

# Catalog Laravel și cereri de ofertă

## Fișă publică de proiect · exerciții și criterii de acceptare

Construiești un catalog de servicii, un formular și o zonă de administrare. Alegi explicit versiunea Laravel și consulți documentația aceleiași ramuri pe tot parcursul.

## Livrabile

- Rute și Blade
- Modele și migrații
- Form Requests
- Policy
- Fișă de deploy

## Criterii de acceptare

- Un utilizator neautorizat nu poate edita
- Formularul invalid nu salvează
- Build-ul și health check-ul sunt verificate

## Contractul primului proiect Laravel

Primul site demonstrativ gestionează solicitări de service, nu plăți. Definește ce vede vizitatorul și ce vede operatorul înainte de instalare. Un proiect mic, complet, este mai util decât multe funcții începute. Folosește Laravel 11 pentru concordanța cu aplicația existentă și notează versiunea PHP; pentru un proiect nou, verifică separat suportul versiunii alese. Nu copia automat configurația unui curs într-un serviciu public.

## Exemplu de lucru

Vizitator: listă servicii, formular, confirmare fără date sensibile.  
Operator autentificat: listă cereri, detalii, schimbare status permisă.  
Date: nume fictiv, email demo, serviciu, mesaj, status.  
Criterii: validare server; acces operator; erori clare; test formular.  
Exclus din primul proiect: plăți și trimitere reală de email.

## Interpretarea rezultatului

Această specificație permite verificarea proiectului fără servicii externe. HTTP 200 pe pagina principală nu dovedește că cererea este salvată sau că operatorul este autorizat. Păstrează o listă de acceptare și testează traseul vizitator-operator de la început la sfârșit.

## Exercițiu de consolidare

Adaugă o cerință de ștergere și explică ce permisiune, confirmare și politică de păstrare ar deveni necesare.

## 1. Ce construim și de ce Laravel

Adaugă o cerință de ștergere și explică ce permisiune, confirmare și politică de păstrare ar deveni necesare.

## Verificare orientativă

Această specificație permite verificarea proiectului fără servicii externe. HTTP 200 pe pagina principală nu dovedește că cererea este salvată sau că operatorul este autorizat. Păstrează o listă de acceptare și testează traseul vizitator-operator de la început la sfârșit.

## De evitat

Nu pretinde că demonstrația reproduce codul privat al unui client.

## Instalare izolată și verificarea mediului

Instalează proiectul într-un director nou și folosește o bază de date de laborator. Nu rula migrații pe o conexiune de producție din greșeală. Composer instalează dependențele PHP; npm gestionează partea frontend dacă proiectul o folosește. Fișierul .env conține configurația locală și nu trebuie publicat în repository. Comenzile de mai jos sunt pentru versiunea didactică 11, nu o recomandare de migrare a site-ului live.

## Exemplu de lucru

```
composer create-project laravel/laravel atelier-demo "^11.0"
cd atelier-demo
php artisan --version
php artisan about
php artisan key:generate
# Confirmă DB_CONNECTION și DB_DATABASE pentru laborator.
php artisan migrate
php artisan serve --host=127.0.0.1 --port=8002
```

## Interpretarea rezultatului

Verifică versiunea afișată și conexiunea înaintea migrate. Dacă instalarea eșuează din cauza extensiilor, rezolvă cerința concretă; nu dezactivează arbitrar verificările platformei. APP\_KEY trebuie generată pentru proiectul propriu și păstrată secretă.

## Exercițiu de consolidare

Identifică directoarele app, routes, resources, database și storage și notează câte un fișier concret cu rolul lui.

## 2. Instalare și structura proiectului

Identifică directoarele app, routes, resources, database și storage și notează câte un fișier concret cu rolul lui.

## Verificare orientativă

Verifică versiunea afișată și conexiunea înaintea migrate. Dacă instalarea eșuează din cauza extensiilor, rezolvă cerința concretă; nu dezactivează arbitrar verificările platformei. APP\_KEY trebuie generată pentru proiectul propriu și păstrată secretă.

## De evitat

Nu amesteca instrucțiuni de instalare pentru versiuni diferite.

## O rută, un controller și un view cu date explicite

Ruta leagă URL-ul de comportament, controllerul pregătește datele, iar Blade le afișează. Nu pune interogări complexe sau efecte secundare într-un template. Pentru primul pas, folosește un array local ca să poți verifica traseul fără DB. Sintaxa {{ }} escapează textul pentru HTML; afișarea brută trebuie rezervată conținutului controlat și sanitizat după politica aplicației.

