Laravel 11 Cheat Sheet

Color Legend: `Blue = Essential/Daily Commands` | `Red = Advanced/Specialized Commands` | `Green = New in Laravel 11`

Installation & Setup

`composer create-project laravel/laravel app-name`	Create new Laravel project
`composer global require laravel/installer`	Install Laravel installer globally
`laravel new app-name`	Create new Laravel project using installer
`laravel new app-name --git`	Create new project with git repo
`laravel new app-name --database=mysql`	Create project with specific database
`php artisan serve`	Start development server
`php artisan serve --host=0.0.0.0 --port=8080`	Start server with custom host/port
`php artisan --version`	Check Laravel version
`composer update`	Update Laravel dependencies
`composer install`	Install project dependencies
`composer install --no-dev --optimize-autoloader`	Production install
`npm install`	Install Node.js dependencies
`npm run dev`	Compile assets for development
`npm run build`	Compile assets for production
`npm run watch`	Watch and recompile assets

Artisan Commands

`php artisan list`	List all available commands
`php artisan help <command>`	Get help for specific command
`php artisan make:model <name>`	Create new model
`php artisan make:model <name> -m`	Create model with migration
`php artisan make:model <name> -mrc`	Create model, migration, resource controller
`php artisan make:model <name> -a`	Create model with all (migration, factory, seeder, policy, controller, form requests)
`php artisan make:controller <name>`	Create new controller

`php artisan make:controller <name> --resource`	Create resource controller
`php artisan make:controller <name> --api`	Create API resource controller
`php artisan make:controller <name> --invokable`	Create single action controller
`php artisan make:migration <name>`	Create new migration
`php artisan make:migration create_users_table`	Create migration with specific name
`php artisan make:migration add_column_to_table --table=users`	Add column to existing table
`php artisan make:seeder <name>`	Create new seeder
`php artisan make:seeder UserSeeder`	Create specific seeder
`php artisan make:factory <name>`	Create new factory
`php artisan make:factory UserFactory --model=User`	Create factory for specific model
`php artisan make:request <name>`	Create new form request
`php artisan make:request StoreUserRequest`	Create specific form request
`php artisan make:middleware <name>`	Create new middleware
`php artisan make:middleware CheckAge`	Create specific middleware
`php artisan make:policy <name>`	Create new policy
`php artisan make:policy UserPolicy --model=User`	Create policy for specific model
`php artisan make:event <name>`	Create new event
`php artisan make:listener <name>`	Create new listener
`php artisan make:listener SendWelcomeEmail --event=UserRegistered`	Create listener for specific event
`php artisan make:job <name>`	Create new job
`php artisan make:job ProcessPayment`	Create specific job
`php artisan make:mail <name>`	Create new mail class
`php artisan make:mail WelcomeEmail --markdown=emails.welcome`	Create mail with markdown template
`php artisan make:notification <name>`	Create new notification
`php artisan make:resource <name>`	Create new API resource
`php artisan make:resource UserResource`	Create specific API resource
`php artisan make:test <name>`	Create new test
`php artisan make:test UserTest --unit`	Create unit test
`php artisan make:test UserCanLoginTest --feature`	Create feature test
`php artisan make:command <name>`	Create new artisan command

`php artisan make:provider <name>`	Create new service provider
`php artisan make:rule <name>`	Create new validation rule
`php artisan make:cast <name>`	Create new custom cast
`php artisan make:component <name>`	Create new Blade component
`php artisan make:observer <name>`	Create new model observer

