

API pentru un catalog de servicii

Fișă publică de proiect · exerciții și criterii de acceptare

Definești contractul înainte de cod: listare, paginare, creare și erori. Protejezi accesul și verifici atât răspunsul, cât și efectul în baza de date.

Livrabile

- Contract API
- Resources
- Validare
- Autentificare
- Teste HTTP

Criterii de acceptare

- Câmpurile private nu apar
- Sortarea are listă permisă
- Cererea invalidă nu modifică date

Contract API stabil pentru client

Un API bun definește resursele, statusurile, erorile și paginarea. Clientul nu trebuie să ghicească dacă o operație a reușit dintr-un mesaj în limba română. Pentru laborator, resursa cerere are id, serviciu și status, iar câmpurile interne nu sunt expuse. Metoda HTTP și răspunsul trebuie să aibă sens consecvent. Autentificarea și autorizarea se verifică pentru fiecare endpoint care cere acces, nu doar la login.

Exemplu de lucru

GET /api/v1/requests → listă paginată
POST /api/v1/requests → 201 + resursa creată
GET /api/v1/requests/42 → 200 sau eroare conform accesului
PATCH /api/v1/requests/42 → schimbare permisă
Exemplu: {"data":{"id":42,"service":"revizie","status":"new"}}

Interpretarea rezultatului

Un răspuns 200 cu textul „nu ai acces” contrazice contractul de status dacă aplicația a stabilit 403 pentru refuz. Păstrează documentația și testele sincronizate. Nu expune adrese sau note interne doar pentru că modelul le serializează implicit.

Exercițiu de consolidare

Definește un răspuns de eroare pentru status invalid și unul pentru resursă inaccesibilă, fără a divulga date suplimentare.

1. Ce înseamnă un API bun

Definește un răspuns de eroare pentru status invalid și unul pentru resursă inaccesibilă, fără a divulga date suplimentare.

Verificare orientativă

Un răspuns 200 cu textul „nu ai acces” contrazice contractul de status dacă aplicația a stabilit 403 pentru refuz. Păstrează documentația și testele sincronizate. Nu expune adrese sau note interne doar pentru că modelul le serializează implicit.

De evitat

Un contract stabil este mai util decât endpoint-uri construite ad-hoc.

API Resource cu listă explicită de câmpuri

Resource-ul controlează reprezentarea publică a modelului. Nu întoarce automat toate coloanele, fiindcă schema poate primi ulterior câmpuri sensibile. Păstrează relațiile condiționate de încărcare pentru a evita interogări ascunse. Controllerul autorizează și pregătește datele, iar resource-ul le transformă. Forma răspunsului trebuie verificată printr-un test care caută atât câmpurile prezente, cât și pe cele interzise.

Exemplu de lucru

```
public function toArray(Request $request): array {
    return [
        'id' => $this->id,
        'service' => $this->service,
        'status' => $this->status,
        'created_at' => $this->created_at->toISOString(),
    ];
}
Controller: return new ServiceRequestResource($serviceRequest);
Nu include: note interne, tokenuri, email fără necesitate.
```

Interpretarea rezultatului

Adăugarea unei coloane în DB nu trebuie să schimbe accidental contractul API. Testează resource-ul cu un model care are note interne și confirmă că nu apar. Null pentru o dată absentă este o decizie de contract care trebuie păstrată consecvent.

Exercițiu de consolidare

Adaugă un câmp public nou într-o versiune compatibilă și descrie ce client ar putea fi afectat dacă schimbă tipul unui câmp existent.

2. Rute, controllere, API Resources

Adaugă un câmp public nou într-o versiune compatibilă și descrie ce client ar putea fi afectat dacă schimbă tipul unui câmp existent.

Verificare orientativă

Adăugarea unei coloane în DB nu trebuie să schimbe accidental contractul API. Testează resource-ul cu un model care are note interne și confirmă că nu apar. Null pentru o dată absentă este o decizie de contract care trebuie păstrată consecvent.

