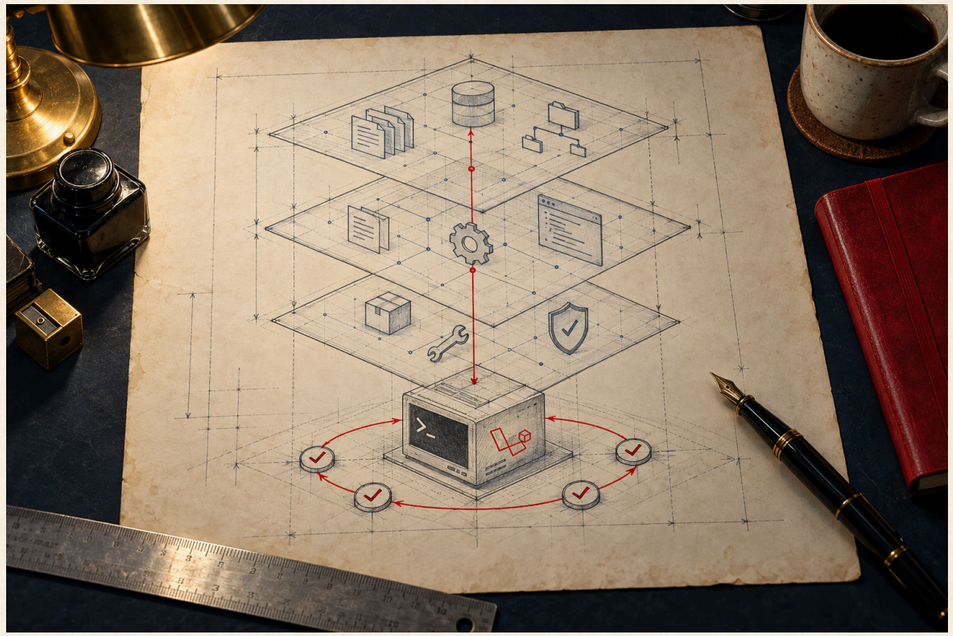
A DOWNLOADABLE COMPANION

THE LARAVEL
ARCHITECT

How I Set Up Claude Code So It Writes My Laravel

For Laravel developers already using Claude Code who want it to follow their project's conventions.

Put recurring corrections where the next session can read them.



CONTEXT → PLAN → VERIFIED CHANGE

Give every rule a place to live.

Personal defaults set the baseline. Project instructions explain this app.
Focused rules hold the decisions that matter for the file being changed.

01 ↓

Personal defaults

`~/.claude/CLAUDE.md`

Your recurring preferences across projects.
Example: plan first and keep changes small.

02 ↓

Project instructions

`AGENTS.md`

The app's workflow and conventions.
TLA tells the agent to read the rules index.

03

Rules for the work

`.ai/rules/index.md → models.md`

Match the file to the relevant rule.
Example: keep HTTP concerns out of Eloquent models.

> Check what loads. AGENTS.md support depends on Claude Code's version and settings.
> Boost maintains the .ai/rules index; TLA's instructions direct the agent to read it.

Reference: Claude Code instruction files and imports

Write the correction you keep making.

Name the responsibility, explain the boundary, and give the agent a way to check its work.

.ai/rules/models.md · ARTICLE EXCERPT

Keep HTTP concerns out of models

Keep Eloquent models focused on persistence and domain behavior.

Do not generate routes, signed URLs, redirects, or other HTTP/presentation output in models.

01 Read the existing flow

Find how this project prepares links and page data before changing the model.

02 Follow the project's pattern

In TLA, presenters prepare model URLs. Reuse that responsibility when it fits.

03 Check the boundary

TLA has architecture tests for this rule. Run relevant checks and inspect the diff.

> Start with one correction. Keep the rule short. Update or remove it when the decision changes.

Ask for evidence before the implementation.

Boost supplies project context and documentation tools. Read the result before trusting the answer.

`search-docs`

Check the API for the installed package version.

`database-schema`

Inspect columns and defaults before using them.

`get-absolute-url`

Resolve the app's URL instead of guessing it.

`record-rule`

Save a settled convention when you explicitly ask.

ILLUSTRATIVE MODEL METHOD · LARAVEL 13

```php
use Illuminate\Database\Eloquent\Attributes\Scope;
use Illuminate\Database\Eloquent\Builder;

#[Scope]
protected function active(Builder $query): void
{
    $query->where('active', 1);
}
```

> Version-aware docs narrow the search. They do not replace reading the code or checking behavior.

Laravel Boost

Laravel 13 Eloquent scopes

Review the plan before the diff.

A short plan gives you a chance to correct the direction while changing it is still cheap.

01	02	03	04	05
Read	**Plan**	**Approve**	**Implement**	**Verify**
Trace the flow. Find the rules.	List the change and its checks.	Confirm intent and boundaries.	Keep the change small and focused.	Run checks. Review the diff.

FOR A BUG

Reproduce the failure → **Make the smallest fix** → **Prove the behavior**

> TLA plan location: meta repo → cycles/YYYY-MM-DD-short-name/summary.md

Use one command for the project’s gates.

Keep human review focused on intent, behavior, and whether the change belongs in the app.

```
composer check
```

Defined by this repository.
Not a built-in Composer command.

Format + types

Pint checks formatting.
PHPStan checks types.
Rector checks modernization.

Behavior

Pest checks behavior.
Architecture tests check boundaries.
Browser tests cover browser flows.

Build + dependencies

Audit dependencies.
Build production assets.
Check asset and deployment scripts.

> While working: run the relevant checks. At the full gate: run the repository’s complete check command.

Be clear about what needs your decision.

Instructions help the agent work within the project. They do not make every decision safe to delegate.

Claude can prepare

01 Trace code and propose a small plan.

02 Implement within the approved scope.

03 Run checks and explain the evidence.

You decide

01 Product direction and architectural tradeoffs.

02 Publishing, production and destructive actions.

03 Whether the result matches your intent.

> Keep rules current. When a decision changes, update its rule. Remove rules that no longer apply.

Make the next task easier to review.

Start with the conventions you already enforce. Add a rule when it saves a repeated explanation.

01 **Connect the context**
Set up Boost for your agent; confirm the tools load.

02 **Write five conventions**
Use concrete examples from your own repository.

03 **Require a short plan**
Read the code, agree on scope, then implement.

04 **Name the checks**
Tell the agent which project scripts prove the change.

05 **Maintain the rules**
Record settled decisions explicitly; prune stale ones.

TRY IT ON YOUR NEXT TASK

Read the relevant code
and project rules.

Propose the smallest change
and the checks it needs.

Wait for my approval
before changing code.

Laravel Boost Claude Code memory Laravel 13 Eloquent scopes