A RESOURCE FOR DESIGN SYSTEM AND ENGINEERING TEAMS

The Moltbook Test.

Can a bot use your design system without guessing?

Find it. Check it. Change it without breaking it.

25
checks, each read from a file

03
levels, each gating the next

10
failures found in the drifting example

Agents use whatever your system leaves unspecified.

An agent building UI does not ask a colleague which button is the real one. It reads what your pipeline exposes, pattern-matches and moves on. Where the system is vague, it guesses, at the speed it writes code.

The test asks the essay's question at three levels. Readability: can an agent find the right component from the docs alone, under one name in Figma and code? Executability: can you validate generated UI mechanically? Survivability: can you change internals without breaking consumers? The levels are cumulative. Perfect versioning with no way to find a component is still Level 0.

Every check reads a file, so the kit runs it. `moltbook-test.mjs` scans a repository and prints the level reached, or scores a self-assessment. The kit's clean example reaches Level 3; its drifting example reaches Level 1 and fails 10 checks above it.

Version 1.0 / Sources checked 24 September 2026
Field Guide 05 of the Design × AI series, with depth in Field Guides 02, 03, 04 and 06. Kit verified on Node 22.22. Specs as of 24 September 2026: llms.txt v2, DTCG 2025.10, Storybook manifests in preview (10.6).

Practical guidance, not a certification. The scanner reads files and never runs them: a pass means the evidence is present and well formed, not that CI runs it. The name borrows from Moltbook; the test has no connection to it. Prepared with AI assistance and edited by hand.

START HERE

Three levels, one question.

Run the scanner first, then read the checks it failed. The levels come from the essay; the checks make each level testable.

Level	The essay's question	You fail when	Depth
1 Readability	Can an agent find the right component from docs alone? Are names consistent across Figma and code?	An agent rebuilds a component you ship, or uses the Figma name	Field Guides 02, 04
2 Executability	If the agent generates UI, can you validate it mechanically?	Invented props, raw hex values or axe failures reach review	Field Guides 02, 03, 04
3 Survivability	Can you change internals without breaking consumers?	A token rename or a prop removal breaks a product silently	Field Guides 03, 06

Scanning a repository

`node scripts/moltbook-test.mjs <repo>`. It reads files, prints the level and every failed check. Start with the kit's two examples to see both outcomes.

No single repo to scan

Copy `self-assessment.template.json`, set each check to pass with a link, na with a reason, fail or open, and run with `--assessment`.

Raising one level

Fix the lowest failing level first. A Level 3 check does not count while a Level 1 check fails, because an agent that cannot find the component never reaches the version it is pinned to.

Keeping the level

Add `--min 2` (or 3) to CI. The scanner exits 1 below that level, so a docs or config change that drops the level fails the build.

Label	Meaning
`SCAN`	The scanner decides from files in the repository.
`EVAL`	Needs an agent run, recorded as a results file the scanner reads.
`N/A ALLOWED`	May be marked not applicable, with a written reason. Only L1.03.
`FG 02` to `FG 06`	Covered in depth by that Field Guide.

> **Where the name comes from.**
>
> Moltbook launched in late January 2026 as a social network for AI agents, alongside the open-source agent now called OpenClaw (briefly Moltbot) [S01]. Its bots post, argue and act without a human in the loop. Your design system now has readers like that.

LEVEL 1

Readability: the agent finds the right component.

An agent chooses components by name and summary, then writes props from whatever list it can find. Level 1 asks whether it can find the right one from your docs alone.

Suggested owners: Docs owner + design-system lead

- [ ] **One index lists every component**

  L1.01 SCAN FG 04

  An agent starts from one file. Publish an llms.txt with an H1, a one-sentence blockquote and a link to every exported component's page. A component missing from the index is a component the agent will rebuild.

  Evidence: llms.txt at the root, `docs/` or `public/` has an H1, a blockquote and a `[Name](` link for every name exported from `src/index`.

  llms.txt v2: the H1 is the only required section, and a file covers the URLs under its path. S02. A complete llms.txt ships with Field Guide 04.

- [ ] **Every component has its own Markdown page**

  L1.02 SCAN FG 04

  One page per component, titled with the exact export name, in plain Markdown. Retrieval serves pages. Guidance that lives only in a docs tab or a Figma annotation does not reach the agent.

  Evidence: `docs/components/<kebab-name>.md` exists for every export, and its H1 is the export name.

