RÉFÉRENCE RAPIDE LARAVEL

Artisan, routage, Eloquent, Blade, middleware, auth

Artisan

Commandes courantes

`php artisan serve`	Démarrer le serveur de développemen
`php artisan make:model Name -m`	Créer un modèle avec une migration
`php artisan make:controller NameController`	Créer une classe contrôleur
`php artisan make:middleware Name`	Créer une classe middleware
`php artisan migrate`	Exécuter les migrations en attente
`php artisan migrate:rollback`	Annuler le dernier lot de migrations
`php artisan db:seed`	Exécuter les seeders de base de données
`php artisan tinker`	REPL interactif pour l'application
`php artisan route:list`	Lister toutes les routes enregistrées
`php artisan cache:clear`	Vider le cache de l'application
`php artisan config:clear`	Vider la configuration en cache
`php artisan queue:work`	Démarrer le traitement des tâches en file

Routage

Routes de base

```
Route::get('/users', [UserController::class, 'index']);
Route::post('/users', [UserController::class, 'store']);
Route::put('/users/{id}', [UserController::class, 'update']);
Route::delete('/users/{id}', [UserController::class, 'destroy']);
```

Paramètres et groupes de routes

```
Route::get('/user/{id}', function (int $id) {
    return User::findOrFail($id);
});

Route::prefix('api')->middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'show']);
});
```

Fonctionnalités de routage

`->name('route.name')`	Route nommée pour la génération d'URL
`->where('id', '[0-9]+')`	Contrainte regex sur un paramètre
`Route::resource()`	Routes de ressource RESTful (7 routes)
`Route::apiResource()`	Ressource API (sans vues create/edit)
`Route::fallback()`	Capture tout pour les routes non trouvées

Contrôleurs

Contrôleur de ressource

```
class PostController extends Controller {
    public function index() {
        return view('posts.index', ['posts' => Post::all()]);
    }

    public function store(Request $request) {
        $validated = $request->validate(['title' => 'required|
max:255']);
        Post::create($validated);
        return redirect()->route('posts.index');
    }
}
```

Méthodes de ressource

`index()`	GET /resource -- lister tout
`create()`	GET /resource/create -- afficher le formulaire
`store()`	POST /resource -- enregistrer
`show($id)`	GET /resource/{id} -- afficher un
`edit($id)`	GET /resource/{id}/edit -- formulaire d'édition
`update($id)`	PUT /resource/{id} -- mettre à jour
`destroy($id)`	DELETE /resource/{id} -- supprimer

Templates Blade

Mise en page et sections

```
{{-- layouts/app.blade.php --}}
<html><body>
  @yield('content')
</body></html>

{{-- pages/home.blade.php --}}
@extends('layouts.app')
@section('content')
  <h1>Home</h1>
@endsection
```

Directives

`{{ $var }}`	Afficher avec échappement HTML
`{!! $html !!}`	Afficher brut (non échappé)
`@if / @elseif / @else`	Blocs conditionnels
`@foreach ($items as $item)`	Itérer sur une collection
`@forelse / @empty`	Boucle avec repli si vide
`@include('partial')`	Inclure une autre vue Blade
`@component / @slot`	Composants Blade réutilisables
`@csrf`	Champ caché du token CSRF
`@auth / @guest`	Vérifier le statut d'authentification
`@error('field')`	Afficher une erreur de validation

Eloquent ORM

Bases du modèle

```
class Post extends Model {
    protected $fillable = ['title', 'body', 'user_id'];

    public function user() {
        return $this->belongsTo(User::class);
    }
}
```

Requêtes

```
Post::all();                            // tous les enregistrements
Post::find(1);                          // par clé primaire
Post::where('status', 'published')->get();
Post::where('views', '>', 100)->orderBy('created_at', 'desc')-
>first();
```

Opérations CRUD

```
$post = Post::create(['title' => 'New', 'body' => '...']);
$post->update(['title' => 'Updated']);
$post->delete();
Post::destroy([1, 2, 3]);   // supprimer par IDs
```

