SCIĄGA LARAVEL · API

Odporne REST API

dominik-dev.pl
pełny artykuł →

Kompatybilność, triada Request/Action/Resource, abilities, paginacja, błędy, limity

TL;DR

API to **kontrakt na granicy aplikacji**: stabilne nazwy, przewidywalna paginacja, użyteczne błędy, bezpieczne ponowienia. Laravel daje elementy — jakość produkcyjna wynika ze świadomej polityki kompatybilności, autoryzacji i awarii.

1. Polityka kompatybilności najpierw

```
Route::prefix('v1')->middleware('auth:sanctum')->group(function (): void {
    Route::apiResource('products', ProductController::class);
});
```

Wersjonuj **publiczne** API, gdy niezależnie wdrażani klienci potrzebują jasnej obietnicy. To **nie** rytuał dla prywatnego backendu Inertia wdrażanego z serwerem. Po publikacji v1 nie zmieniaj po cichu znaczenia pola — wystaw v2 obok, określ okres migracji. Route model binding + policy rozstrzygają dostęp; nie duplikuj kontroli własności w kontrolerach.

2. Rozdziel wejście / przypadek użycia / reprezentację

```
// Resource = jedyne miejsce publicznej reprezentacji
final class ProductResource extends JsonResource {
    /** @return array<string, int|string> */
    public function toArray(Request $request): array {
        return [
            'id' => $this->resource->getKey(),
            'name' => $this->resource->name,
            'price_cents' => $this->resource->price_cents,
        ];
    }
}
// zwracaj ProductResource::collection($products) — nie kolekcję Eloquent
```

Form Request waliduje wejście, akcja wykonuje przypadek użycia, Resource posiada reprezentację. Przypadkowa kolumna modelu nie staje się częścią API na zawsze.

3. Wąskie abilities tokenów

```
$token = $user->createToken('warehouse-sync', ['products:read']);

Route::get('/v1/products', ProductController::class)
    ->middleware(['auth:sanctum', 'abilities:products:read']);
```

Tokeny Sanctum to **poświadczenia, nie role**. Policy = dostęp użytkownika, abilities = zakres konkretnego tokenu. Token raportowania nie powinien usuwać produktów. Rotuj, unieważniaj, zapisuj cel; nigdy długowieczny token w JS przeglądarki.

4. Paginacja dobrana do zapytania

	Offset `paginate()`	**Cursor** `cursorPaginate()`
Daje	total, numery stron	szuka po stabilnym, zindeksowanym porządku
Kiedy	człowiek faktycznie potrzebuje strony 42	feedy, duże tabele dopisywane w czasie
Ryzyko	inserty przesuwają elementy; duży offset kosztowny	kolejność musi być deterministyczna (np. `created_at, id`)

```
Product::query()->orderBy('id')->cursorPaginate(perPage: 50);   // nie paginuj po wartości obliczanej
```

5. Błędy czytelne dla maszyny (RFC 9457)

```
// application/problem+json — jeden rozpoznawalny kształt
{"type":"https://api.example.com/problems/validation",
 "title":"The request is invalid.","status":422,
 "errors":{"name":["The name field is required."]}}
```

Stabilne `type`, krótkie `title`, `status`, `instance`/ID korelacyjne; błędy pól **tylko** przy walidacji. Mapowanie centralizuj w konfiguracji wyjątków — kontrolery zwracają zasoby sukcesu, nie różne tablice błędów. **Nigdy** wiadomości wyjątku, SQL ani stack trace do klienta.

6. Limituj chronioną możliwość

```
RateLimiter::for('partner-api', fn (Request $r): Limit =>
    Limit::perMinute(120)->by((string) $r->user()?->getAuthIdentifier()));
// trasy: ->middleware('throttle:partner-api')
```

Nazwane limitery w providerze (klucz i polityka widoczne w review). Dla ruchu anonimowego — zaufana tożsamość, **nie** niezaufany forwarded-IP. Zwracaj `429` z informacją o ponowieniu; alarmuj o stałym limitowaniu partnera.

Pułapki / kiedy NIE

- Wersjonowany REST tylko „bo brzmi nowocześnie" — wewnętrzny formularz nie potrzebuje publicznej obietnicy.
- Kursor przy dowolnym sortowaniu bez indeksu i tie-breakera.
- Abilities Sanctum jako pełny model uprawnień.
- **HTTP retry ≠ exactly-once** — płatności i webhooki potrzebują idempotency key + unikalnego ograniczenia bazy.
- Przed publikacją: feature test każdego elementu kontraktu — sukces, oczekiwany błąd, autoryzacja, paginacja, ponowienie.