

Laravel¹³

Style Guide

Conventions for writing, structuring, testing and shipping
Laravel 13 apps on PHP 8.5, with attribute-first models and a
Livewire or Inertia frontend.

```
#[Table('invoices')]  
#[Fillable(['customer_id', 'number', 'status'])]  
final class Invoice extends Model {  
    use HasUuids; // Declare, don't configure.  
}
```

Inside this guide

01	Purpose and how to read this guide	2
02	Baseline and toolchain	3
03	Project layout	5
04	Naming	8
05	Formatting and static analysis	10
06	Modern PHP usage	11
07	Eloquent models: attributes for everything	13
08	Migrations, factories and seeders	19
09	Queries and performance	22
10	Routing and controllers	24
11	Validation and form requests	26
12	Authorization	28
13	Actions, services and DTOs	29
14	Frontend: Livewire or Inertia	32
15	APIs and resources	37
16	Jobs, events, queues and commands	39
17	Configuration and environment	42
18	Errors, logging and observability	43
19	Security	44
20	Testing with Pest	45
21	Git, commits and code review	48
A	Appendix	49

Rule levels

MUST	MUST NOT	SHOULD	SHOULD NOT	MAY	AVOID
------	----------	--------	------------	-----	-------

01

Purpose and how to read this guide

This guide defines how Laravel 13+ applications in `Project` are written, structured, tested and reviewed. It is generic: swap `Project` for your application name and the `Invoice` examples for your own domain. It covers both supported frontend stacks — **Livewire 4 + Flux UI** and **Inertia v3 + React** — and calls out where the two differ.

The guiding idea is the same one Laravel 13 leans into: **declare, don't configure**. Wherever the framework offers a PHP attribute for something (model tables, fillable fields, job retries, command signatures, controller middleware), we use the attribute instead of a protected property or a method override. Configuration lives on the class declaration where a reader sees it first.

Rule levels

LEVEL	MEANING
<code>MUST</code> / <code>MUST NOT</code>	Required. CI or review blocks the change.
<code>SHOULD</code> / <code>SHOULD NOT</code>	The default. Deviate only with a short comment explaining why.
<code>MAY</code>	Allowed when it helps; no justification needed.
<code>AVOID</code>	Legal but discouraged; expect a review question.

Principles

- **Readable over clever.** Code is read far more than it is written.
- **Thin edges, rich core.** Controllers, Livewire components and commands stay thin; business rules live in actions and models.
- **Let the tools enforce it.** Anything Pint, Larastan or Rector can check is not debated in review.
- **Small pieces.** Small classes, small components, small pull requests.
- **Framework first.** Reach for a Laravel feature before a package, and a package before custom infrastructure.

02

Baseline and toolchain

AREA	STANDARD
Framework	Laravel 13.x (released 17 March 2026; bug fixes to Q3 2027, security fixes to 17 March 2028)
PHP	8.5 minimum (the newest version Laravel 13 supports)
Database	PostgreSQL 17+ (MySQL 8.4+ allowed); UUIDv7 primary keys
Frontend A	Livewire 4 + Flux UI Pro + Tailwind CSS 4
Frontend B	Inertia v3 + React 19 + TypeScript (strict) + Tailwind CSS 4
Build	Vite
Tests	Pest 4 (with the browser plugin for end-to-end tests)
Formatting	Laravel Pint (<code>laravel</code> preset + project rules)
Static analysis	Larastan at level 8 minimum
Refactoring	Rector with the Laravel set
Queues	Redis via Horizon in production; <code>database</code> driver locally
Local dev	Laravel Herd or Sail; <code>composer dev</code> runs server, queue, logs and Vite together

Rules

- MUST** LARAVEL-02.1 pin `"php": "^8.5"` and `"laravel/framework": "^13.0"` in `composer.json` and commit `composer.lock`.
- MUST** LARAVEL-02.2 use `declare(strict_types=1);` at the top of every PHP file in `app/`, `database/`, `routes/` and `tests/`.
- MUST** LARAVEL-02.3 keep the Laravel 11+ slim skeleton: middleware, exceptions and routing are configured in `bootstrap/app.php`, not in a `Kernel` class.
- SHOULD** LARAVEL-02.4 follow each Laravel upgrade guide within one minor release; Laravel 13 has minimal breaking changes, so stay current.

- **MUST NOT** LARAVEL-02.5 add a package that duplicates a first-party feature (for example, a separate UUID package — `HasUuids` already generates UUIDv7).
- **MAY** LARAVEL-02.6 use first-party packages freely: Horizon, Pulse, Reverb, Scout, Sanctum, Socialite, Pennant, Cashier, the AI SDK and Boost.

03

Project layout

Start from the default Laravel 13 skeleton and add folders only when they earn their keep. Group by **type first, then by domain** inside each type folder once a folder passes about 15

files.

```
app/  
  Actions/           # One class per use case: CreateInvoice, SendInvoice  
    Invoices/  
  Data/             # Readonly DTOs passed between layers  
  Enums/            # Backed enums: InvoiceStatus  
  Events/ Listeners/  
  Exceptions/        # Domain exceptions: InvoiceAlreadyPaid  
  Http/  
    Controllers/  
    Requests/        # Form requests  
    Resources/       # API resources  
  Jobs/  
  Livewire/          # Livewire stack: class-based components only (rare)  
  Models/  
    Concerns/        # Model traits: Invoice/HasInvoiceRelationships.php  
    Builders/         # Custom Eloquent builders  
    Collections/     # Custom collections  
  Observers/  
  Policies/  
  Providers/  
  Queries/           # Reusable read-side query objects  
bootstrap/app.php    # Routing, middleware, exceptions  
config/  
database/  
  factories/ migrations/ seeders/  
resources/  
  css/app.css  
  views/             # Blade layouts, Blade components, Livewire SFCs  
    components/      # Blade + Livewire SFCs (⚡ invoice-table.blade.php)  
    layouts/         # layouts:: namespace  
    pages/           # pages:: full-page Livewire components  
  js/                # Inertia stack only  
    pages/           # One file per Inertia page: invoices/index.tsx  
    components/      # Small, reusable React components  
    layouts/  
    hooks/  
    types/           # Generated + hand-written TS types  
routes/  
  web.php api.php console.php  
tests/  
  Feature/ Unit/ Browser/
```

Rules

- **MUST** LARAVEL-03.1 keep one class per file, with the file named after the class.
- **MUST** LARAVEL-03.2 mirror `app/` inside `tests/Feature` and `tests/Unit` so each class's tests are easy to find (`app/Actions/Invoices/CreateInvoice.php` → `tests/Unit/Actions/Invoices/CreateInvoiceTest.php`).

- **MUST NOT** LARAVEL-03.3 create `Helpers`, `Utils`, `Misc` or `Common` dumping-ground folders.
- **SHOULD** LARAVEL-03.4 split a model's relationships, scopes and accessors into traits under `app/Models/Concerns/<Model>/` once the model grows past about 150 lines (see §7).
- **SHOULD** LARAVEL-03.5 keep `routes/web.php` short by grouping with `Route::controller()` and `Route::prefix()`; split into `routes/web/*.php` files only past about 150 lines.
- **AVOID** LARAVEL-03.6 deep module systems (`Modules/Billing/...`) unless the team is large enough to own modules independently.

04

Naming

THING	CONVENTION	EXAMPLE
Model	Singular PascalCase	<code>Invoice</code> , <code>InvoiceLine</code>
Table	Plural snake_case	<code>invoices</code> , <code>invoice_lines</code>
Pivot table	Singular models, alphabetical	<code>customer_tag</code>
Column	snake_case	<code>due_date</code> , <code>paid_at</code>
Foreign key	<code><model>_id</code>	<code>customer_id</code>
Boolean column	<code>is_</code> / <code>has_</code> prefix	<code>is_archived</code>
Timestamp column	<code>_at</code> suffix	<code>sent_at</code>
Controller	Singular + <code>Controller</code>	<code>InvoiceController</code>
Invokable controller	Verb phrase + <code>Controller</code>	<code>SendInvoiceController</code>
Action	Verb phrase	<code>CreateInvoice</code> , <code>MarkInvoicePaid</code>
Form request	Verb + model + <code>Request</code>	<code>StoreInvoiceRequest</code>
API resource	Model + <code>Resource</code>	<code>InvoiceResource</code>
Job	Imperative verb phrase	<code>GenerateInvoicePdf</code>
Event	Past tense	<code>InvoicePaid</code>
Listener	What it does	<code>SendPaymentReceipt</code>
Policy	Model + <code>Policy</code>	<code>InvoicePolicy</code>
Enum	Singular PascalCase, PascalCase cases	<code>InvoiceStatus::Overdue</code>
Model trait	<code>Has</code> + model + concern	<code>HasInvoiceRelationships</code>
Livewire component	kebab-case view / PascalCase class	<code>invoices.create-invoice</code>
Inertia page	kebab-case file under <code>pages/</code>	<code>pages/invoices/show.tsx</code>

THING	CONVENTION	EXAMPLE
React component	PascalCase file and export	<code>InvoiceStatusBadge.tsx</code>
Route name	Plural resource + dot action	<code>invoices.show</code>
URL	Plural kebab-case	<code>/invoices/{invoice}/line-items</code>
Config key	snake_case	<code>billing.default_currency</code>
Variable / method	camelCase	<code>\$dueDate</code> , <code>markPaid()</code>
Constant	UPPER_SNAKE	<code>MAX_LINES</code>

Rules

- **MUST** LARAVEL-04.1 name methods for what they do from the caller's view: `$invoice→markPaid()`, not `$invoice→updateStatusToPaid()`.
- **MUST** LARAVEL-04.2 name booleans so they read as a question: `$isOverdue` , `canBeSent()` .
- **MUST NOT** LARAVEL-04.3 use abbreviations other than well-known ones (`id` , `url` , `pdf` , `api`).
- **SHOULD** LARAVEL-04.4 name collections in the plural and single items in the singular: `$invoices` , `$invoice` .
- **AVOID** LARAVEL-04.5 suffixing classes with their pattern when the folder already says it (`Actions\CreateInvoice` , not `CreateInvoiceAction`).

