

AUTOMATED ASSESSMENT — MARCH 2026

Codebase Health Assessment

CLIENT	Acme Corp
PROJECT	payments-api
DATE	March 2026
ENGAGEMENT	Tier 2 — Improvement Sprint

OVERALL HEALTH — 4 FILES

COMPOSITE HEALTH SCORE

6.2 / 10

Readability		5.8
Simplicity		6.4
Logic Clarity		4.2
Error Handling		3.1
Type Safety		5.4
Performance		7.8

OF FIXES COST \$0

ISSUES FOUND

FUNCTIONS ANALYZED



EXECUTIVE SUMMARY

37 anti-patterns across 4 files,
22 fixable at zero cost.

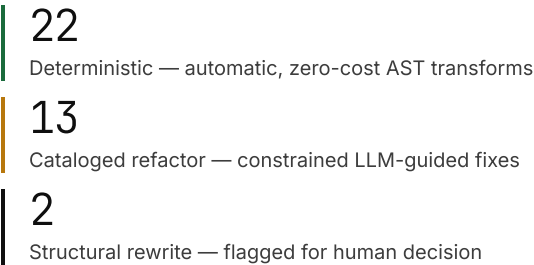
Smelt analyzed 4 source files containing 16 functions across TypeScript, TSX, and Python. The codebase exhibits significant quality debt concentrated in error handling patterns and debug artifact removal. The majority of findings are deterministic fixes that can be applied automatically with no risk.



SEVERITY DISTRIBUTION



FIX TYPE DISTRIBUTION



WORST FILES BY ANTI-PATTERN DENSITY

1	known_bad.ts	19 findings	
2	known_bad.py	8 findings	
3	patch_before.ts	7 findings	
4	component_bad.tsx	3 findings	

59% of identified issues can be fixed automatically at zero cost. Applying all deterministic fixes immediately removes 22 anti-patterns with no human intervention required.

FINDINGS BY FILE

File-level analysis

Detailed breakdown of the two highest-density files in the analysis.

tests/fixtures/known_bad.ts				
Health: 13.9 /100 · 19 anti-patterns · 120 lines				
FUNCTION	ANTI-PATTERN	SEVERITY	FIX TYPE	LINE
fetchUserData	try-catch-rethrow	MEDIUM	DETERM.	13
processPayment	console-log-production	LOW	DETERM.	24
processPayment	console-log-production	LOW	DETERM.	26
processPayment	console-log-production	LOW	DETERM.	28
calculateDiscount	debugger-statement	HIGH	DETERM.	36
isAuthenticated	double-negation	LOW	DETERM.	43
isAuthenticated	any-type-usage	MEDIUM	CATALOG	42
getConfig	redundant-return-await	LOW	DETERM.	48
handleRequest	deep-nesting	HIGH	CATALOG	52
handleRequest	try-catch-rethrow	MEDIUM	DETERM.	62
handleRequest	console-log-production	LOW	DETERM.	64
syncAllData	try-catch-rethrow (×3)	MEDIUM	DETERM.	95+
syncAllData	console-log-production (×2)	LOW	DETERM.	89+
syncAllData	any-type-usage (×3)	MEDIUM	CATALOG	85+

tests/fixtures/python/known_bad.py				
Health: 16.7 /100 · 8 anti-patterns · 20 lines				
FUNCTION	ANTI-PATTERN	SEVERITY	FIX TYPE	LINE
mutable_default	python-eval	CRITICAL	STRUCT.	9
mutable_default	python-exec	CRITICAL	STRUCT.	10
mutable_default	python-mutable-default	HIGH	DETERM.	6
mutable_default	python-breakpoint	HIGH	DETERM.	8
mutable_default	python-bare-except	HIGH	CATALOG	13
mutable_default	python-broad-except	MEDIUM	CATALOG	17
mutable_default	python-type-ignore-bare	MEDIUM	CATALOG	19
mutable_default	python-print-production	LOW	DETERM.	7

TOP FINDINGS

Most critical findings in detail

Each finding includes function metrics, evidence, and recommended remediation.

#1mutable_default · known_bad.py:9CRITICAL

STRUCTURAL REWRITE

eval() — arbitrary code execution risk. The eval() call on line 9 allows arbitrary code execution and represents a severe security vulnerability. This cannot be safely auto-fixed and requires human review to determine the intended functionality and safe alternative.

1COMPLEXITY	1NESTING	14LINES	8PATTERNS
-------------	----------	---------	-----------

RECOMMENDED: Replace eval() with safe parsing (json.loads, ast.literal_eval, or schema-validated input)

#2handleRequest · known_bad.ts:52HIGH

CATALOGED REFACTOR

Nesting depth 6 exceeds maximum of 4. The handleRequest function contains deeply nested control flow that severely impacts readability and maintainability. Combined with 3 additional anti-patterns (try-catch-rethrow, console.log), this function is the highest-complexity hotspot in the codebase.

