



How I Decide What to Test in Laravel

Name the promise.
Choose the boundary.
Prove the failure that matters.

PROMISE

What result must hold?



BOUNDARY

What can prove that result?



EVIDENCE

What would make it fail?

Start with what the test should prove.

The suites describe how the test enters the code and what it exercises.

01	Unit	Pure logic with ordinary inputs. My Unit suite does not boot Laravel.
----	-------------	--

02	Integration	Call application code directly with the collaborators its contract needs.
----	--------------------	---

03	Feature	Enter through a request, command, or component's server-side behavior.
----	----------------	--

Laravel defaults to Unit and Feature. A separate Integration suite is my convention.

Assert the result the application relies on.

A green test helps only if it would fail when the promised behavior breaks.

01 Name the behavior

A test name should explain the broken promise without requiring an implementation tour.

02 Make the difference visible

Use a clear setup, one operation, and assertions about the observable result.

03 Protect the right contract

Check stored state, visible and excluded content, or errors and promised preserved input.

Add cases that catch a different defect. Do not repeat every calculation at every layer.

Prove the contact handoff is atomic.

This site saves the inquiry and queues email work in one database transaction.

01 Force the failure

The integration test deliberately fails the insert into the test queue table.

02 Check the rollback

Neither the inquiry nor its queued job should remain after that failure.

03 Try the action again

A new call succeeds: one inquiry and one queued job exist.

A queue fake cannot prove the database rollback. It replaces the interaction being tested.

Check both the listing and the direct URL.

Private content has more than one possible route out of the application.

01 Arrange meaningful states

Create published, draft, and future-dated posts.

02 Request the public listing

The published title appears.
The draft and future titles stay out.

03 Request individual posts

Draft and future posts return 404
when their URLs are requested directly.

For a denied write, test both the expected response and the unchanged stored record.

A fake changes what the test can prove.

Keep external effects isolated while retaining the collaborators the question needs.

- | | | |
|----|--------------------------|--|
| 01 | Queue fake | Proves that code requested a job.
It does not execute the job's business operation. |
| 02 | Mail or HTTP fake | Arranges controlled external behavior.
It does not prove real provider delivery. |
| 03 | Database boundary | Use real test inserts when persistence matters.
SQLite does not prove another database's locks. |

Tests should not email customers or send unexpected requests to live services.

Choose checks that answer different questions.

An HTTP response cannot prove every interaction or rendering result.

- | | | |
|----|--------------------------------|---|
| 01 | Browser | Use it for JavaScript, loaded images, mobile overflow, and working controls. |
| 02 | Architecture | Enforce agreed source conventions, such as dependencies between layers. |
| 03 | Keep each check focused | Server-side permissions and persistence still need their appropriate test boundaries. |

A mobile menu in the markup is not proof that it opens, closes, or follows a link.

Start with the failure you care about.

For a bug: reproduce the failure, make it pass, then refactor while keeping it correct.

- ☐ **Observable promise** What result am I protecting?
- ☐ **Boundary** Which entry point can actually prove it?
- ☐ **Failure case** What failure or forbidden case matters?
- ☐ **Collaborators** What must be real, and what should I isolate?
- ☐ **Test strength** Could the behavior break while this still passes?

[Read the full article: How I Decide What to Test in Laravel](#)