05

Formatting and static analysis

Formatting is never a review topic: Pint decides, CI enforces.

Formatting

- **MUST** LARAVEL-05.1 run Laravel Pint with the committed `pint.json` (Appendix A). CI runs `pint --test` and fails on any diff.
- **MUST** LARAVEL-05.2 use **4-space** indentation in PHP (PSR-12, which Pint enforces) and **2-space** indentation in Blade, TypeScript, CSS, JSON and YAML (`.editorconfig` in Appendix A).
- **SHOULD** LARAVEL-05.3 keep lines under **120** characters; break long method chains one call per line.
- **MUST** LARAVEL-05.4 use short array syntax, trailing commas in multi-line arrays, arguments and parameters, and single quotes unless interpolating.
- **MUST** LARAVEL-05.5 order class members: constants, properties, constructor, public methods, protected methods, private methods. Attributes sit directly above the element they configure, one attribute per line.
- **MUST** LARAVEL-05.6 import every class with `use`; no fully qualified names inline.

Static analysis

- **MUST** LARAVEL-05.7 run Larastan at **level 8** or higher (Appendix B). New code adds no baseline entries; the baseline only shrinks.
- **MUST** LARAVEL-05.8 give every parameter, property and return a native type. Use PHPDoc only for what PHP cannot express: generics (`Collection<int, Invoice>`), array shapes (`array{id: string, total: int}`) and `@property-read` for model columns.
- **MUST NOT** LARAVEL-05.9 use `@phpstan-ignore` without a trailing reason: `// @phpstan-ignore argument.type (vendor stub is wrong)`.
- **SHOULD** LARAVEL-05.10 run Rector with the Laravel and PHP 8.5 sets in dry-run mode in CI; apply its changes in separate refactor commits.

One command

- **MUST** LARAVEL-05.11 expose `composer lint` (Pint, Larastan, Rector dry run) and `composer test` (Pest in parallel). Both run in CI on every pull request (Appendix D).

06

Modern PHP usage

| Classes

- **MUST** LARAVEL-06.1 declare classes `final` by default. Remove `final` only when a class is designed for extension (models extended by a package, abstract bases).
- **MUST** LARAVEL-06.2 use constructor property promotion and `readonly` for injected dependencies and DTOs; a DTO is a `final readonly class`.
- **SHOULD** LARAVEL-06.3 use PHP 8.4 asymmetric visibility (`public private(set)`) instead of a getter for a value that is read publicly but written only internally.
- **MAY** LARAVEL-06.4 use PHP 8.4 property hooks on plain objects; **MUST NOT** use them on Eloquent models (they bypass Eloquent's attribute system — use casts and accessors there).
- **SHOULD** LARAVEL-06.5 chain on `new` without parentheses (PHP 8.4): `new InvoiceNumber($raw)->formatted()`.

| Types and values

- **MUST** LARAVEL-06.6 use backed enums for every fixed set of values (statuses, types, roles) and cast model columns to them. Put labels and colours on the enum as methods.
- **MUST** LARAVEL-06.7 use `match` instead of `switch`; let an unhandled case throw `UnhandledMatchError` rather than adding a silent default.
- **MUST** LARAVEL-06.8 use strict comparison (`===` , `in_array(..., strict: true)`).
- **SHOULD** LARAVEL-06.9 use named arguments when a call has a boolean or more than three positional arguments: `Number::currency($total, in: 'EUR')`.
- **MUST** LARAVEL-06.10 store money as integer minor units (cents) or with `brick/money`; **MUST NOT** use floats for money.
- **SHOULD** LARAVEL-06.11 use `CarbonImmutable` (set with `Date::use(CarbonImmutable::class)` in a service provider).

| Functions and flow

- **SHOULD** LARAVEL-06.12 return early instead of nesting `if` / `else`.
- **SHOULD** LARAVEL-06.13 use first-class callable syntax: `array_map($this->format(...), $lines)`.
- **MUST NOT** LARAVEL-06.14 use `compact()`, `extract()`, variable variables or `@` error suppression.

- **AVOID** LARAVEL-06.15 static helper classes; prefer an injected service or a method on the model or value object.
- **SHOULD** LARAVEL-06.16 use Laravel helpers (`str()`, `Str`, `Arr`, `Number`, `collect()`) over raw PHP string and array functions when they read more clearly.

PHP 8.5 features

PHP 8.5 is the minimum, so these are available everywhere:

- **SHOULD** LARAVEL-06.17 use the pipe operator `>` for short linear transformations of one value: `$slug = $title > trim(...) > Str::slug(...);`. **AVOID** pipes longer than four steps or with side effects; use a collection or a named method instead.
- **SHOULD** LARAVEL-06.18 use `clone($dto, ['dueAt' => $newDate])` (clone with) to derive a modified copy of a readonly DTO or value object instead of writing `with*()` methods by hand.
- **SHOULD** LARAVEL-06.19 mark methods whose return value must not be ignored (results, new immutable instances) with `#[\NoDiscard]`.
- **SHOULD** LARAVEL-06.20 use `array_first()` / `array_last()` over `reset()` / `end()` or `$array[array_key_first($array)]`.
- **MAY** LARAVEL-06.21 use attributes on constants and `#[\Override]` on properties; **MUST** use `#[\Override]` on every method that overrides a parent or implements an interface method outside the framework's magic hooks.

```
enum InvoiceStatus: string
{
    case Draft = 'draft';
    case Sent = 'sent';
    case Paid = 'paid';
    case Overdue = 'overdue';

    public function label(): string
    {
        return match ($this) {
            self::Draft => __('Draft'),
            self::Sent => __('Sent'),
            self::Paid => __('Paid'),
            self::Overdue => __('Overdue'),
        };
    }

    public function isOpen(): bool
    {
        return in_array($this, [self::Sent, self::Overdue], strict: true);
    }
}
```

07

Eloquent models: attributes for everything

Laravel 13 lets a model declare its configuration with PHP attributes instead of protected properties. **Every model in `Project` uses attributes for everything an attribute exists for.** A reader sees the table, mass-assignment rules, serialization, observers, factory, policy and collection at the top of the class, before any code.

Property → attribute map

All attributes below live in `Illuminate\Database\Eloquent\Attributes` unless noted.

INSTEAD OF	USE
<code>\$table</code> , <code>\$primaryKey</code> , <code>\$keyType</code> , <code>\$incrementing</code> , <code>\$timestamps</code> , <code>\$dateFormat</code>	<code>#[Table('invoices', key: ..., keyType: ..., incrementing: ..., timestamps: ..., dateFormat: ...)]</code>
<code>\$incrementing = false</code>	<code>#[WithoutIncrementing]</code>
<code>\$timestamps = false</code>	<code>#[WithoutTimestamps]</code>
<code>\$connection</code>	<code>#[Connection('reporting')]</code>
<code>\$fillable</code>	<code>#[Fillable([...])]</code>
<code>\$guarded</code> / <code>unguard()</code>	<code>#[Guarded([...])]</code> / <code>#[Unguarded]</code> (not allowed, see rules)
<code>\$hidden</code> / <code>\$visible</code>	<code>#[Hidden([...])]</code> / <code>#[Visible([...])]</code>
<code>\$appends</code>	<code>#[Appends([...])]</code>
<code>\$touches</code>	<code>#[Touches(['customer'])]</code>
<code>refresh()</code> after every save	<code>#[Refreshes]</code>
<code>scopeOverdue()</code> naming convention	<code>#[Scope]</code> on a protected method
<code>static::addGlobalScope()</code> in <code>booted()</code>	<code>#[ScopedBy([TenantScope::class])]</code>
<code>static::observe()</code> in a provider	<code>#[ObservedBy([InvoiceObserver::class])]</code>

INSTEAD OF	USE
<code>newCollection()</code> override	<code>#[CollectedBy(InvoiceCollection::class)]</code>
<code>newFactory()</code> / factory guessing	<code>#[UseFactory(InvoiceFactory::class)]</code>
<code>newEloquentBuilder()</code> override	<code>#[UseEloquentBuilder(InvoiceBuilder::class)]</code>
<code>Gate::policy()</code> / policy guessing	<code>#[UsePolicy(InvoicePolicy::class)]</code>
<code>toResource()</code> guessing	<code>#[UseResource(InvoiceResource::class)]</code> , <code>#[UseResourceCollection(...)]</code>
<code>bootHasX()</code> / <code>initializeHasX()</code> trait naming	<code>#[Boot]</code> / <code>#[Initialize]</code> on the trait method

Three things still have no attribute and stay as code: **casts** (the `casts()` method), **default values** (`$attributes`) and `$dispatchesEvents` . Keep those directly below the `use` statements.