  Guidance written in the wrong place drops out of the manifest agents read. S08. Page structure: Field Guide 04.

- [ ] **Names match in code, Figma and contracts**

  L1.03 SCAN N/A ALLOWED

  The essay's second Level 1 question. `PrimaryButton` in Figma and `Button` in code gives an agent reading Figma through MCP one name and the package another. It picks the one it saw last.

  Evidence: The scanner compares three sets: package exports, component-set names from the Figma REST API saved as `figma/component-sets.json`, and contract names. A name in one set and not the others fails. N/A only with a reason, for systems with no Figma library.

  Figma `GET /v1/files/:file_key/component_sets`. S07. Code Connect ties design components to code for Dev Mode and the Figma MCP server. S06.

- [ ] **Each component says when not to use it, and what instead**

  L1.04 SCAN FG 02

  Finding a component is half the job; rejecting the wrong one is the other half. Navigation goes to `Link`, a setting goes to `Switch`, and the contract says so.

  Evidence: Every contract has a non-empty `whenNotToUse[]`, and every `alternative` is an exported component.

  Field Guide 02, item 01.04.

- [ ] **Agent instructions route to the docs and forbid invention**

  L1.05 SCAN FG 04

  The instruction file in consuming repositories points at the index and says, in plain words, not to invent props, values or components.

  Evidence: `AGENTS.md`, `CLAUDE.md` or `.github/copilot-instructions.md` links llms.txt and contains a do-not-invent instruction.

  AGENTS.md: the closest file to the edited code wins. S03. Storybook's MCP guidance tells agents never to use a property they have not looked up. S05. Ready-made files: Field Guide 04.

- [ ] **The docs name what does not exist**

  L1.06 SCAN FG 04

  Agents guess `PrimaryButton`, `variant` and `shadow` because other systems have them. List those guesses with the real alternative, and add each new guess an eval finds.

  Evidence: A "does not export" or "Does not exist" list in llms.txt or a docs page.

  Field Guide 04, rule W16. S05.

- [ ] **Every export has a machine-readable contract**

  L1.07 SCAN FG 02

  Prose is for people. An agent that can read a props list with enums and a props schema does not have to infer them from examples.

  Evidence: A `*.contract.json` with `props` and `propsSchema` for every export.

  Field Guide 02. Storybook's component manifest is another source, but it is in preview and its schema is not yet a stable public API. S04.

- [ ] **A discovery eval proves agents pick the right component**

  L1.08 EVAL

  The essay's first Level 1 question, asked for real. Give an agent only the docs and a task, and record which component it names. Five tasks is the floor; add every task an agent has ever got wrong.

  Evidence: `evals/discovery.results.json` records the model, the date and at least 5 tasks, each answered with the expected component. Start from `discovery-tasks.template.json`. The fixture's results are illustrative, not a recorded run.

  Vercel measured an always-loaded docs index at 100% against 79% for a skill with explicit instructions [S09]; Atlassian found DESIGN.md alone needed about 92% more tokens than its MCP server [S10]. Both are single-task vendor evals: run your own.

If the agent has to guess the name, it will guess one you do not ship.

LEVEL 2

Executability: generated UI is checked by machines.

The essay names four mechanical checks: token linting, prop constraints, accessibility checks and visual regression thresholds. Level 2 adds the conditions that make them hold: nobody can silence them, one command runs them, and they are proven to fail.

Suggested owners: Engineering lead + accessibility lead

- [ ] **Token lint rejects raw values and private tokens**

  L2.01 SCAN FG 03

  Application code, whoever writes it, may use semantic tokens only. Raw hex values and `--ds-core-*` or `--ds-comp-*` variables are lint errors, not review comments.

  Evidence: The ESLint or stylelint config contains the `--ds-(core|comp)-` pattern and a raw-hex rule.

  Field Guide 03, rules R15 and R16.

- [ ] **Nobody can silence the token gate with a comment**

  L2.02 SCAN FG 03

  An agent that hits a lint error will sometimes add a disable comment. Run the gate so that comments cannot switch it off.

  Evidence: Package scripts run ESLint with `--no-inline-config` and stylelint with `--ignore-disables`.

  Field Guide 03, rule R20.

- [ ] **Contracts reject invented props**

  L2.03 SCAN FG 02

  `additionalProperties: false` in every props schema, so `shadow`, `elevated` or any plausible prop an agent invents fails validation.