De evitat

Serializarea întregului model poate divulga coloane noi.

Filtre și sortare cu allowlist

Numele coloanei de sortare nu trebuie luat liber din cerere. Definește o listă de opțiuni publice și mapează-le la coloane cunoscute. Limitează per_page pentru a preveni răspunsuri excesive. Filtrul după status trebuie validat și aplicat înaintea paginării. Păstrează query string în linkurile de paginare dacă acesta face parte din experiența clientului. Nu filtra în PHP după ce ai paginat, deoarece totalurile ar deveni incoerente.

Exemplu de lucru

```
$data = $request->validate([
    'status' => ['nullable', 'in:new,approved,cancelled'],
    'sort' => ['nullable', 'in:newest,oldest'],
    'per_page' => ['nullable', 'integer', 'min:1', 'max:100'],
]);
$query = $request->user()->serviceRequests();
if (!empty($data['status'])) $query->where('status', $data['status']);
$direction = ($data['sort'] ?? 'newest') === 'oldest' ? 'asc' : 'desc';
return $query->orderBy('id', $direction)->paginate($data['per_page'] ?? 20)->withQueryString();
```

Interpretarea rezultatului

Sortarea după id este o alegere simplă și stabilă pentru laborator. Dacă sortezi după un câmp neunic, adaugă un criteriu secundar pentru ordine deterministă. Testează per_page=100000 și sort=coloana_interna; ambele trebuie respinse de contract.

Exercițiu de consolidare

Adaugă un filtru după serviciu și testează împreună filtrarea, totalul și linkul către pagina a doua.

3. Paginare, filtrare, sortare

Adaugă un filtru după serviciu și testează împreună filtrarea, totalul și linkul către pagina a doua.

Verificare orientativă

Sortarea după id este o alegere simplă și stabilă pentru laborator. Dacă sortezi după un câmp neunic, adaugă un criteriu secundar pentru ordine deterministă. Testează per_page=100000 și sort=coloana_interna; ambele trebuie respinse de contract.

De evitat

Numele de coloane nu se validează prin simplă parametrizare SQL.

Erori consistente fără detalii interne

Clientul are nevoie de un cod stabil și un mesaj util, nu de stack trace sau SQL. Separă erorile de validare, acces și conflict de stare. Logul intern poate păstra identificatorul operației pentru investigare, dar nu trebuie să expună secrete. Nu transforma toate excepțiile în 200, fiindcă monitorizarea și clienții nu mai pot interpreta succesul. Contractul exact poate varia, însă trebuie documentat și testat.

Exemplu de lucru

```
Validare: 422, code=validation_failed, errors pe câmpuri.
Acces: 403, code=forbidden.
Conflict: 409, code=state_conflict.
Eroare internă: 500, code=internal_error, request_id.
Exemplu: {"error":{"code":"state_conflict","message":"Cererea nu mai poate fi aprobată."}}
```

Interpretarea rezultatului

Nu include numele tabelelor, căi locale sau valori de token în răspunsul public. Un request_id ajută suportul să găsească logul fără a copia tot stack trace-ul în interfață. Testează și că APP_DEBUG nu schimbă necontrolat expunerea în producție.

Exercițiu de consolidare

Scrie două teste care verifică forma erorii și absența unui câmp sensibil din răspuns.

4. Erori consistente

Scrie două teste care verifică forma erorii și absența unui câmp sensibil din răspuns.

Verificare orientativă

Nu include numele tabelelor, căi locale sau valori de token în răspunsul public. Un request_id ajută suportul să găsească logul fără a copia tot stack trace-ul în interfață. Testează și că APP_DEBUG nu schimbă necontrolat expunerea în producție.

De evitat

Mesajele interne de excepție nu trebuie trimise public.