Rules

- **MUST** LARAVEL-07.1 configure every model through attributes wherever one exists. **MUST NOT** declare `$table` , `$fillable` , `$guarded` , `$hidden` , `$visible` , `$appends` , `$touches` , `$connection` , `$primaryKey` , `$keyType` , `$incrementing` or `$timestamps` properties.
- **MUST** LARAVEL-07.2 list mass-assignable columns with `#[Fillable]` . **MUST NOT** use `#[Unguarded]` or an empty `#[Guarded([])]` .
- **MUST** LARAVEL-07.3 declare `#[Table]` explicitly even when it matches the convention, so the table name is greppable.
- **MUST** LARAVEL-07.4 wire the factory, policy and observers with `#[UseFactory]` , `#[UsePolicy]` and `#[ObservedBy]` rather than relying on name guessing or provider registration.
- **MUST** LARAVEL-07.5 hide secrets and internal columns with `#[Hidden]` .
- **MUST** LARAVEL-07.6 define local scopes with `#[Scope]` on a `protected` method that takes `Builder` `$query` and returns `void` . **MUST NOT** use the old `scopeX` prefix.
- **MUST** LARAVEL-07.7 write casts in the `casts()` method and cast every enum, date, boolean, JSON and encrypted column.
- **SHOULD** LARAVEL-07.8 order attributes: table and connection, mass assignment, serialization, relations (`Touches`), scopes, wiring (`ObservedBy` , `CollectedBy` , `UseFactory` , `UseEloquentBuilder` , `UsePolicy` , `UseResource`).
- **MUST** LARAVEL-07.9 declare models `final` .

UUIDv7 primary keys

- **MUST** LARAVEL-07.10 give every model a UUID primary key through the `HasUuids` trait. In Laravel 13, `HasUuids` generates **ordered UUIDv7** values, which index well and sort by creation time. No custom `newUniqueId()` is needed.
- **MUST** LARAVEL-07.11 create the column with `$table->uuid('id')->primary()` and foreign keys with `foreignUuid()` (§8).
- **MUST NOT** LARAVEL-07.12 expose integer IDs anywhere; route model binding uses the UUID.
- **MAY** LARAVEL-07.13 override `uniqueIds()` to also generate a UUID for another column, such as a public `reference`.

Split models with traits

Models grow. Keep the class file about configuration and behaviour, and move the rest into traits under `app/Models/Concerns/<Model>/`.

TRAIT	HOLDS
<code>Has<Model>Relationships</code>	All relationship methods
<code>Has<Model>Scopes</code>	<code>#[Scope]</code> methods (when there are more than three)
<code>Has<Model>Accessors</code>	Accessors and mutators (<code>Attribute::make</code>)
<code>Has<Model>Behaviour</code>	Optional: domain methods such as <code>markPaid()</code> once they outgrow the class

- **MUST** LARAVEL-07.14 move relationships into a `Has<Model>Relationships` trait for every model with more than two relationships.
- **MUST** LARAVEL-07.15 give every relationship method a return type with generics in the PHPDoc: `/** @return BelongsTo<Customer, $this> */`.
- **SHOULD** LARAVEL-07.16 mark trait boot logic with `#[Boot]` / `#[Initialize]` instead of relying on `bootTraitName()` naming.
- **MUST NOT** LARAVEL-07.17 put queries that belong to other models, HTTP concerns or notifications in a model trait.

The model

```
<?php

declare(strict_types=1);

namespace App\Models;

use App\Enums\InvoiceStatus;
use App\Http\Resources\InvoiceResource;
use App\Models\Builders\InvoiceBuilder;
use App\Models\Collections\InvoiceCollection;
use App\Models\Concerns\Invoice\HasInvoiceAccessors;
use App\Models\Concerns\Invoice\HasInvoiceRelationships;
use App\Models\Scopes\TenantScope;
use App\Observers\InvoiceObserver;
use App\Policies\InvoicePolicy;
use Database\Factories\InvoiceFactory;
use Illuminate\Database\Eloquent\Attributes\Append;
use Illuminate\Database\Eloquent\Attributes\CollectedBy;
use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Attributes\Hidden;
use Illuminate\Database\Eloquent\Attributes\ObservedBy;
use Illuminate\Database\Eloquent\Attributes\Scope;
use Illuminate\Database\Eloquent\Attributes\ScopedBy;
use Illuminate\Database\Eloquent\Attributes\Table;
use Illuminate\Database\Eloquent\Attributes\Touches;
use Illuminate\Database\Eloquent\Attributes\UseEloquentBuilder;
use Illuminate\Database\Eloquent\Attributes\UseFactory;
use Illuminate\Database\Eloquent\Attributes\UsePolicy;
use Illuminate\Database\Eloquent\Attributes\UseResource;
use Illuminate\Database\Eloquent\Builder;
use Illuminate\Database\Eloquent\Concerns\HasUuids;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

#[Table('invoices')]
#[Fillable(['customer_id', 'number', 'status', 'issued_at', 'due_at', 'paid_at', 'total_cents',
'currency', 'notes'])]
#[Hidden(['internal_notes'])]
#[Append(['is_overdue'])]
#[Touches(['customer'])]
#[ScopedBy([TenantScope::class])]
#[ObservedBy([InvoiceObserver::class])]
#[CollectedBy(InvoiceCollection::class)]
#[UseFactory(InvoiceFactory::class)]
#[UseEloquentBuilder(InvoiceBuilder::class)]
#[UsePolicy(InvoicePolicy::class)]
#[UseResource(InvoiceResource::class)]
final class Invoice extends Model
{
    /** @use HasFactory<InvoiceFactory> */
    use HasFactory;
```

```

use HasInvoiceAccessors;
use HasInvoiceRelationships;
use HasUuids;
use SoftDeletes;

/** @var array<string, mixed> */
protected $attributes = [
    'status' => InvoiceStatus::Draft,
    'currency' => 'EUR',
];

protected function casts(): array
{
    return [
        'status' => InvoiceStatus::class,
        'issued_at' => 'immutable_date',
        'due_at' => 'immutable_date',
        'paid_at' => 'immutable_datetime',
        'total_cents' => 'integer',
        'internal_notes' => 'encrypted',
    ];
}

public function markPaid(): void
{
    $this->update([
        'status' => InvoiceStatus::Paid,
        'paid_at' => now(),
    ]);
}

#[Scope]
protected function overdue(Builder $query): void
{
    $query->where('status', InvoiceStatus::Sent)
        ->whereDate('due_at', '<', today());
}
}

```

The relationships trait

```
<?php

declare(strict_types=1);

namespace App\Models\Concerns\Invoice;

use App\Models\Customer;
use App\Models\InvoiceLine;
use App\Models\Payment;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

trait HasInvoiceRelationships
{
    /** @return BelongsTo<Customer, $this> */
    public function customer(): BelongsTo
    {
        return $this->belongsTo(Customer::class);
    }

    /** @return HasMany<InvoiceLine, $this> */
    public function lines(): HasMany
    {
        return $this->hasMany(InvoiceLine::class)->orderBy('position');
    }

    /** @return HasMany<Payment, $this> */
    public function payments(): HasMany
    {
        return $this->hasMany(Payment::class);
    }
}
```

Pivot models

Pivot models follow the same rules: `#[Table('customer_tag')]`, `#[WithoutTimestamps]` when the pivot has none, and `HasUuids` when the pivot has its own `id`.