  Evidence: Every contract's `propsSchema.additionalProperties` is `false`.

  Field Guide 02, item 02.02. S05.

- [ ] **Forbidden combinations fail the compiler**

  L2.04 SCAN FG 02

  A ghost Button is never full width; an icon-only Button needs a label. Model the props so those combinations do not type-check, and keep one `@ts-expect-error` line per rule.

  Evidence: Every contract with `propsSchema.allOf` rules has a `<name>.test-d.ts` containing `@ts-expect-error`. The fixture's Button type test compiles under this site's own `tsc --noEmit`.

  Field Guide 02, items 02.03 and 08.03.

- [ ] **Accessibility violations fail the build**

  L2.05 SCAN

  Axe runs on every story and a violation is a failed test. A warning that nobody reads is not a check. Automated rules catch a subset of issues; people still test with assistive technology.

  Evidence: `.storybook/preview` sets `parameters.a11y.test` to `"error"`. `"todo"` only warns and fails this check.

  Storybook accessibility testing: `error`, `todo` and `off`. S11. What people test: Field Guide 02, section 05.

- [ ] **Visual regression runs with an explicit threshold**

L2.06 SCAN

A threshold is a decision. Write it in config, review changes to it like code and run it in every theme, so a generated change that shifts a layout fails with a diff.

Evidence: `playwright.config` sets `expect.toHaveScreenshot` with `maxDiffPixels` or `maxDiffPixelRatio`. The fixture uses a ratio of 0.001 with light and dark projects; the kit does not run Playwright against it.

Playwright: `maxDiffPixels` and `maxDiffPixelRatio` are unset by default; the per-pixel `threshold` defaults to 0.2. S12.

- [ ] **Every docs example compiles**

L2.07 SCAN FG 04

Agents copy examples more reliably than they follow prose, so a broken example teaches the wrong API with full confidence.

Evidence: A package script extracts docs code blocks and typechecks them (`extract-doc-code`).

Field Guide 04, rule W19.

- [ ] **One command runs the whole gate, for people and agents**

L2.08 SCAN FG 03

Pre-commit, CI and the agent's own verify step run the same `npm run check`, and the agent instructions say so. Storybook's MCP testing tools aim at the same loop: generate, test, fix.

Evidence: A `check` script exists and the agent instruction file tells agents to run `npm run check`.

Field Guide 03, rule R21. S05.

- [ ] **The gate is proven to fail on known-bad code**

L2.09 SCAN

A gate that never fails looks the same as no gate. Keep a fixture of known violations and a script that passes only when the gate rejects it.

Evidence: A `*violations*` fixture and a package script with `gate` in its name.

Field Guide 03's `demo:gate` reports 7 ESLint and 12 stylelint errors on its violations examples.

Fail Level 2 and generated UI still ships. You debug it in production instead.

LEVEL 3

Survivability: internals change, consumers do not break.

The essay's three conditions: semantic tokens as a stable API, versioned contracts and predictable deprecations. Level 3 checks each one, plus the release discipline and decision rights that keep them true.

Suggested owners: Design-system lead + release owner

- [ ] **Core tokens never ship to consumers**

  L3.01 SCAN FG 03

  What a package does not export, nobody can depend on. Keep core and component tokens out of the `exports` map, so you can change them without a breaking release.

  Evidence: No `exports` key in `package.json` matches `core` or `component`.

  Field Guide 03, rule R03.

- [ ] **Every mode has the same semantic keys**

  L3.02 SCAN FG 03

  Light, dark and high contrast differ in values only. A key that exists in one mode breaks the others silently.

  Evidence: The key sets of all `tokens/semantic/*.tokens.json` files are identical.

  Field Guide 03, rule R07.

- [ ] **Every contract carries its own version**

  L3.03 SCAN FG 02

  A contract that changes without a version is a breaking change nobody can see. Give each one a semantic version and let decisions, not edits, move it.

  Evidence: Every contract has a semver `contractVersion`.

  Field Guide 02, item 07.01. Nathan Curtis: decisions in ADRs drive new schema versions. S16.

- [ ] **Deprecations name the replacement and the removal version**

  L3.04 SCAN FG 03

  `$deprecated: true` tells an agent to stop and not what to use instead. Write the replacement and the release that removes it, for tokens and for contract props.

  Evidence: Every token `$deprecated` is a string with "Use <replacement>" and a version; every contract deprecation has `replacement` and `removalPlannedIn`.