## Exemplu de lucru

```
routes/web.php:
Route::get('/servicii', [ServiceController::class, 'index']);
Controller:
public function index() {
    return view('services.index', ['services' => ['Revizie', 'Reglaj frâne']]);
}
resources/views/services/index.blade.php:
<h1>Servicii</h1>
<ul>@foreach($services as $service)<li>{{ $service }}</li>@endforeach</ul>
Generare controller: php artisan make:controller ServiceController
```

## Interpretarea rezultatului

Adaugă importul controllerului în fișierul de rute. Pagina trebuie să afișeze două servicii. Dacă view-ul lipsește, eroarea aparține rezolvării template-ului, nu rutei. Verifică route:list pentru URL și metodă înainte de a căuta probleme în CSS.

## Exercițiu de consolidare

Înlocuiește un nume cu text care conține markup și confirmă că Blade îl afișează ca text.

## 3. Rute, controllere, Blade

Înlocuiește un nume cu text care conține markup și confirmă că Blade îl afișează ca text.

### Verificare orientativă

Adaugă importul controllerului în fișierul de rute. Pagina trebuie să afișeze două servicii. Dacă view-ul lipsește, eroarea aparține rezolvării template-ului, nu rutei. Verifică route:list pentru URL și metodă înainte de a căuta probleme în CSS.

### De evitat

Output-ul raw trebuie rezervat conținutului HTML de încredere.

## Migrare și model pentru o cerere de service

Schema definește câmpurile și constrângerile, iar modelul expune operațiile asupra datelor. Începe cu un status implicit și cu timestampuri pentru urmărire. Nu accepta din formular toate câmpurile modelului; validarea și lista permisă sunt separate de migrare. Folosește numele tabelului și al modelului consecvent. O migrare rulată nu este dovada că aplicația salvează datele corect; verifică și înregistrarea creată.

## Exemplu de lucru

```
php artisan make:model ServiceRequest -m
În migrare:
Schema::create('service_requests', function (Blueprint $table) {
    $table->id();
    $table->string('email');
    $table->string('service');
    $table->text('message');
    $table->string('status')->default('new');
    $table->timestamps();
});
În model: protected $fillable = ['email', 'service', 'message'];
```

## Interpretarea rezultatului

Statusul este stabilit de aplicație și nu se află în lista câmpurilor editabile de vizitator. Pentru integritate, definește separat valorile de status permise și tranzițiile. Testează o cerere care trimite status=approved și verifică să nu poată ocoli fluxul.

## Exercițiu de consolidare

Adaugă un câmp opțional telefon și verifică migrarea pe o bază cu înregistrări existente, în laborator.

## 4. Baza de date cu migrații și Eloquent

Adaugă un câmp opțional telefon și verifică migrarea pe o bază cu înregistrări existente, în laborator.

### Verificare orientativă

Statusul este stabilit de aplicație și nu se află în lista câmpurilor editabile de vizitator. Pentru integritate, definește separat valorile de status permise și tranzițiile. Testează o cerere care trimite status=approved și verifică să nu poată ocoli fluxul.

### De evitat

Nu testa rollback pe baza de date de producție.

## Form Request cu câmpuri permise și erori testabile

Un Form Request grupează validarea și poate decide dacă solicitantul este autorizat pentru operație. Pentru formularul public demonstrativ, authorize permite depunerea; pentru o operație privată trebuie folosită regula de acces potrivită. Controllerul folosește validated, nu all. Regula in limitează serviciile acceptate, iar trim și politica pentru spații trebuie verificate în contextul middleware-urilor aplicației.

### Exemplu de lucru

```
php artisan make:request StoreServiceRequest
public function authorize(): bool { return true; }
public function rules(): array {
    return [
        'email' => ['required', 'email', 'max:190'],
        'service' => ['required', 'in:revizie,frane'],
        'message' => ['required', 'string', 'min:10', 'max:1000'],
    ];
}
```

Controller: ServiceRequest::create(\$request->validated());

### Interpretarea rezultatului

Validarea respinge un serviciu necunoscut și nu include un status introdus suplimentar. Într-o cerere JSON, erorile au forma răspunsului de validare al aplicației; într-un formular web, fluxul poate redirecționa cu erori. Testează contractul concret, nu presupune aceeași experiență pentru ambele.

## Exercițiu de consolidare

Scrie un test pentru email invalid și unul pentru câmp suplimentar status, verificând și baza de date.

## 5. Formulare, validare, Form Requests

Scrie un test pentru email invalid și unul pentru câmp suplimentar status, verificând și baza de date.

### Verificare orientativă

Validarea respinge un serviciu necunoscut și nu include un status introdus suplimentar. Într-o cerere JSON, erorile au forma răspunsului de validare al aplicației; într-un formular web, fluxul poate redirecționa cu erori. Testează contractul concret, nu presupune aceeași experiență pentru ambele.