08

Migrations, factories and seeders

Migrations

- **MUST** LARAVEL-08.1 use anonymous migration classes (`return new class extends Migration`).
- **MUST** LARAVEL-08.2 use `$table->uuid('id')->primary()` for primary keys and `$table->foreignUuid('customer_id')->constrained()` for foreign keys.
- **MUST** LARAVEL-08.3 declare an explicit `onDelete` behaviour on every foreign key (`cascadeOnDelete()`, `restrictOnDelete()` or `nullOnDelete()`). Pick on purpose; there is no default answer.
- **MUST** LARAVEL-08.4 index every column used in a `where`, `orderBy` or join that runs on a hot path, and add composite indexes in the order the query filters.
- **MUST NOT** LARAVEL-08.5 edit a migration that has run in any shared environment. Write a new one.
- **SHOULD** LARAVEL-08.6 write a working `down()` method; when a change cannot be reversed safely, throw an exception in `down()` that says so.
- **MUST NOT** LARAVEL-08.7 use models in migrations. Use the query builder for data fixes, or move the fix into a one-off command.
- **SHOULD** LARAVEL-08.8 split schema changes and data backfills into separate migrations, and run large backfills as queued jobs.
- **SHOULD** LARAVEL-08.9 store money as `unsignedBigInteger('total_cents')` with a `char('currency', 3)` column, and enums as `string` columns (not native database enums).
- **SHOULD** LARAVEL-08.10 use `$table->timestampsTz()` / `softDeletesTz()` on PostgreSQL.

```

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('invoices', function (Blueprint $table): void {
            $table->uuid('id')->primary();
            $table->foreignUuid('customer_id')->constrained()->restrictOnDelete();
            $table->string('number')->unique();
            $table->string('status')->index();
            $table->date('issued_at')->nullable();
            $table->date('due_at')->nullable();
            $table->timestampTz('paid_at')->nullable();
            $table->unsignedBigInteger('total_cents')->default(0);
            $table->char('currency', 3);
            $table->text('notes')->nullable();
            $table->text('internal_notes')->nullable();
            $table->timestampsTz();
            $table->softDeletesTz();

            $table->index(['customer_id', 'status', 'due_at']);
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('invoices');
    }
};

```

Factories

- **MUST** LARAVEL-08.11 have a factory for every model, bound to it with `#[UseModel(Invoice::class)]` (`Illuminate\Database\Eloquent\Factories\Attributes`) instead of a `$model` property.
- **MUST** LARAVEL-08.12 produce a valid, saveable model from `definition()` with no extra state.
- **SHOULD** LARAVEL-08.13 express variations as named states: `→paid()`, `→overdue()`, `→forCustomer($customer)`.
- **MUST NOT** LARAVEL-08.14 hit external services or the filesystem from a factory.

```

#[UseModel(Invoice::class)]
final class InvoiceFactory extends Factory
{
    public function definition(): array
    {
        return [
            'customer_id' => Customer::factory(),
            'number' => fake()->unique()->numerify('INV-#####'),
            'status' => InvoiceStatus::Draft,
            'issued_at' => now(),
            'due_at' => now()->addDays(30),
            'total_cents' => fake()->numberBetween(1_000, 500_000),
            'currency' => 'EUR',
        ];
    }

    public function overdue(): self
    {
        return $this->state([
            'status' => InvoiceStatus::Sent,
            'due_at' => now()->subDays(5),
        ]);
    }
}

```

Seeders

- **MUST** LARAVEL-08.15 keep `DatabaseSeeder` runnable on an empty database to give a usable local environment in one command (`php artisan migrate:fresh --seed`).
- **MUST NOT** LARAVEL-08.16 seed production with demo data; production reference data goes in a migration or a dedicated, idempotent seeder.

09

Queries and performance

Strict mode

- **MUST** LARAVEL-09.1 enable strict mode outside production in `AppServiceProvider::boot()`:
`Model::shouldBeStrict(! app()→isProduction())`. This throws on lazy loading, on silently discarded fillable attributes and on missing attributes.
- **MUST** LARAVEL-09.2 log lazy-loading violations in production instead of throwing:
`Model::handleLazyLoadingViolationUsing(...)`.

Writing queries

- **MUST** LARAVEL-09.3 eager load every relation a view, resource or Inertia page uses:
`Invoice::with(['customer', 'lines'])`.
- **SHOULD** LARAVEL-09.4 use `withCount()`, `withSum()` and `withExists()` instead of loading a relation just to count or sum it.
- **MUST** LARAVEL-09.5 select only the columns you need on large tables and in exports.
- **MUST** LARAVEL-09.6 use `chunkById()`, `lazyById()` or `cursor()` to iterate over more than about 1,000 rows. **MUST NOT** call `→get()` or `→all()` on an unbounded table.
- **MUST** LARAVEL-09.7 paginate every list endpoint and screen; prefer `cursorPaginate()` for infinite scroll and APIs.
- **SHOULD** LARAVEL-09.8 move reusable query logic into `#[Scope]` methods or a custom builder bound with `#[UseEloquentBuilder]`. A query used in more than one place with more than three clauses becomes a scope, builder method or `app/Queries` class.
- **MUST NOT** LARAVEL-09.9 build SQL by concatenating input. Use bindings; `DB::raw()` and `whereRaw()` take only constants or bound values.
- **MUST** LARAVEL-09.10 wrap multi-write operations in `DB::transaction()`, and dispatch side effects with `→afterCommit()`.
- **SHOULD** LARAVEL-09.11 use `upsert()` or `insert()` for bulk writes instead of saving models in a loop (remember that bulk writes skip model events).
- **MAY** LARAVEL-09.12 use `whereVectorSimilarTo()` (Laravel 13, PostgreSQL + pgvector) for semantic search; keep embeddings in their own column and index.

Caching

- **SHOULD** LARAVEL-09.13 cache expensive reads with `Cache::flexible()` (stale-while-revalidate) and tagged keys; use `Cache::touch()` to extend a TTL without re-reading the value.
- **MUST** LARAVEL-09.14 namespace cache keys by model and ID: `invoice:{id}:totals`.
- **MUST** LARAVEL-09.15 clear or update caches from the observer or action that changes the data, never from a controller.

A custom builder

```
/** @extends Builder<Invoice> */
final class InvoiceBuilder extends Builder
{
    public function forCustomer(Customer $customer): self
    {
        return $this->whereBelongsTo($customer);
    }

    public function open(): self
    {
        return $this->whereIn('status', [InvoiceStatus::Sent, InvoiceStatus::Overdue]);
    }
}

// Usage
$invoices = Invoice::query()
    ->forCustomer($customer)
    ->open()
    ->with('customer')
    ->withSum('payments', 'amount_cents')
    ->latest('issued_at')
    ->paginate(25);
```

10

Routing and controllers

Routes

- **MUST** LARAVEL-10.1 name every route, and generate URLs with `route()` (or Wayfinder / Ziggy on the Inertia stack). **MUST NOT** hard-code paths.
- **MUST** LARAVEL-10.2 use resource routes (`Route::resource()`, `Route::apiResource()`) with `→only()` for CRUD, and invokable controllers for single actions.
- **MUST** LARAVEL-10.3 use route model binding with the UUID key; scope nested bindings with `→scopeBindings()`.
- **MUST NOT** LARAVEL-10.4 put closures with logic in route files; closures may only return a view.
- **SHOULD** LARAVEL-10.5 rate-limit authentication, password reset and public API routes with named limiters defined in `AppServiceProvider`.

```
Route::middleware(['auth', 'verified'])→group(function (): void {  
    Route::resource('invoices', InvoiceController::class)→only(['index', 'show', 'store',  
    'update']);  
    Route::post('invoices/{invoice}/send', SendInvoiceController::class)→name('invoices.send');  
});
```

Controllers: attributes, not constructors

Laravel 10 lets controllers declare middleware and authorization with attributes from `Illuminate\Routing\Attributes\Controllers`.

- **MUST** LARAVEL-10.6 declare controller middleware with `#[Middleware]` on the class or method, using `only:` / `except:` where needed. **MUST NOT** implement `HasMiddleware` or call `$this→middleware()`.
- **SHOULD** LARAVEL-10.7 declare per-action authorization with `#[Authorize]` instead of a `Gate::authorize()` call at the top of the method.
- **MUST** LARAVEL-10.8 keep controllers thin: validate (form request), authorize, call one action, return a response. Target fewer than 15 lines per method.
- **MUST NOT** LARAVEL-10.9 query or write models directly in a controller beyond a simple read for `index` / `show`.
- **MUST** LARAVEL-10.10 return typed responses: `View`, `Response` (Inertia), `RedirectResponse`, `JsonResource`.

```

#[Middleware('auth')]
#[Middleware('throttle:invoices', only: ['store', 'update'])]
final class InvoiceController
{
    #[Authorize('viewAny', Invoice::class)]
    public function index(): InertiaResponse
    {
        return Inertia::render('invoices/index', [
            'invoices' => InvoiceResource::collection(
                Invoice::query()>with('customer')>latest()>paginate(25),
            ),
        ]);
    }

    #[Authorize('create', Invoice::class)]
    public function store(StoreInvoiceRequest $request, CreateInvoice $createInvoice):
RedirectResponse
    {
        $invoice = $createInvoice->handle($request->user(), $request->toData());

        return to_route('invoices.show', $invoice)->with('status', __('Invoice created.'));
    }
}

```

Livewire vs Inertia routing

	LIVEWIRE STACK	INERTIA STACK
Full-page screen	<code>Route::livewire('/invoices', 'pages::invoices.index')</code> — the component is the controller	Controller method returns <code>Inertia::render('invoices/index', \$props)</code>
Form submit	Component method (<code>save()</code>) calls an action	<code>POST</code> to a controller that calls an action and redirects
Simple static page	<code>Route::view()</code>	<code>Route::inertia('/about', 'about')</code>

- **MUST** LARAVEL-10.11 keep the same rule on both stacks: the entry point (component or controller) validates, authorizes and delegates to an action.