Relations

`hasOne`	Un-à-un (User -> Phone)
`hasMany`	Un-à-plusieurs (Post -> Comments)
`belongsTo`	Inverse de hasOne/hasMany
`belongsToMany`	Plusieurs-à-plusieurs avec table pivot
`hasManyThrough`	Has-many via un modèle intermédiaire

Migrations

Créer des tables

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained()-
>cascadeOnDelete();
    $table->string('title');
    $table->text('body')->nullable();
    $table->timestamps();
});
```

Types de colonnes

`$table->id()`	Clé primaire BIGINT auto-incrémentée
`$table->string('col', 100)`	VARCHAR avec longueur optionnelle
`$table->text('col')`	Colonne TEXT
`$table->integer('col')`	Colonne INTEGER
`$table->boolean('col')`	Colonne BOOLEAN
`$table->json('col')`	Colonne JSON
`$table->timestamp('col')`	Colonne TIMESTAMP
`$table->timestamps()`	created_at et updated_at
`$table->softDeletes()`	deleted_at pour les suppressions douces

Middleware

Middleware personnalisé

```
class EnsureAdmin {
    public function handle(Request $request, Closure $next) {
        if (! $request->user()?->is_admin) {
            abort(403);
        }
        return $next($request);
    }
}
```

Enregistrement et utilisation

```
// bootstrap/app.php
->withMiddleware(function (Middleware $middleware) {
    $middleware->alias(['admin' => EnsureAdmin::class]);
})

// Dans les routes
Route::get('/admin', fn() => '...')->middleware('admin');
```

Middleware intégrés

`auth`	Exiger l'authentification
`guest`	Rediriger si authentifié
`throttle:60,1`	Limitation de débit (60 req/min)
`verified`	Exiger la vérification email
`signed`	Valider l'URL signée

Authentification

Helpers Auth

```
Auth::check();           // l'utilisateur est-il connecté ?
Auth::user();            // modèle User courant
Auth::id();              // ID de l'utilisateur courant
Auth::attempt(['email' => $e, 'password' => $p]);
Auth::logout();
```

Kits de démarrage

`Laravel Breeze`	Scaffold d'auth minimal (Blade ou Inertia)
`Laravel Jetstream`	Complet (équipes, 2FA, tokens API)
`Sanctum`	Authentification par token SPA / mobile
`Passport`	Implémentation complète de serveur OAuth2

Protéger les routes

```
Route::middleware('auth')->group(function () {
    Route::get('/dashboard', [DashController::class, 'index']);
});
```

Validation

Validation dans le contrôleur

```
$validated = $request->validate([
    'title' => 'required|string|max:255',
    'email' => 'required|email|unique:users',
    'age'   => 'nullable|integer|min:0',
]);
```

Form Request

```
class StorePostRequest extends FormRequest {
    public function rules(): array {
        return [
            'title' => 'required|max:255',
            'body'  => 'required|min:10',
        ];
    }
}
```

Règles courantes

`required`	Le champ doit être présent et non vide
`string | integer | boolean`	Validation de type
`min:N | max:N`	Longueur ou valeur min/max
`email`	Format email valide
`unique:table,column`	Doit être unique dans la table DB
`exists:table,column`	Doit exister dans la table DB
`in:a,b,c`	Doit être l'une des valeurs listées
`confirmed`	Nécessite un champ _confirmation correspondant
`date | after:date`	Validation de date

Modèles courants

Réponse API

```
return response()->json(['data' => $users], 200);
return response()->json(['error' => 'Not found'], 404);
```

Environnement et configuration

```
env('APP_KEY');              // lire une valeur .env
config('app.name');          // lire une valeur de config
config(['app.debug' => true]); // définir à l'exécution
```

Helpers utiles

`route('name', $params)`	Générer l'URL pour une route nommée
`redirect()->route('name')`	Rediriger vers une route nommée
`back()->withErrors()`	Rediriger en arrière avec les erreurs de validation
`abort(404)`	Lancer une exception HTTP
`collect($array)`	Créer une Collection depuis un tableau
`now()`	Datetime Carbon actuelle
`cache()->remember()`	Mettre en cache une valeur avec TTL