  DTCG 2025.10 allows `true`, `false` or an explanation string. S13. Field Guide 03, rule R22; Field Guide 02, item 06.06.

- [ ] **Every rename ships a migration map**

  L3.05 SCAN FG 03

  `{ "color.link": "color.text.link" }` lets a codemod or an agent apply the rename mechanically. Every deprecated token appears in it, and every target exists.

  Evidence: `token-renames.json` maps each deprecated token to a token that exists.

  Field Guide 03, rule R24.

- [ ] **Breaking changes cannot ship without a major version**

  L3.06 SCAN

  Record the public surface of the last release: semantic token keys and contract versions. A removal, or a contract major bump, fails until the package version is a major bump too.

  Evidence: `baseline/api-baseline.json` compared with the working tree and `package.json` version.

  SemVer 2.0.0: incompatible API changes increment MAJOR. S14.

- [ ] **A changelog and a deprecation policy are published**

  L3.07 SCAN

  Predictable deprecations need a stated window. SemVer asks for at least one minor release that carries a deprecation before removal; Acme's policy says two.

  Evidence: `CHANGELOG.md` has an entry for the current version, and `docs/deprecation-policy.md` states a window in releases or time.

  SemVer 2.0.0 FAQ on deprecating functionality. S14.

- [ ] **Breaking changes have one named decider and a deadline**

  L3.08 SCAN FG 06

  Survivability is a governance property too. If "remove this token" has no single decider and no SLA, removals stall until someone ships them anyway.

  Evidence: `decision-rights.json` defines one-way decisions, each with one person as decider (not a group) and `slaBusinessDays`. The full checker ships with Field Guide 06.

  Field Guide 06, sections 01 and 02.

Private internals are the only ones you can change freely.

APPENDIX A

How the scanner scores.

Each check ends as pass, fail (evidence found and wrong) or missing (no evidence found). A level counts only when all its checks and all lower levels pass.

- **Attestations cover gaps, never failures.** A `moltbook.assessment.json` in the repository root may mark a missing check as passed, with a URL or an existing file as evidence. If the scanner found a violation, the attestation is ignored and the report says so.
- **N/A needs a reason.** Only L1.03 accepts it, for a system with no Figma library. The reason must be at least 10 characters.
- **Reading is not running.** The scanner checks that configs and results exist and have the right shape. Your CI proves they run.

terminal: the drifting example

```
$ node scripts/moltbook-test.mjs examples/acme-drifting --min 2
Moltbook Test: Acme Design System (drifting)
Level 1  Readability    8/8
Level 2  Executability  4/9
Level 3  Survivability  3/8
Level reached: 1 of 3
  FAIL     L2.01 Token lint rejects raw values and private tokens: no rule
           against --ds-core-* and --ds-comp-* variables
  FAIL     L2.03 Contracts reject invented props: propsSchema accepts
           unknown props: Link
  FAIL     L2.05 Accessibility violations fail the build: parameters.a11y.test
           is not error at project level (attestation ignored: the scan
           found a violation)
  ATTESTED L2.06 Visual regression runs with an explicit threshold:
           https://ci.acme.example/visual-tests/config
  FAIL     L3.06 Breaking changes cannot ship without a major version:
           semantic tokens removed: color.bg.brand-hover, but package
           stays 4.3.0 (baseline 4.2.0)
  FAIL     L3.08 Breaking changes have one named decider and a deadline:
           D07 is decided by a group (ds-council)
  ... 5 more failures
$ echo $?
1
```

Output from the kit, wrapped for print and shortened to 6 of 11 lines. The clean example, `examples/acme-ready`, reports 8/8, 9/9 and 8/8: Level 3 of 3.

APPENDIX B

What the scanner reads.

Paths are the scanner's defaults and match the kit's example repositories. Move a file and the check reports it missing; change the path in the script or attest it.