Routing

`Route::get('/uri', [Controller::class, 'method']);`	Basic GET route
`Route::post('/uri', [Controller::class, 'method']);`	POST route
`Route::put('/uri', [Controller::class, 'method']);`	PUT route
`Route::patch('/uri', [Controller::class, 'method']);`	PATCH route
`Route::delete('/uri', [Controller::class, 'method']);`	DELETE route
`Route::any('/uri', [Controller::class, 'method']);`	Route that responds to any HTTP verb
`Route::match(['get', 'post'], '/uri', [Controller::class, 'method']);`	Route responding to multiple verbs
`Route::resource('users', UserController::class);`	Resource route
`Route::apiResource('users', UserController::class);`	API resource route (no create/edit)
`Route::resource('users', UserController::class)->only(['index', 'show']);`	Partial resource routes
`Route::resource('users', UserController::class)->except(['destroy']);`	Resource routes except destroy
`Route::group(['prefix' => 'admin'], function () { ... });`	Route group with prefix
`Route::group(['middleware' => 'auth'], function () { ... });`	Route group with middleware
`Route::group(['namespace' => 'Admin'], function () { ... });`	Route group with namespace
`Route::middleware(['auth'])->group(function () { ... });`	Route group with middleware
`Route::name('profile')->get('/profile', ...);`	Named route
`route('profile')`	Generate URL for named route
`route('profile', ['id' => 1])`	Generate URL with parameters
`Route::redirect('/here', '/there');`	Redirect route
`Route::redirect('/here', '/there', 301);`	Permanent redirect route
`Route::view('/welcome', 'welcome');`	Return view directly

`Route::view('/welcome', 'welcome', ['name' => 'Taylor']);`	Return view with data
`Route::fallback(function () { ... });`	Fallback route
`Route::domain('{account}.example.com')->group(...);`	Subdomain routing
`Route::where('id', '[0-9]+')->get('/user/{id}', ...);`	Route parameter constraints
`Route::whereNumber('id')->get('/user/{id}', ...);`	Numeric parameter constraint
`Route::whereAlpha('name')->get('/user/{name}', ...);`	Alphabetic parameter constraint
`Route::whereUuid('id')->get('/user/{id}', ...);`	UUID parameter constraint
`php artisan route:list`	List all routes
`php artisan route:list --name=user`	List routes with specific name
`php artisan route:list --method=GET`	List routes with specific method
`php artisan route:cache`	Cache routes for performance
`php artisan route:clear`	Clear route cache

Database & Migrations

`php artisan migrate`	Run pending migrations
`php artisan migrate --force`	Force run migrations in production
`php artisan migrate --pretend`	Show SQL that would be executed
`php artisan migrate --step`	Run migrations one by one
`php artisan migrate:rollback`	Rollback last migration
`php artisan migrate:rollback --step=5`	Rollback specific number of migrations
`php artisan migrate:reset`	Reset all migrations
`php artisan migrate:refresh`	Reset and re-run all migrations
`php artisan migrate:refresh --seed`	Reset, re-run migrations and seed
`php artisan migrate:fresh`	Drop all tables and re-run migrations
`php artisan migrate:fresh --seed`	Drop all tables, re-run migrations and seed
`php artisan migrate:status`	Show migration status
`php artisan make:migration create_users_table`	Create migration
`php artisan make:migration add_email_to_users_table --table=users`	Add column migration
`php artisan make:migration create_users_table --create=users`	Create table migration
`php artisan db:seed`	Run database seeders
`php artisan db:seed --class=UserSeeder`	Run specific seeder
`php artisan db:seed --force`	Force run seeders in production
`php artisan db:wipe`	Drop all tables, views, and types
`php artisan db:show`	Display information about database
`php artisan db:table users`	Display information about table
`php artisan db:monitor`	Monitor database connections
`php artisan tinker`	Interactive PHP shell
`php artisan schema:dump`	Dump current database schema
`php artisan schema:dump --prune`	Dump schema and prune migration files