Sanctum după tipul clientului

Sanctum poate fi folosit pentru autentificare cu sesiune în aplicații first-party sau pentru tokenuri API, iar configurarea diferă. Nu prezenta un token ca soluție universală pentru orice frontend. Pentru un client de laborator cu token, păstrează abilități restrânse și verifică și policy pe resursă. Abilitatea „requests:read” nu înseamnă că utilizatorul poate citi cererile altora. Tokenul se arată doar când este emis și se păstrează secret.

Exemplu de lucru

```
Rută: Route::middleware('auth:sanctum')->get('/requests', ...);
Token demonstrativ: $user->createToken('laborator', ['requests:read'])->plainTextToken;
Client: Authorization: Bearer <token-propriu>
Teste: fără token; token invalid; token valid; resursă a altui user.
Revocare: șterge tokenul potrivit din evidența utilizatorului.
```

Interpretarea rezultatului

Nu pune tokenul în URL sau repository. Pentru SPA first-party, verifică fluxul de cookie, CSRF și domenii stateful al versiunii folosite. Fragmentul de token nu configurează singur toate aceste componente și nu trebuie amestecat cu fluxul de sesiune fără justificare.

Exercițiu de consolidare

Definește un token numai pentru citire și testează că endpointul de modificare nu îl acceptă doar fiindcă este autentificat.

5. Autentificare cu Sanctum

Definește un token numai pentru citire și testează că endpointul de modificare nu îl acceptă doar fiindcă este autentificat.

Verificare orientativă

Nu pune tokenul în URL sau repository. Pentru SPA first-party, verifică fluxul de cookie, CSRF și domenii stateful al versiunii folosite. Fragmentul de token nu configurează singur toate aceste componente și nu trebuie amestecat cu fluxul de sesiune fără justificare.

De evitat

Tokenurile nu se pun în exemple publice sau loguri.

Limitare și versionare cu un motiv concret

Rate limiting protejează resursele și stabilește un contract de utilizare; pragurile trebuie alese după cost și trafic, nu copiate ca adevăr universal. Cheia poate depinde de utilizator și, pentru vizitatori, de o politică potrivită aplicației. Versionarea se introduce când contractul se schimbă incompatibil, nu pentru fiecare patch intern. Un client trebuie să poată interpreta răspunsul de limită și să nu repete agresiv aceeași cerere.

Exemplu de lucru

Exemplu didactic:
`RateLimiter::for('requests-api', fn (Request $r) =>
 Limit::perMinute(30)->by($r->user()->id ?? $r->ip())
);
Route::middleware(['auth:sanctum', 'throttle:requests-api'])->prefix('v1')->group(...);
Pragul 30/minut este ilustrativ și se calibrează pentru produs.`

Interpretarea rezultatului

O limită pe IP poate afecta mai mulți utilizatori din aceeași rețea și nu oprește toate formele de abuz. Testează răspunsul 429 și comportamentul clientului. Pentru versiuni, păstrează v1 disponibilă conform politicii de migrare declarate, fără promisiuni indefinite.

Exercițiu de consolidare

Describe o schimbare compatibilă și una incompatibilă a resursei JSON, apoi decide care cere versiune nouă.

6. Rate limiting și versionare

Describe o schimbare compatibilă și una incompatibilă a resursei JSON, apoi decide care cere versiune nouă.

Verificare orientativă

O limită pe IP poate afecta mai mulți utilizatori din aceeași rețea și nu oprește toate formele de abuz. Testează răspunsul 429 și comportamentul clientului. Pentru versiuni, păstrează v1 disponibilă conform politicii de migrare declarate, fără promisiuni indefinite.

De evitat

Schimbarea contractului fără versiune rupe clienții existenți.

Teste API pentru succes, validare și acces

O suită utilă nu se limitează la `assertStatus(200)`. Verifică forma JSON, persistența și faptul că un utilizator nu poate accesa resursa altuia. Folosește baze de test izolate și date create de test, nu identificatori existenți pe server. Pentru endpointuri cu efect extern, înlocuiește furnizorul cu un fake controlat. Un test care trece numai pe baza ta locală cu date vechi nu este reproductibil.