7COMPLEXITY	6NESTING	23LINES	3PATTERNS
-------------	----------	---------	-----------

RECOMMENDED: Extract-guard-clauses — flatten nesting with early returns

#3UserDashboard · component_bad.tsx:16HIGH

CATALOGED REFACTOR

8 useState hooks exceed the threshold of 5. The UserDashboard component has excessive local state that should be consolidated. At 65 lines with 8 useState hooks, this component is a prime candidate for useReducer extraction or state management refactoring.

5COMPLEXITY	8USESTATE	65LINES	3PATTERNS
-------------	-----------	---------	-----------

RECOMMENDED: Extract-to-useReducer — consolidate related state into reducer

+ 34 additional findings in full report

RECOMMENDED ACTION PLAN

Three-phase improvement roadmap

Fixes are classified into three tiers by execution method. Phase 1 is immediate and automated. Phase 2 requires constrained LLM assistance. Phase 3 requires human architectural judgment.

PHASE 1 — DETERMINISTIC FIXES		22 fixes · Automated · \$0 cost
console-log-production	Remove 8 console.log/debug/info calls	3 files
try-catch-rethrow	Unwrap 6 redundant try-catch blocks	3 files
debugger-statement	Remove 2 debugger statements	2 files
redundant-return-await	Remove 3 unnecessary await in return	2 files
double-negation	Replace 2 !! with Boolean()	2 files
python-mutable-default	Fix 1 mutable default argument (list → None)	1 file
python-breakpoint	Remove 1 breakpoint() call	1 file
python-print-production	Remove 1 print() call	1 file

PHASE 2 — CATALOGED REFACTORS		13 fixes · LLM-guided · Estimated 2-4 hours
any-type-usage (×4)	Add proper TypeScript types to replace explicit any	known_bad.ts
deep-nesting (×1)	Flatten handleRequest with guard clauses	known_bad.ts
excessive-usestate (×1)	Consolidate 8 useState hooks into useReducer	component_bad.tsx
function-too-long (×1)	Extract UserDashboard sub-components	component_bad.tsx
python-bare-except (×1)	Replace bare except with specific exception type	known_bad.py
python-broad-except (×1)	Narrow Exception to specific error types	known_bad.py
python-type-ignore (×1)	Add specific error codes to # type: ignore	known_bad.py

PHASE 3 — STRUCTURAL REVIEW		2 findings · Human decision required
python-eval	Replace eval() with safe parsing — security risk	known_bad.py:9
python-exec	Replace exec() with safe alternative — security risk	known_bad.py:10

Applying all Phase 1 fixes immediately improves the health score by an estimated 1.8 points and removes 22 anti-patterns — with zero risk and zero cost.

APPENDIX

Assessment methodology

Every Smelt audit follows a standardized, reproducible process. Findings are generated by deterministic AST analysis — not heuristics, not regex, not opinion.

ANALYSIS ENGINE

DETECTION

125+ anti-patterns detected via ast-grep — Rust-based AST analysis with zero false positives from regex. Every finding maps to a concrete syntax tree node.

SCORING

12-dimension scoring model evaluates each function individually. File scores are weighted aggregates. Mechanical metrics (25%) combine with subjective review scores (75%) for composite health.

CLASSIFICATION

Three-tier fix classification ensures every finding has a clear, cost-appropriate remediation path. No open-ended suggestions — every fix is either deterministic, constrained, or flagged.

12 QUALITY DIMENSIONS

- 1

Readability
- 2

Simplicity
- 3

Logic Clarity
- 4

Error Handling
- 5

Type Safety
- 6

API Design
- 7

State Mgmt
- 8

Async Safety
- 9

Performance
- 10

Naming
- 11

Decomposition
- 12

Test Strategy

FIX CLASSIFICATION

- DETERMINISTIC

AST transforms applied automatically. No LLM. Zero cost. Verified by re-parse + lint.
- CATALOGED

Named refactoring procedure executed by constrained LLM. Scoped, not open-ended.
- STRUCTURAL

Flagged for human decision. Requires architectural context beyond automated analysis.

CONFIDENTIALITY NOTICE

This report was prepared exclusively for Acme Corp in connection with the payments-api Tier 2 engagement. It contains proprietary analysis and recommendations. Distribution, reproduction, or disclosure to third parties without written authorization from Smelt is prohibited.

WEB

smelt.sh

EMAIL

hello@smelt.sh

PHONE

647-889-1342

© 2025 Smelt™. All rights reserved.