Eloquent ORM

`User::all()`	Get all records
`User::find($id)`	Find record by ID

`User::findOrFail($id)`	Find record by ID or throw exception
`User::first()`	Get first record
`User::firstOrFail()`	Get first record or throw exception
`User::latest()->get()`	Get records ordered by latest
`User::oldest()->get()`	Get records ordered by oldest
`User::where('name', 'John')->get()`	Query with where clause
`User::where('age', '>', 18)->get()`	Query with comparison operator
`User::whereIn('id', [1, 2, 3])->get()`	Query with whereIn
`User::whereBetween('age', [18, 65])->get()`	Query with whereBetween
`User::whereNull('email_verified_at')->get()`	Query with whereNull
`User::whereNotNull('email_verified_at')->get()`	Query with whereNotNull
`User::whereDate('created_at', '2023-01-01')->get()`	Query by date
`User::whereYear('created_at', 2023)->get()`	Query by year
`User::whereMonth('created_at', 1)->get()`	Query by month
`User::select('name', 'email')->get()`	Select specific columns
`User::distinct()->get()`	Get distinct records
`User::orderBy('name', 'asc')->get()`	Order results ascending
`User::orderBy('created_at', 'desc')->get()`	Order results descending
`User::take(10)->get()`	Limit results
`User::skip(10)->take(10)->get()`	Skip and take (pagination)
`User::paginate(15)`	Paginate results
`User::simplePaginate(15)`	Simple pagination
`User::count()`	Count records
`User::max('age')`	Get maximum value
`User::min('age')`	Get minimum value
`User::avg('age')`	Get average value
`User::sum('salary')`	Get sum of values
`User::create(['name' => 'John', 'email' => '...'])`	Create new record
`User::insert([['name' => 'John'], ['name' => 'Jane']])`	Insert multiple records
`User::updateOrCreate(['email' => '...'], ['name' => 'John'])`	Update or create record

`User::firstOrCreate(['email' => '...'], ['name' => 'John'])`	Find or create record
`$user->update(['name' => 'Jane'])`	Update record
`User::where('active', false)->update(['active' => true])`	Update multiple records
`$user->delete()`	Delete record
`User::destroy([1, 2, 3])`	Delete multiple records by ID
`User::where('active', false)->delete()`	Delete multiple records by query
`User::with('posts')->get()`	Eager loading
`User::with(['posts', 'comments'])->get()`	Multiple eager loading
`User::with('posts:id,title,user_id')->get()`	Eager loading specific columns
`User::withCount('posts')->get()`	Eager loading with count

Blade Templates

`{{ $variable }}`	Echo variable (escaped)
`{!! $variable !!}`	Echo variable (unescaped)
`@if($condition) ... @endif`	Conditional statement
`@foreach($items as $item) ... @endforeach`	Loop through items
`@extends('layout.app')`	Extend layout
`@section('content') ... @endsection`	Define section
`@yield('content')`	Yield section content
`@include('partials.header')`	Include partial view
`@auth ... @endauth`	Check if user is authenticated
`@guest ... @endguest`	Check if user is guest
`@csrf`	CSRF token field
`@method('PUT')`	Method spoofing field

Authentication

`php artisan make:auth`	Scaffold authentication views
`php artisan ui:auth`	Generate authentication scaffolding
`Auth::check()`	Check if user is authenticated
`Auth::user()`	Get authenticated user
`Auth::login($user)`	Log in user
`Auth::logout()`	Log out user
`auth()->user()`	Helper for authenticated user
`auth()->check()`	Helper to check authentication
`@auth ... @endauth`	Blade directive for auth check
`Route::middleware('auth')->group(...)`	Protect routes with auth

Middleware

`php artisan make:middleware CheckAge`	Create middleware
`Route::middleware('auth')->get(...)`	Apply middleware to route
`Route::middleware(['auth', 'admin'])->get(...)`	Multiple middleware
`protected $middleware = [...] in Kernel.php`	Global middleware
`protected $middlewareGroups = [...] in Kernel.php`	Middleware groups
`protected $routeMiddleware = [...] in Kernel.php`	Route middleware
`$request->user()`	Access user in middleware
`return $next($request)`	Pass request to next middleware
`abort(403)`	Deny access in middleware

Validation Rules

`$request->validate(['email' => 'required|email'])`	Basic validation
`'required|string|max:255'`	Common string validation
`'required|email|unique:users'`	Email validation with unique
`'nullable|integer|min:1'`	Optional integer validation
`'required|array|min:1'`	Array validation
`'required|file|mimes:jpg,png|max:2048'`	File validation
`'required|date|after:today'`	Date validation