11

Validation and form requests

- **MUST** LARAVEL-11.1 validate every write in a form request (controllers) or a Livewire form object (Livewire). **MUST NOT** call `$request→validate()` inline in a controller.
- **MUST** LARAVEL-11.2 configure form-request behaviour with attributes from `Illuminate\Foundation\Http\Attributes`: `#[StopOnFirstFailure]`, `#[ErrorBag('invoice')]`, `#[RedirectTo('/invoices')]` or `#[RedirectToRoute('invoices.index')]`. **MUST NOT** set the matching `$stopOnFirstFailure`, `$errorBag`, `$redirect` or `$redirectToRoute` properties.
- **MUST** LARAVEL-11.3 write rules as arrays, not pipe strings, and use rule objects (`Rule::enum()`, `Rule::unique()`, `Password::defaults()`, `File::types()`).
- **MUST** LARAVEL-11.4 read validated data only: `$request→validated()`, `$request→safe()`, or a typed DTO built from them. **MUST NOT** pass `$request→all()` to a model.
- **SHOULD** LARAVEL-11.5 expose a `toData()` method that turns validated input into a readonly DTO for the action.
- **SHOULD** LARAVEL-11.6 put authorization in the policy and leave `authorize()` out of the form request (it defaults to `true` when absent), so authorization lives in one place.
- **SHOULD** LARAVEL-11.7 write custom rules as invokable rule classes in `app/Rules`; **AVOID** long closure rules.
- **MUST** LARAVEL-11.8 translate messages through `lang/` files; no hard-coded English in `messages()`.

```

#[StopOnFirstFailure]
#[ErrorBag('invoice')]
final class StoreInvoiceRequest extends FormRequest
{
    /** @return array<string, array<int, mixed>> */
    public function rules(): array
    {
        return [
            'customer_id' => ['required', 'uuid', Rule::exists('customers', 'id')],
            'due_at' => ['required', 'date', 'after_or_equal:today'],
            'currency' => ['required', 'string', 'size:3'],
            'lines' => ['required', 'array', 'min:1', 'max:200'],
            'lines.*.description' => ['required', 'string', 'max:255'],
            'lines.*.quantity' => ['required', 'integer', 'min:1'],
            'lines.*.unit_price_cents' => ['required', 'integer', 'min:0'],
        ];
    }

    public function toData(): CreateInvoiceData
    {
        return CreateInvoiceData::fromArray($this->validated());
    }
}

```

On each frontend stack

- **Livewire:** **MUST** LARAVEL-11.9 validate with a form object (`Livewire\Form`) using `#[Validate]` attributes on its properties, or a `rules()` method for dynamic rules. Call `$this->form->validate()` before handing data to an action.
- **Inertia:** **MUST** LARAVEL-11.10 rely on the form request; validation errors arrive in `useForm().errors` (or the `<Form>` component's `errors`) automatically. **MUST NOT** duplicate server rules as the only check on the client; client-side checks are for convenience only.

12

Authorization

- **MUST** LARAVEL-12.1 authorize every action that reads or changes a model through a **policy**, bound to the model with `#[UsePolicy]`.
- **MUST** LARAVEL-12.2 authorize at the entry point: `#[Authorize]` on controller methods, `$this->authorize()` in Livewire actions, and `Gate::authorize()` in jobs or commands that act for a user.
- **MUST** LARAVEL-12.3 re-authorize in every Livewire action method. A component's `mount()` check does not cover later requests to the same component.
- **MUST** LARAVEL-12.4 return `Response::denyAsNotFound()` from a policy when the user should not learn that the record exists (for example, another tenant's invoice).
- **SHOULD** LARAVEL-12.5 use `before()` in a policy only for true super-admins, and log its use.
- **MUST NOT** LARAVEL-12.6 check roles inline (`if ($user->role === 'admin')`) outside policies and gates.
- **SHOULD** LARAVEL-12.7 pass abilities to the frontend instead of re-deriving rules there: `'can' => ['update' => $user->can('update', $invoice)]` in an Inertia prop or resource, and `@can` / `$user->can()` in Blade and Livewire.
- **MAY** LARAVEL-12.8 inject the current user with the `#[CurrentUser]` container attribute instead of calling `auth()->user()`.

```
final class InvoicePolicy
{
    public function view(User $user, Invoice $invoice): Response
    {
        return $user->team_id === $invoice->customer->team_id
            ? Response::allow()
            : Response::denyAsNotFound();
    }

    public function update(User $user, Invoice $invoice): bool
    {
        return $user->team_id === $invoice->customer->team_id
            && $invoice->status === InvoiceStatus::Draft;
    }
}
```

13

Actions, services and DTOs

Business logic lives in **actions**: small classes that do one use case. The same action serves a controller, a Livewire component, a job, a command and a test, so both frontend stacks share one core.

Actions

- **MUST** LARAVEL-13.1 give each action one public method, `handle()`, with typed parameters and a typed return.
- **MUST** LARAVEL-13.2 inject dependencies through the constructor; resolve actions from the container (method injection or `app()`), never with `new`.
- **MUST** LARAVEL-13.3 wrap multi-step writes in `DB::transaction()` inside the action.
- **SHOULD** LARAVEL-13.4 compose actions by injecting one into another, not by inheritance.
- **MUST NOT** LARAVEL-13.5 read the request, session or `auth()` inside an action; pass the user and data in.

```

final readonly class CreateInvoice
{
    public function __construct(
        private NextInvoiceNumber $nextNumber,
    ) {}

    public function handle(User $user, CreateInvoiceData $data): Invoice
    {
        return DB::transaction(function () use ($user, $data): Invoice {
            $invoice = Invoice::create([
                'customer_id' => $data->customerId,
                'number' => $this->nextNumber->for($user->team),
                'due_at' => $data->dueAt,
                'currency' => $data->currency,
                'total_cents' => $data->totalCents(),
            ]);

            $invoice->lines()->createMany($data->linesForInsert());

            InvoiceCreated::dispatch($invoice);

            return $invoice;
        });
    }
}

```

DTOs

- **MUST** LARAVEL-13.6 pass structured data between layers as `final readonly class` DTOs in `app/Data`, not as arrays.
- **SHOULD** LARAVEL-13.7 give each DTO a named constructor (`fromArray()`, `fromRequest()`) and keep validation out of it (the form request already did that).
- **MAY** LARAVEL-13.8 use `spatie/laravel-data` when DTOs also need to serialize to TypeScript for the Inertia stack; keep the same naming.

Services and container attributes

- **SHOULD** LARAVEL-13.9 use a service class only for an integration with state or configuration (a payment gateway client, a PDF renderer). Business rules still go in actions.
- **MUST** LARAVEL-13.10 bind interfaces to implementations with class attributes — `#` `[Bind(StripeGateway::class)]` on the interface, `#[Singleton]` or `#[Scoped]` on the class — instead of registering them in a service provider, when the binding is static.
- **SHOULD** LARAVEL-13.11 inject configuration and framework services with parameter attributes from `Illuminate\Container\Attributes`: `#[Config('billing.currency')] string $currency`, `#` `[Storage('invoices')] Filesystem $disk`, `#[Log('billing')] LoggerInterface $log`, `#[CurrentUser] User $user`, `#[Tag('exporters')] iterable $exporters`.

- **MUST NOT** LARAVEL-13.12 call `config()`, `Storage::disk()` or `auth()` inside a constructor when a container attribute can inject the value.

```
#[Bind(StripePaymentGateway::class)]
interface PaymentGateway
{
    public function charge(Invoice $invoice): PaymentResult;
}

#[Singleton]
final readonly class StripePaymentGateway implements PaymentGateway
{
    public function __construct(
        #[Config('services.stripe.secret')] private string $secret,
        #[Log('billing')] private LoggerInterface $log,
    ) {}

    // ...
}
```

14

Frontend: Livewire or Inertia

Project uses one of two stacks per application. Pick one at the start and do not mix them in the same app.

	LIVEWIRE STACK	INERTIA STACK
Versions	Livewire 4 + Flux UI Pro + Tailwind 4	Inertia v3 + React 19 + TypeScript + Tailwind 4
Choose when	The team is PHP-first; screens are forms, tables and dashboards	Rich client interactions, a design system in React, or shared code with a React Native / SPA client
State lives	On the server, in the component	In React, with props from the server
Data contract	Public properties and computed values	Props built from API resources, typed in TypeScript

Shared rules (both stacks)

- MUST** LARAVEL-14.1 keep UI components **small**: one job per component, roughly under 150 lines of template. Split a screen into a page plus focused child components (`InvoiceTable` , `InvoiceFilters` , `InvoiceStatusBadge`).
- MUST** LARAVEL-14.2 keep business logic out of the view layer: components call actions; they never write models directly with complex rules.
- MUST** LARAVEL-14.3 localise every user-facing string (`__()` in PHP/Blade, a translation helper in React).
- MUST** LARAVEL-14.4 meet WCAG 2.2 AA: labelled inputs, visible focus, keyboard-reachable actions, colour contrast.
- SHOULD** LARAVEL-14.5 share layouts: one app layout, one guest layout.

Livewire 4

Components

- SHOULD** LARAVEL-14.6 write components as **single-file components** (`resources/views/components/⚡ invoice-table.blade.php`), the Livewire 4 default. Switch to a **multi-file component** (`--mfc`) once the PHP part passes about 100 lines, and keep the colocated `*.test.php` file there.
- MUST** LARAVEL-14.7 put full-page components in the `pages::` namespace and route them with `Route::livewire()` .