Level	Files
1	`llms.txt`, `src/index.(ts|js)`, `docs/components/*.md`, `figma/component-sets.json`, `*.contract.json`, `AGENTS.md` (or `CLAUDE.md`, Copilot instructions), `evals/discovery.results.json`
2	`eslint.config.*`, `stylelint.config.*`, `package.json` scripts, `*.contract.json`, `*.test-d.ts`, `.storybook/preview.*`, `playwright.config.*`, `*violations*` fixtures
3	`package.json` exports and version, `tokens/**/*.tokens.json`, `*.contract.json`, `token-renames.json`, `baseline/api-baseline.json`, `CHANGELOG.md`, `docs/deprecation-policy.md`, `decision-rights.json`

examples/self-assessment.example.json (excerpt)

```
{
  "system": "Northwind UI (example self-assessment)",
  "assessedOn": "2026-09-24",
  "checks": {
    "L1.01": { "status": "pass", "evidence": "https://design.northwind.example/llms.txt" },
    "L1.03": { "status": "na",
               "reason": "No Figma library: the system is designed and documented in code
only." },
    "L3.02": { "status": "fail", "note": "High-contrast mode lacks 4 semantic keys." },
    "L3.04": { "status": "open" }
  }
}
```

Scored with `--assessment`, this example reaches Level 2: Level 3 has one fail and five open checks. A pass without evidence is scored as open.

.github/workflows/design-system.yml (steps)

```
- run: npm ci
- run: npm run check
- name: Moltbook Test (fail below Level 2)
  run: node scripts/moltbook-test.mjs . --min 2
```

Raise `--min` when you reach the next level, so the level can only go up.

KEEP WITH THE SYSTEM

Leave a readiness record.

One record per assessment. Run it each quarter and after any change to docs, lint or release tooling. The trend matters more than the score.

System / package version	**Assessed on / by**
Level reached (scan) / level reached (self-assessment)	**Failed checks and owners**
Attested checks and their evidence	**N/A checks and reasons**
Discovery eval: model, date, tasks, misses	**CI minimum level (--min)**
Next assessment date	**Decision log link (Field Guide 06)**

> **Do not attest what you can fix.**
> An attestation is for evidence that lives outside the repository. If the file could exist in the repository, add the file.

SOURCES / MAINTENANCE

Keep the guide current.

Checked 24 September 2026. Vendor eval figures are single-task results.

S01 / Wikipedia, OpenClaw
Clawdbot to Moltbot to OpenClaw, January 2026; Moltbook, a network for AI agents.
https://en.wikipedia.org/wiki/OpenClaw

S02 / llms.txt proposal (v2)
H1 is the only required section; a file covers the URLs under its path.
https://llmstxt.org/index.md

S03 / AGENTS.md
A README for agents; the closest file wins.
https://agents.md/

S04 / Storybook, component manifests
Docgen props, JSDoc, stories; preview in 10.6, schema not yet stable.
https://storybook.js.org/docs/ai/manifests

S05 / Storybook, MCP server
Docs, development and testing tools; check every property first.
https://storybook.js.org/docs/ai/mcp/overview

S06 / Figma, Code Connect
Connects code components to Figma for Dev Mode and the Figma MCP server.
https://developers.figma.com/docs/code-connect/

S07 / Figma REST API, component endpoints
GET /v1/files/:file_key/component_sets.
https://developers.figma.com/docs/rest-api/component-endpoints/

S08 / Rachel Cantor, agents read a different system
The manifest is a lossy copy of the docs (July 2026).
https://rachel.fyi/posts/your-agent-is-reading-a-different-design-system

S09 / Vercel, AGENTS.md outperforms skills
Docs index 100%, skill with instructions 79% (January 2026).
https://vercel.com/blog/agents-md-outperforms-skills-in-our-agent-evals

S10 / Atlassian, testing DESIGN.md
DESIGN.md alone used about 92% more tokens than MCP (June 2026).
https://www.atlassian.com/blog/how-we-build/atlassians-design-md-is-here-what-we-learned-testing-portable-design-context-in-practice

S11 / Storybook, accessibility testing
a11y.test: error fails, todo warns, off disables.
https://storybook.js.org/docs/writing-tests/accessibility-testing

S12 / Playwright, toHaveScreenshot
maxDiffPixels and maxDiffPixelRatio unset by default; threshold 0.2.
https://playwright.dev/docs/api/class-pageassertions

S13 / DTCG, Format module 2025.10
$deprecated as true, false or an explanation string.
https://www.designtokens.org/TR/2025.10/format/

S14 / Semantic Versioning 2.0.0
MAJOR for incompatible changes; deprecate in a minor first.
https://semver.org/

S16 / Nathan Curtis, Component Contracts and Schemas
ADRs drive new contract schema versions (2026).
https://nathanacurtis.substack.com/p/component-contracts-and-schemas

Maintenance: recheck llms.txt, AGENTS.md and the Storybook AI pages each quarter. When evidence moves to a new standard file, update the scanner and the check together.