



How I Handle Authorization in Laravel

Name the permission.
Enforce it on the server.
Prove the denied path.

IDENTITY

Who is making the request?



PERMISSION

May they perform this action?



ENFORCEMENT

Stop before protected work

A valid login is only the starting point.

Ask whether this person may perform this action on this resource.

01 Authentication

Establish who is making the request.
That alone does not grant every permission.

02 Authorization

Name a verb and a resource:
view this post, update it, or publish it.

03 Guest access

Some public operations deliberately allow
guests. Make that choice explicit in the rule.

Two authors can both sign in without being allowed to edit each other's drafts.

Keep related permission rules together.

Each entry point should ask the same rule about the same ability.

- | | | |
|----|---------------------|---|
| 01 | Policy | Group permissions for a model or resource, such as viewing or updating a post. |
| 02 | Gate | Define a general ability, such as access to an administrative dashboard. |
| 03 | Use the rule | Laravel's Gate API can evaluate policy abilities. For create, pass the model class, not a dummy record. |

Gates and policies organize rules. The caller still has to enforce the answer.

Stop unauthorized work on the server.

A hidden button helps the interface. It cannot protect the endpoint.

01 Check before the operation

Use the entry point's existing authorization: middleware, Form Request, or Gate::authorize().

02 Handle the denial

A false result helps only if the caller stops. Authorization exceptions normally produce 403.

03 Keep controls consistent

Use the same ability to show permitted controls. Direct requests still need server enforcement.

A deliberate 404 denial can hide a resource's existence. Test the response you choose.

Permission does not settle every state rule.

“Who may do this?” and “is it valid now?” are different questions.

01 Permission

A staff member may have the ability to cancel orders.

02 Business rule

A shipped order may no longer be cancellable. The cancellation operation must enforce that rule.

03 Check the current state

A prior decision or visible button cannot guarantee the record has not changed.

A policy may consider state. The operation still protects the business rule it depends on.

Know what an override skips.

This site's broad administrator rule is a deliberate choice for a small publishing app.

```
Gate::before(fn (User $user): bool => $user->is_admin);
```

true

Allow the ability.
Skip the ordinary rule.

false

Deny the ability.
Skip the ordinary rule.

null

Continue to the ordinary
authorization check.

TLA returns true or false. A super-admin bypass with ordinary policies needs a null fallback.

Test the request someone should not send.

A missing button is not proof that a protected operation rejects a caller.

01 Allowed and denied users

Try the same operation with users who should and should not have permission.

02 Check the protected result

For writes, confirm the record stayed unchanged. For reads, confirm private content did not appear.

03 Include boundary cases

In workspace apps, try another workspace's record using a guessed identifier.

TLA tests its broad admin rule. A richer permission model needs richer cases.

Follow the permission all the way through.

Pick an actual operation and make each responsibility visible.

- ☐ **Identity** Who is calling, and are guests allowed?
- ☐ **Ability + resource** Which permission does the server check?
- ☐ **Enforcement** Where does denial stop protected work?
- ☐ **Business rules** What must still hold when the operation runs?
- ☐ **Evidence** Does a denied request leave protected data safe?
- ☐ **Interface** Do controls reflect the server's permission?

[Read the full article: How I Handle Authorization in Laravel](#)