- **MUST** LARAVEL-14.8 use **Flux UI** components (`<flux:input>` , `<flux:button>` , `<flux:table>` , `<flux:modal>`) before writing custom markup; wrap repeated Flux combinations in an anonymous Blade component.
- **SHOULD** LARAVEL-14.9 use plain Blade components (no Livewire) for anything without server state: badges, cards, empty states.

State and security

- **MUST** LARAVEL-14.10 treat every public property as user input: anyone can change it from the browser. Mark IDs and anything the user must not change with `#[Locked]` .
- **MUST** LARAVEL-14.11 store models in properties (Livewire locks the model ID) or as `#[Locked]` UUIDs, never a raw editable ID.
- **MUST** LARAVEL-14.12 authorize inside every action method (`$this->authorize('update', $this->invoice)`), not only in `mount()` .
- **MUST** LARAVEL-14.13 move form fields into a `Livewire\Form` object with `#[Validate]` attributes when a form has more than three fields.
- **SHOULD** LARAVEL-14.14 use `#[Computed]` for derived data (queries, totals) instead of storing it in public properties; this keeps the payload small and the data fresh.
- **SHOULD** LARAVEL-14.15 use `#[Url]` for filters, search and sort so screens are linkable.
- **SHOULD** LARAVEL-14.16 listen for events with `#[On('invoice-paid')]` ; **AVOID** long chains of component-to-component events — lift the state up or use a shared parent.

Performance

- **MUST** LARAVEL-14.17 default to `wire:model` (sync on next request); use `wire:model.live.debounce.300ms` only where instant feedback matters (search).
- **SHOULD** LARAVEL-14.18 wrap independently refreshing parts of a page in `@island` blocks (with `lazy:true` for slow widgets) instead of splitting into many nested components.
- **SHOULD** LARAVEL-14.19 mark slow components lazy (`#[Lazy]` or `lazy` on the tag) and provide an `@placeholder` .
- **SHOULD** LARAVEL-14.20 use the automatic `data-loading` attribute with Tailwind variants for loading states instead of hand-written `wire:loading` blocks.
- **MAY** LARAVEL-14.21 use `wire:sort` for drag-and-drop ordering and `wire:transition` for page transitions.

```

<?php

use App\Actions\Invoices\MarkInvoicePaid;
use App\Models\Invoice;
use Livewire\Attributes\Computed;
use Livewire\Attributes\Url;
use Livewire\Component;
use Livewire\WithPagination;

new class extends Component
{
    use WithPagination;

    #[Url]
    public string $search = '';

    #[Computed]
    public function invoices()
    {
        return Invoice::query()
            →with('customer')
            →when($this→search, fn ($q) ⇒ $q→whereLike('number', "%{$this→search}%"))
            →latest('issued_at')
            →paginate(25);
    }

    public function markPaid(Invoice $invoice, MarkInvoicePaid $markPaid): void
    {
        $this→authorize('update', $invoice);

        $markPaid→handle($invoice);
    }
};
?>

<div class="space-y-4">
    <flux:input wire:model.live.debounce.300ms="search" icon="magnifying-glass"
    :placeholder="__('Search invoices')" />

    <flux:table :paginate="$this→invoices">
        @foreach ($this→invoices as $invoice)
            <x-invoices.row :$invoice wire:key="{ $invoice→id }" />
        @endforeach
    </flux:table>
</div>

```

Inertia v3 + React

Pages and components

- MUST** LARAVEL-14.22 write one page per file under `resources/js/pages`, matching the `Inertia::render()` name (`invoices/show` → `pages/invoices/show.tsx`).

- **MUST** LARAVEL-14.23 keep pages thin: they receive props and compose small components from `resources/js/components`.
- **MUST** LARAVEL-14.24 use TypeScript in strict mode; **MUST NOT** use `any`. Type page props with an interface per page.
- **SHOULD** LARAVEL-14.25 generate route helpers and TypeScript types from PHP (Wayfinder for routes and actions; `spatie/laravel-typescript-transformer` or `laravel-data` for DTOs and enums) instead of hand-writing them.
- **SHOULD** LARAVEL-14.26 use persistent layouts and `useLayoutProps` / `setLayoutProps` (v3) for layout state such as page titles and breadcrumbs, instead of context or event buses.

Props and data

- **MUST** LARAVEL-14.27 build props from API resources (`InvoiceResource::make($invoice)`), never pass raw models — everything in a prop is visible in the page source.
- **MUST** LARAVEL-14.28 keep shared data in `HandleInertiaRequests::share()` minimal (auth user summary, flash, abilities) and lazy (`fn () => ...`).
- **SHOULD** LARAVEL-14.29 use `Inertia::defer()` for slow, non-critical props and `Inertia::optional()` for props loaded only on partial reloads; use `Inertia::merge()` / `Inertia::scroll()` for infinite lists.
- **SHOULD** LARAVEL-14.30 use partial reloads (`router.reload({ only: ['invoices'] })`) instead of full visits when only one prop changes.

Forms and requests

- **MUST** LARAVEL-14.31 submit forms with the `<Form>` component or `useForm`, and show `errors` from the server next to each field.
- **SHOULD** LARAVEL-14.32 use v3 **optimistic updates** (`form.optimistic(...)` , `router.optimistic(...)`) for quick toggles and list changes; Inertia rolls them back on a non-2xx response.
- **MUST** LARAVEL-14.33 use `useHttp` (v3's built-in client) for JSON endpoints that do not return an Inertia response. **MUST NOT** add Axios unless you need its interceptors.
- **MUST** LARAVEL-14.34 redirect after a successful `POST` / `PUT` / `DELETE`; **MUST NOT** return `Inertia::render()` from a write route.

```

import { Form, Head } from '@inertiajs/react';
import { update } from '@actions/App/Http/Controllers/InvoiceController';
import { InvoiceStatusBadge } from '@components/invoices/InvoiceStatusBadge';
import { Button, Field } from '@components/ui';
import type { Invoice } from '@types/generated';

interface Props {
  invoice: Invoice;
  can: { update: boolean };
}

export default function ShowInvoice({ invoice, can }: Props) {
  return (
    <
      <Head title={invoice.number} />
      <header className="flex items-center gap-3">
        <h1 className="text-xl font-semibold">{invoice.number}</h1>
        <InvoiceStatusBadge status={invoice.status} />
      </header>

      {can.update && (
        <Form {...update.form(invoice.id)}>
          ({ { errors, processing } }) => (
            <Field name="due_at" type="date" defaultValue={invoice.due_at} error={errors.due_at} />

            <Button type="submit" disabled={processing}>Save</Button>
          )
        </Form>
      )}
    </>
  );
}

```

Inertia setup

- **MUST** LARAVEL-14.35 install the `@inertiajs/vite` plugin and call `createInertiaApp()` with `strictMode: true`.
- **SHOULD** LARAVEL-14.36 enable SSR for public, indexable pages (`vite build --ssr` + `php artisan inertia:start-ssr`); authenticated app screens **MAY** skip it.

15

APIs and resources

- **MUST** LARAVEL-15.1 return every API response (and every Inertia prop built from a model) through an API resource. **MUST NOT** return a model or `toArray()` directly.
- **MUST** LARAVEL-15.2 configure resources with attributes from `Illuminate\Http\Resources\Attributes`: `#[Collects(InvoiceResource::class)]` on a resource collection, `#[PreserveKeys]` when keys matter. **MUST NOT** set the `$collects` or `$preserveKeys` properties.
- **SHOULD** LARAVEL-15.3 use Laravel 13's first-party **JSON:API resources** for public or partner APIs, so filtering, sparse fieldsets, includes and links follow the spec without custom code. Internal APIs **MAY** use plain resources.
- **MUST** LARAVEL-15.4 use `whenLoaded()`, `whenCounted()` and `whenAggregated()` so a resource never triggers a query.
- **MUST** LARAVEL-15.5 format money, dates and enums consistently: money as integer minor units plus currency, dates as ISO 8601 in UTC, enums as their backing value.
- **MUST** LARAVEL-15.6 version public APIs in the URL (`/api/v1/...`) and keep a version stable once published.
- **MUST** LARAVEL-15.7 authenticate APIs with Sanctum (first-party SPA cookies or tokens with abilities) and rate-limit every route.
- **MUST** LARAVEL-15.8 return errors in one shape (`message` + `errors` , or JSON:API `errors`) with correct status codes: 401, 403, 404, 409, 422, 429.
- **SHOULD** LARAVEL-15.9 document public APIs with an OpenAPI spec generated from code (for example Scramble) and check it in CI.

```

final class InvoiceResource extends JsonResource
{
    /** @return array<string, mixed> */
    public function toArray(Request $request): array
    {
        return [
            'id' => $this->id,
            'number' => $this->number,
            'status' => $this->status,
            'status_label' => $this->status->label(),
            'due_at' => $this->due_at->toDateString(),
            'total' => ['amount' => $this->total_cents, 'currency' => $this->currency],
            'is_overdue' => $this->is_overdue,
            'customer' => CustomerResource::make($this->whenLoaded('customer')),
            'lines_count' => $this->whenCounted('lines'),
        ];
    }
}

#[Collects(InvoiceResource::class)]
final class InvoiceResourceCollection extends ResourceCollection {}

```

16

Jobs, events, queues and commands

Jobs: attributes for everything

Laravel 13 job attributes live in `Illuminate\Queue\Attributes`.