Exemplu de lucru

```
$owner = User::factory()->create();
$other = User::factory()->create();
$record = ServiceRequest::factory()->for($owner)->create();
$this->actingAs($owner)->getJson('/api/v1/requests/'.$record->id)
    ->assertOk()->assertJsonPath('data.id', $record->id);
$this->actingAs($other)->getJson('/api/v1/requests/'.$record->id)->assertForbidden();
$this->assertDatabaseHas('service_requests', ['id' => $record->id]);
```

Interpretarea rezultatului

Factory-ul și ruta trebuie definite în proiectul de laborator. Dacă politica aleasă ascunde existența prin 404, testul trebuie să reflecte acel contract, nu să forțeze 403. Păstrează consecvența între listă, detaliu și operația de modificare.

Exercițiu de consolidare

Adaugă un test care trimite user_id al altcuiva la creare și verifică proprietarul salvat din identitatea autenticată.

7. Testarea endpoint-urilor

Adaugă un test care trimite user_id al altcuiva la creare și verifică proprietarul salvat din identitatea autenticată.

Verificare orientativă

Factory-ul și ruta trebuie definite în proiectul de laborator. Dacă politica aleasă ascunde existența prin 404, testul trebuie să reflecte acel contract, nu să forțeze 403. Păstrează consecvența între listă, detaliu și operația de modificare.

De evitat

Un test care verifică doar 200 ratează erori de conținut.

Atelier final: Contract API pentru listare și creare

Timp de lucru estimat: 40 de minute. Folosește datele de laborator de mai jos și salvează rezultatul separat de orice proiect real.

Date de pornire

API demonstrativ: GET /api/services cu paginare; POST /api/services pentru un rol autorizat. Intrări de test: title valid, title gol, sort necunoscut, utilizator fără drepturi.

1. Pregătire — 5 minute

Extrage intrările și condițiile din enunț. Notează ce rezultat trebuie produs și ce informații nu sunt confirmate. Pregătește fișierele sau tabelul în care vei păstra dovada lucrului.

2. Construire — 15 minute

Scrie contractul JSON și erorile înaintea controllerului. Folosește Resources pentru câmpurile publice și verifică efectele în baza de date pentru fiecare cerere.

3. Variație și verificare — 15 minute

Adaugă încă o pagină și verifică limitele paginării. Definește comportamentul pentru o pagină fără rezultate, fără să o confunzi cu o eroare de server.

4. Autoevaluare — 5 minute

Compară rezultatul cu reperele de mai jos. Înregistrează cel puțin o eroare identificată sau un caz-limită verificat, plus corectura ori explicația comportamentului observat.

Repere de rezolvare

Listarea nu expune câmpuri private. Sortarea folosește o listă permisă. Cererea invalidă și cea neautorizată nu creează servicii; răspunsul are format consecvent.

Ce predai

- Contract API
- Resources
- Validare
- Autentificare
- Teste HTTP

Criterii de acceptare

- Câmpurile private nu apar
- Sortarea are listă permisă
- Cererea invalidă nu modifică date

Păstrează o versiune inițială și una revizuită, cu explicația diferențelor. Dacă un criteriu nu este îndeplinit, revino la lecția relevantă și repetă verificarea. Durata este orientativă; instalarea mediului și aprofundarea pot necesita timp suplimentar.

Registrul de lucru

Pentru fiecare etapă notează intrarea, rezultatul, dovada verificării, eroarea găsită și corectura. Nu introduce date personale sau secrete în fișa partajată.

Documentație și surse

- Documentația Laravel 11 — versiunea aplicației demonstrative — Laravel
- OWASP Top 10 — OWASP

<https://laravel.com/docs/11.x>

<https://owasp.org/projects/top-ten>