## De evitat

Regulile din frontend nu înlocuiesc validarea serverului.

## Zonă privată și verificarea accesului pe fiecare rută

Middleware-ul auth cere autentificare, dar nu decide singur dacă un utilizator poate vedea toate cererile. Pentru un operator unic, poți avea o regulă explicită de rol; pentru mai mulți utilizatori, limitează resursele după proprietar sau policy. Nu te baza pe ascunderea linkului din meniu. O cerere directă la URL trebuie să fie verificată la fel. Folosește mecanismul de autentificare al proiectului, fără a inventa un login în controller.

### Exemplu de lucru

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

În controller: autorizare explicită pentru rolul operator.

Teste: vizitator → login; utilizator fără rol → 403; operator → 200.

Nu folosi: if în Blade ca singur control de acces.

### Interpretarea rezultatului

Cele trei rezultate reprezintă stări distincte. Autentificarea reușită nu este echivalentă cu rolul operator. Verifică și operațiile POST de schimbare a statusului; protejarea paginii GET nu protejează automat endpointul care modifică datele.

### Exercițiu de consolidare

Încearcă direct endpointul POST cu un utilizator fără rol și verifică faptul că statusul rămâne neschimbat.

## 6. Autentificare și zonă privată

Încearcă direct endpointul POST cu un utilizator fără rol și verifică faptul că statusul rămâne neschimbat.

### Verificare orientativă

Cele trei rezultate reprezintă stări distincte. Autentificarea reușită nu este echivalentă cu rolul operator. Verifică și operațiile POST de schimbare a statusului; protejarea paginii GET nu protejează automat endpointul care modifică datele.

## De evitat

Ascunderea butonului nu protejează endpoint-ul.

## Deploy verificat printr-un traseu real de utilizare

Publicarea Laravel cere ca document root să fie public, iar configurația și storage privat să rămână inaccesibile prin web. Instalează dependențele potrivite, compilează asseturile și confirmă conexiunea DB înaintea migrațiilor. Un health check simplu nu înlocuiește testarea formularului și accesului operatorului. Păstrează backup și o versiune anterioară care poate fi restaurată. Comenzile se adaptează infrastructurii, nu se rulează orbește ca root.

### Exemplu de lucru

Checklist: document root → proiect/public; .env nepublic; APP\_DEBUG=false; cheie proprie; permisiuni storage; backup DB; build; migrații revizuite; cache-uri; HTTPS.

Comenzi uzuale: composer install --no-dev --optimize-autoloader; npm ci; npm run build; php artisan migrate --force.

După deploy: pagină, formular, acces, loguri, resurse statice.

## Interpretarea rezultatului

--force elimină o confirmare interactivă, nu face migrarea sigură. Verifică înainte ce SQL și ce impact are schimbarea. Un rollback al codului poate fi incompatibil cu schema nouă; planul de revenire trebuie să țină cont de ambele.

## Exercițiu de consolidare

Construiește o listă de cinci cereri HTTP și două verificări DB care ar demonstra că proiectul de service funcționează după deploy.

## 7. Deploy pe un server real

Construiește o listă de cinci cereri HTTP și două verificări DB care ar demonstra că proiectul de service funcționează după deploy.

### Verificare orientativă

--force elimină o confirmare interactivă, nu face migrarea sigură. Verifică înainte ce SQL și ce impact are schimbarea. Un rollback al codului poate fi incompatibil cu schema nouă; planul de revenire trebuie să țină cont de ambele.

### De evitat

Un deploy reușit în terminal poate lăsa aplicația nefuncțională.

## Atelier final: Catalog Laravel cu administrare protejată

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

Proiect local Laravel 11: Service are title și price\_cents; cererea de ofertă are email și service\_id. Vizitatorul poate vedea catalogul, administratorul poate modifica serviciu.

### 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

Creează migrarea, modelul, rutele, Blade, Form Request și policy. Testează un service\_id inexistent și un utilizator fără drept de editare.

### 3. Variație și verificare — 15 minute

Adaugă un serviciu retras din ofertă. Decide explicit dacă rămâne vizibil în cereri istorice și testează că nu poate fi ales pentru o cerere nouă.

### 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

Prețul este stocat în unități întregi, iar afișarea îl formatează. ID-ul invalid nu creează cererea. Butonul ascuns în Blade nu înlocuiește controlul de acces pe server.

## Ce predai

- Rute și Blade
- Modele și migrații
- Form Requests
- Policy
- Fișă de deploy

## Criterii de acceptare

- Un utilizator neautorizat nu poate edita
- Formularul invalid nu salvează
- Build-ul și health check-ul sunt verificate

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>