INSTEAD OF	USE
<code>\$tries</code>	<code>#[Tries(3)]</code>
<code>\$timeout</code>	<code>#[Timeout(120)]</code>
<code>\$backoff</code> / <code>backoff()</code>	<code>#[Backoff([10, 60, 300])]</code>
<code>\$maxExceptions</code>	<code>#[MaxExceptions(2)]</code>
<code>\$failOnTimeout</code>	<code>#[FailOnTimeout]</code>
<code>\$deleteWhenMissingModels</code>	<code>#[DeleteWhenMissingModels]</code>
<code>\$uniqueFor</code>	<code>#[UniqueFor(3600)]</code>
<code>onQueue()</code> in the constructor	<code>#[Queue('billing')]</code>
<code>onConnection()</code>	<code>#[Connection('redis')]</code>
<code>withoutRelations()</code> on each model	<code>#[WithoutRelations]</code>

- MUST** LARAVEL-16.1 configure retries, timeouts, backoff, queue and connection with these attributes. **MUST NOT** declare the matching properties.
- MUST** LARAVEL-16.2 set `#[Tries]` and `#[Timeout]` on every job, and keep the timeout below the queue's `retry_after`.
- MUST** LARAVEL-16.3 make jobs **idempotent**: running twice must not charge twice or send two emails. Use `ShouldBeUnique`, a guard on the model state, or an idempotency key.
- MUST** LARAVEL-16.4 pass models or IDs, never large arrays or closures; add `#[WithoutRelations]` so queued payloads stay small.
- MUST** LARAVEL-16.5 implement `failed()` for jobs whose failure a person must hear about.
- SHOULD** LARAVEL-16.6 use `Queue::route()` (Laravel 13) in a service provider to route job classes to queues centrally when many jobs share a queue; use `#[Queue]` for one-off placement.

- **SHOULD** LARAVEL-16.7 dispatch after commit (`→afterCommit()` or `after_commit` in config) when the job reads data written in the same transaction.
- **SHOULD** LARAVEL-16.8 use `Bus::batch()` for fan-out work with progress, and `Bus::chain()` for ordered steps.

```
#[Queue('billing')]
#[Tries(5)]
#[Timeout(120)]
#[Backoff([10, 60, 300])]
#[DeleteWhenMissingModels]
#[WithoutRelations]
final class GenerateInvoicePdf implements ShouldQueue, ShouldBeUnique
{
    use Queueable;

    public function __construct(public Invoice $invoice) {}

    public function uniqueId(): string
    {
        return $this->invoice->id;
    }

    public function handle(RenderInvoicePdf $render): void
    {
        $render->handle($this->invoice);
    }
}
```

Events and listeners

- **MUST** LARAVEL-16.9 name events in the past tense and keep them as plain data carriers (public readonly properties).
- **SHOULD** LARAVEL-16.10 make listeners that do I/O queued (`ShouldQueue`), and rely on event discovery instead of registering listeners by hand.
- **AVOID** LARAVEL-16.11 using model events (`creating` , `saved`) for business workflows. Use them in observers for model housekeeping only (setting a number, clearing a cache); dispatch domain events from actions.
- **SHOULD** LARAVEL-16.12 broadcast real-time updates with Reverb; on Livewire listen with `#On('echo: ...')`, on Inertia with `useEcho` from `@Laravel/echo-react`.

Console commands

- **MUST** LARAVEL-16.13 declare commands with attributes from `Illuminate\Console\Attributes`: `#Signature('invoices:remind [--days=3]')`, `#Description('...')`, and `#Help`, `#Usage` or `#Hidden` as needed. **MUST NOT** set `$signature` or `$description`.

- **MUST** LARAVEL-16.14 keep commands thin: parse input, call an action, report the result. Return `self::SUCCESS` / `self::FAILURE`.
- **MUST** LARAVEL-16.15 schedule tasks in `routes/console.php` with `→withoutOverlapping()` and `→onOneServer()` for anything that must not run twice.

```
#[Signature('invoices:remind [--days=3 : Days before the due date]')]
#[Description('Send reminders for invoices that are due soon.')]
final class RemindDueInvoices extends Command
{
    public function handle(SendDueReminders $sendReminders): int
    {
        $count = $sendReminders→handle(days: (int) $this→option('days'));

        $this→components→info("Queued {$count} reminders.");

        return self::SUCCESS;
    }
}
```

17

Configuration and environment

- **MUST** LARAVEL-17.1 call `env()` only inside `config/*.php`. Everywhere else read `config()` or inject with `#[Config('billing.currency')]`; `env()` returns `null` once config is cached.
- **MUST** LARAVEL-17.2 keep `.env.example` complete and current, with safe placeholder values and a comment for anything unusual.
- **MUST NOT** LARAVEL-17.3 commit `.env` or any secret. Production secrets live in the host's secret store or an encrypted `.env.production.encrypted` (`php artisan env:encrypt`).
- **SHOULD** LARAVEL-17.4 group app-specific settings in their own files (`config/billing.php`) rather than adding keys to `config/app.php`.
- **MUST** LARAVEL-17.5 run `php artisan optimize` on deploy (config, routes, views, events cached).
- **SHOULD** LARAVEL-17.6 use Pennant feature flags for gradual rollouts instead of config booleans that need a deploy.
- **MUST** LARAVEL-17.7 keep timezone `UTC` in config and the database; convert to the user's timezone only for display.

18

Errors, logging and observability

Exceptions

- **MUST** LARAVEL-18.1 throw domain exceptions (`InvoiceAlreadyPaid extends DomainException`) for broken business rules, and give each a clear message.
- **SHOULD** LARAVEL-18.2 let an exception render itself (`render()` returning a response) or register its rendering in `bootstrap/app.php` with `→withExceptions()`. **MUST NOT** catch an exception only to rethrow or ignore it.
- **MUST** LARAVEL-18.3 catch exceptions only where you can recover or add context; otherwise let them reach the handler.
- **MUST NOT** LARAVEL-18.4 leak stack traces or SQL to users: `APP_DEBUG=false` everywhere except local.
- **SHOULD** LARAVEL-18.5 throttle noisy exceptions with `→throttle()` in `withExceptions()` and stop reporting expected ones with `→dontReport()`.

Logging

- **MUST** LARAVEL-18.6 log with context arrays, not interpolated strings: `Log::info('Invoice sent', ['invoice_id' => $invoice->id])`.
- **SHOULD** LARAVEL-18.7 add request-wide context with the `Context` facade (request ID, user ID, tenant); it flows into logs and queued jobs automatically.
- **MUST NOT** LARAVEL-18.8 log secrets, tokens, passwords, full card data or personal data beyond IDs.
- **SHOULD** LARAVEL-18.9 use dedicated channels for noisy areas (`#[Log('billing')]`) and JSON formatting in production.

Observability

- **SHOULD** LARAVEL-18.10 run Pulse in production for slow queries, slow jobs, exceptions and usage, and Horizon for queue health.
- **SHOULD** LARAVEL-18.11 send exceptions to an error tracker (Nightwatch, Sentry or Flare) with release and environment tags.
- **MUST** LARAVEL-18.12 expose the built-in `/up` health route and monitor it.

19

Security

- **MUST** LARAVEL-19.1 keep Laravel 13's `PreventRequestForgery` middleware (origin-aware CSRF protection) enabled for all web routes; exclude a route only for signed webhooks, and verify the signature.
- **MUST** LARAVEL-19.2 mass-assign only through `#[Fillable]` lists and validated data (§7, §11).
- **MUST** LARAVEL-19.3 escape output: `{{ }}` in Blade. `{!! !!}` is allowed only for HTML sanitised on the server, with a comment saying where. In React, **MUST NOT** use `dangerouslySetInnerHTML` without the same sanitising.
- **MUST** LARAVEL-19.4 authorize every route, Livewire action and job that acts for a user (§12), and scope queries to the current tenant with a global scope (`#[ScopedBy]`).
- **MUST** LARAVEL-19.5 use signed URLs (`URL::temporarySignedRoute()`) for links in emails and downloads.
- **MUST** LARAVEL-19.6 store uploads on a private disk and serve them through an authorized route or temporary URL; validate type and size with `File::types()→max()` .
- **MUST** LARAVEL-19.7 hash passwords with the default hasher and enforce `Password::defaults()` set in a provider (min length, mixed case, uncompromised in production).
- **MUST** LARAVEL-19.8 cast secrets and personal data columns as `encrypted` where the app does not need to query them.
- **MUST** LARAVEL-19.9 set security headers (CSP, HSTS, `X-Frame-Options` , `Referrer-Policy`) through middleware.
- **MUST** LARAVEL-19.10 run `composer audit` and `npm audit` in CI and fix high or critical issues before release.
- **MUST NOT** LARAVEL-19.11 expose Telescope, Horizon, Pulse or debug tools in production without an authorization gate.