`'required|confirmed'`	Password confirmation
`'sometimes|nullable|string'`	Conditional validation
`Rule::exists('users', 'id')`	Database validation rule
`Rule::unique('users')->ignore($user->id)`	Unique with ignore
`'required|regex:/^[A-Za-z]+$/'`	Regex validation
`php artisan make:rule Uppercase`	Custom validation rule

API Development

`php artisan make:resource UserResource`	Create API resource
`php artisan make:resource UserCollection`	Create API collection
`return new UserResource($user)`	Return single resource
`return UserResource::collection($users)`	Return resource collection
`php artisan install:api`	Install Laravel Sanctum
`$user->createToken('token-name')`	Create API token
`Route::middleware('auth:sanctum')->get(...)`	Protect API route
`return response()->json($data)`	Return JSON response
`return response()->json($data, 201)`	Return JSON with status
`abort_if($condition, 403)`	Conditional abort
`$request->expectsJson()`	Check if request expects JSON

Queues & Jobs

`php artisan queue:work`	Start processing jobs
`php artisan queue:listen`	Listen for new jobs
`php artisan queue:restart`	Restart queue workers
`php artisan queue:failed`	List failed jobs
`php artisan queue:retry all`	Retry all failed jobs
`php artisan queue:retry 5`	Retry specific failed job
`php artisan queue:flush`	Delete all failed jobs
`php artisan queue:clear`	Delete all jobs from queue
`php artisan make:job ProcessPayment`	Create new job
`php artisan horizon`	Start Laravel Horizon dashboard

Cache Management

`Cache::put('key', 'value', 3600)`	Store cache item
`Cache::get('key')`	Get cache item
`Cache::remember('key', 3600, fn() => expensive_operation())`	Cache with fallback
`Cache::forget('key')`	Remove cache item
`Cache::flush()`	Clear all cache
`php artisan cache:clear`	Clear application cache
`php artisan cache:forget key`	Forget specific cache key
`Cache::tags(['people', 'artists'])->put('John', $john, 60)`	Tagged cache
`Cache::lock('order-processing')->get(function () {...})`	Cache locks

Storage & Files

`Storage::disk('public')->put('file.txt', $contents)`	Store file
`Storage::get('file.txt')`	Get file contents
`Storage::download('file.txt')`	Download file
`Storage::delete('file.txt')`	Delete file
`Storage::exists('file.txt')`	Check if file exists

`$request->file('upload')->store('uploads')`	Store uploaded file
`php artisan storage:link`	Create storage symlink
`Storage::url('file.txt')`	Get file URL
`Storage::size('file.txt')`	Get file size
`Storage::lastModified('file.txt')`	Get last modified time

Laravel Sail (Docker)

`./vendor/bin/sail up`	Start all services
`./vendor/bin/sail up -d`	Start services in background
`./vendor/bin/sail down`	Stop all services
`./vendor/bin/sail artisan migrate`	Run migrations in container
`./vendor/bin/sail composer install`	Install dependencies in container
`./vendor/bin/sail npm run dev`	Run npm in container
`./vendor/bin/sail test`	Run tests in container
`./vendor/bin/sail shell`	Access container shell
`./vendor/bin/sail mysql`	Access MySQL in container
`./vendor/bin/sail redis`	Access Redis in container

Testing

`php artisan make:test UserTest`	Create test class
`php artisan test`	Run all tests
`php artisan test --filter=UserTest`	Run specific test
`$this->assertEquals($expected, $actual)`	Assert equality
`$this->assertTrue($condition)`	Assert true
`$this->assertDatabaseHas('users', [...])`	Assert database record exists
`$this->get('/users')`	Make GET request in test
`$this->post('/users', $data)`	Make POST request in test
`$this->actingAs($user)`	Authenticate user in test
`$this->assertRedirect('/dashboard')`	Assert redirect

