frontend (Vite): https://laravel.com/docs/12.x/ (see specific pages)
- PHP 8.4 — Release announcement & changelog: https://www.php.net/releases/8.4/en.php and https://www.php.net/ChangeLog-8.php
- HTML Living Standard (WHATWG): https://html.spec.whatwg.org/
- ECMAScript 2025 (ECMA-262 16th edition): https://ecma-international.org/wp-content/uploads/ECMA-262_16th_edition_june_2025.pdf
- Vite build tool: https://vite.dev/
- WCAG 2.2 (W3C): https://www.w3.org/TR/WCAG22/
- OWASP Top 10 (2021 edition): https://owasp.org/www-project-top-ten/ / https://owasp.org/Top10/
- PurgeCSS: https://purgecss.com/