20

Testing with Pest

What to test

LAYER	TEST TYPE	LOCATION
Actions, value objects, enums, DTOs	Unit	<code>tests/Unit/...</code>
Controllers, form requests, policies, jobs, commands	Feature (HTTP / dispatch)	<code>tests/Feature/...</code>
Livewire components	Livewire feature test (<code>Livewire::test()</code>)	Next to an MFC component, or <code>tests/Feature/Livewire/...</code>
Inertia pages	Feature test with <code>assertInertia()</code>	<code>tests/Feature/...</code>
Critical journeys (sign-up, checkout, pay invoice)	Browser (Pest 4 browser plugin)	<code>tests/Browser/...</code>
React components (Inertia stack)	Vitest + Testing Library	Next to the component: <code>InvoiceStatusBadge.test.tsx</code>

Rules

- MUST** LARAVEL-20.1 write tests with Pest, using `it()` / `test()` descriptions that read as sentences.
- MUST** LARAVEL-20.2 cover every action, policy and form request; a bug fix starts with a failing test.
- MUST** LARAVEL-20.3 use `RefreshDatabase` (or `LazilyRefreshDatabase`) and factories. **MUST NOT** depend on seeded data or on test order.
- MUST** LARAVEL-20.4 configure test classes and hooks with attributes from `Illuminate\Foundation\Testing\Attributes`, such as `#[Seed]` / `#[Seeder(BillingSeeder::class)]` instead of `$seed` / `$seeder` properties, and `#[SetUp]` / `#[TearDown]` on hook methods in reusable testing traits.
- MUST** LARAVEL-20.5 fake the outside world: `Http::fake()` with `Http::preventStrayRequests()`, `Queue::fake()`, `Mail::fake()`, `Notification::fake()`, `Storage::fake()`, `Event::fake()` (only the events you assert).
- SHOULD** LARAVEL-20.6 freeze time with `$this->travelTo()` / `freezeTime()` when dates matter.
- MUST** LARAVEL-20.7 run tests in parallel in CI (`pest --parallel`) and keep the suite under five minutes.

- **SHOULD** LARAVEL-20.8 add architecture tests with Pest's `arch()` to enforce this guide: strict types, final classes, no `env()` outside config, no debugging functions.
- **SHOULD** LARAVEL-20.9 keep coverage of `app/Actions` and `app/Policies` above 90% and the whole app above 80%.

```
it('marks an overdue invoice as paid', function (): void {
    $user = User::factory()→create();
    $invoice = Invoice::factory()→for(Customer::factory()→for($user→team))→overdue()→create();

    $this→actingAs($user)
        →post(route('invoices.pay', $invoice))
        →assertRedirect(route('invoices.show', $invoice));

    expect($invoice→refresh())
        →status→toBe(InvoiceStatus::Paid)
        →paid_at→not→toBeNull();
});

it('hides other teams\' invoices', function (): void {
    $invoice = Invoice::factory()→create();

    $this→actingAs(User::factory()→create())
        →get(route('invoices.show', $invoice))
        →assertNotFound();
});

arch('app code is strict and final')
    →expect('App')
    →toUseStrictTypes()
    →classes()→toBeFinal()→ignoring('App\\Providers');

arch('no debugging calls')
    →expect(['dd', 'dump', 'ray', 'var_dump'])
    →not→toBeUsed();
```

Testing each frontend stack

```
// Livewire
Livewire::actingAs($user)
    →test('invoice-table')
    →set('search', 'INV-001')
    →assertSee('INV-001')
    →call('markPaid', $invoice→id)
    →assertHasNoErrors();

// Inertia
$this→actingAs($user)
    →get(route('invoices.show', $invoice))
    →assertInertia(fn (AsserttableInertia $page) => $page
        →component('invoices/show')
        →where('invoice.id', $invoice→id)
        →where('can.update', true));
```

Git, commits and code review

Commits

- **MUST** LARAVEL-21.1 follow **Conventional Commits**: `type(scope): summary` in the imperative, lower case, no trailing period, under 72 characters.
- Types: `feat`, `fix`, `refactor`, `perf`, `test`, `docs`, `build`, `ci`, `chore`, `style`, `revert`. Scope is the domain or area: `feat(invoices): add overdue reminders`.
- **MUST** LARAVEL-21.2 mark breaking changes with `!` and a `BREAKING CHANGE:` footer.
- **SHOULD** LARAVEL-21.3 explain the *why* in the body when the change is not obvious from the summary.
- **MUST** LARAVEL-21.4 keep each commit building and passing tests; squash fix-up commits before merging.

Branches and pull requests

- **MUST** LARAVEL-21.5 branch from `main` with a short typed name: `feat/invoice-reminders`, `fix/overdue-scope`.
- **SHOULD** LARAVEL-21.6 keep pull requests under about 400 changed lines (excluding generated files and lock files); split larger work behind a Pennant flag.
- **MUST** LARAVEL-21.7 pass `composer lint`, `composer test` and the frontend checks (`npm run lint`, `npm run types`, `npm run test`) before review.
- **MUST** LARAVEL-21.8 include migrations, factories and tests in the same pull request as the feature they support.
- **MUST** LARAVEL-21.9 describe what changed, why, how it was tested, and any deploy steps (migrations, config, queue restarts).

Review checklist

- Does every model use attributes instead of properties, `HasUuids`, and traits once it grows?
- Are writes validated by a form request or Livewire form, and authorized by a policy?
- Is business logic in an action, with controllers and components thin?
- Are queries eager loaded, paginated and indexed?
- Are jobs idempotent with `#[Tries]` and `#[Timeout]` ?
- Are new UI pieces small components, accessible and translated?
- Are there tests for the happy path, the permission failure and the validation failure?

A

Appendix

A. `pint.json` and `.editorconfig`

```
{
  "preset": "laravel",
  "rules": {
    "declare_strict_types": true,
    "final_class": true,
    "strict_comparison": true,
    "fully_qualified_strict_types": true,
    "global_namespace_import": { "import_classes": true, "import_constants": true,
"import_functions": true },
    "ordered_class_elements": {
      "order": ["use_trait", "case", "constant", "property", "construct", "method_public",
"method_protected", "method_private"]
    },
    "ordered_imports": { "sort_algorithm": "alpha", "imports_order": ["class", "function", "const"]
  },
    "trailing_comma_in_multiline": { "elements": ["arrays", "arguments", "parameters", "match"] },
    "mb_str_functions": true
  }
}
```

```
root = true

[*]
charset = utf-8
end_of_line = lf
insert_final_newline = true
trim_trailing_whitespace = true
indent_style = space
indent_size = 2

[*.php]
indent_size = 4

[*.md]
trim_trailing_whitespace = false
```

B. phpstan.neon and rector.php

```
includes:
- vendor/larastan/larastan/extension.neon
- phpstan-baseline.neon

parameters:
  level: 8
  paths:
    - app
    - config
    - database
    - routes
  checkMissingCallableSignature: true
  treatPhpDocTypesAsCertain: false
```

```
<?php

declare(strict_types=1);

use Rector\Config\RectorConfig;
use RectorLaravel\Set\LaravelSetList;

return RectorConfig::configure()
    →withPaths([__DIR__.'/app', __DIR__.'/database', __DIR__.'/routes', __DIR__.'/tests'])
    →withPhpSets/php85: true)
    →withSets([LaravelSetList::LARAVEL_130])
    →withPreparedSets(deadCode: true, codeQuality: true, typeDeclarations: true, earlyReturn:
true);
```

C. New model checklist

- ☐ Migration with `uuid('id')→primary()`, `foreignUuid()` keys with explicit `onDelete`, and indexes for hot queries.
- ☐ `final` model with `HasUuids` and `HasFactory`.
- ☐ `#[Table]` and `#[Fillable]` declared; `#[Hidden]` for secrets; `#[Appends]` only for cheap accessors.
- ☐ `#[UseFactory]`, `#[UsePolicy]`, `#[ObservedBy]` (if any), `#[CollectedBy]` / `#[UseEloquentBuilder]` / `#[UseResource]` where used.
- ☐ `casts()` covers every enum, date, boolean, money and encrypted column.
- ☐ Scopes use `#[Scope]` on protected methods.
- ☐ Relationships in `Has<Model>Relationships` once there are more than two, each with a generic return type.
- ☐ Factory with `#[UseModel]` and named states; policy; resource; tests.

D. Composer scripts

```
{
  "scripts": {
    "dev": [
      "Composer\\Config::disableProcessTimeout",
      "npx concurrently -c \"#93c5fd,#c4b5fd,#fb7185,#fdb74\" \"php artisan serve\" \"php artisan queue:listen --tries=1\" \"php artisan pail --timeout=0\" \"npm run dev\" -- names=server,queue,logs,vite"
    ],
    "lint": [
      "pint --test",
      "phpstan analyse --memory-limit=2G",
      "rector --dry-run"
    ],
    "fix": [
      "rector",
      "pint"
    ],
    "test": [
      "@php artisan config:clear --ansi",
      "pest --parallel"
    ]
  }
}
```

E. Sources

- [Laravel 13 release notes](#)
- [Laravel 13 Eloquent documentation](#)
- [Laravel 13 PHP attributes overview \(Laravel Daily\)](#)
- [Everything new in Livewire 4 \(Laravel News\)](#)
- [Inertia 3.0 \(Laravel News\)](#)
- [Inertia v3 upgrade guide](#)