Inertia.js Integration

`composer require inertiajs/inertia-laravel`	Install Inertia.js Laravel adapter
`php artisan inertia:middleware`	Create Inertia middleware
`npm install @inertiajs/vue3`	Install Inertia Vue 3 adapter
`npm install @inertiajs/react`	Install Inertia React adapter
`Inertia::render('Users/Index', ['users' => $users])`	Render Inertia page with data
`return inertia('Users/Show', compact('user'))`	Return Inertia response (helper)
`Inertia::location('/dashboard')`	Redirect with Inertia
`$request->header('X-Inertia')`	Check if request is from Inertia
`Inertia::share('auth.user', fn() => auth()->user())`	Share data globally
`Inertia::version(fn() => md5_file(public_path('mix-manifest.json')))`	Asset versioning
`<Head title='Page Title' />`	Set page title (Vue/React)
`$page.props.user`	Access shared props (Vue/React)
`import { Link } from '@inertiajs/vue3'`	Inertia Link component (Vue)
`import { router } from '@inertiajs/vue3'`	Inertia router (Vue)
`router.visit('/users')`	Programmatic navigation
`router.post('/users', form)`	POST request with Inertia
`$page.props.errors`	Access validation errors

Livewire Components

`composer require livewire/livewire`	Install Livewire
`php artisan make:livewire Counter`	Create Livewire component
`php artisan make:livewire Users/Index`	Create nested Livewire component
`<livewire:counter />`	Render Livewire component
`@livewire('counter')`	Render with Blade directive
`public $count = 0;`	Define public property
`public function increment() { $this->count++; }`	Define action method
`wire:click='increment'`	Wire click event
`wire:model='name'`	Two-way data binding
`wire:submit.prevent='save'`	Wire form submission

`$this->validate(['name' => 'required']);`	Validate in Livewire
`$this->emit('userSaved');`	Emit event
`protected $listeners = ['userSaved' => 'refreshUsers'];`	Listen to events
`wire:loading`	Show loading state
`wire:offline`	Show offline state
`$this->skipRender();`	Skip component re-render

Deployment & Optimization

`php artisan config:cache`	Cache configuration
`php artisan route:cache`	Cache routes
`php artisan view:cache`	Cache views
`php artisan config:clear`	Clear config cache
`php artisan route:clear`	Clear route cache
`php artisan view:clear`	Clear view cache
`php artisan cache:clear`	Clear application cache
`composer install --optimize-autoloader --no-dev`	Optimize for production
`php artisan migrate --force`	Run migrations in production
`npm run build`	Build assets for production

■ **Laravel 11 Tips & Best Practices:**

• Use Eloquent relationships and eager loading for performance
• Always validate user input using Form Requests
• Leverage middleware for cross-cutting concerns (auth, CORS, etc.)
• Use Laravel Sanctum for API authentication
• Implement proper error handling with custom exception classes
• Use database seeders and factories for testing data
• Write comprehensive tests (Feature + Unit tests)
• Use Laravel's built-in caching mechanisms (Redis recommended)
• Follow PSR standards and use Laravel Pint for code formatting
• Use environment variables for all configuration
• Implement proper database indexing and query optimization
• Use Laravel's queue system for background jobs
• Use Laravel Horizon for queue monitoring in production
• Leverage Laravel Sail for consistent development environments

■ **Laravel 11 New Features:**
• Improved artisan commands with better UX
• Enhanced API resource handling
• Better Docker integration with Sail
• Improved testing capabilities
• Enhanced security features

■ **Essential Packages for Laravel 11:**
• Laravel Debugbar (barryvdh/laravel-debugbar)
• Laravel IDE Helper (barryvdh/laravel-ide-helper)
• Laravel Telescope (laravel/telescope)
• Laravel Horizon (laravel/horizon)
• Laravel Sanctum (built-in API authentication)
• Laravel Pint (built-in code formatting)
• Spatie Laravel packages (permissions, media, etc.)
• Laravel Livewire for reactive components
• Inertia.js for modern SPA development