

Modul de comenzi cu responsabilități separate

Fișă publică de proiect · exerciții și criterii de acceptare

Evoluezi o aplicație demonstrativă de la controller mare la request, service, policy și jobs. Introduci abstracții numai pentru responsabilități concrete.

Livrabile

- Hartă de flux
- Serviciu testabil
- Policy
- Job după commit
- Teste de regresie

Criterii de acceptare

- Refactorizarea păstrează comportamentul
- Notificarea nu pleacă la rollback
- Accesul este verificat server-side

Granițe între HTTP, regulă de business și persistență

O aplicație crește mai ușor când regula principală nu depinde direct de cererea HTTP. Controllerul validează și autorizează, serviciul execută operația, iar modelul reprezintă datele. Nu este necesar un strat pentru fiecare linie de cod; separarea are sens când o regulă este reutilizată sau are efecte coordonate. Pentru aprobarea unei solicitări, definește starea inițială permisă și rezultatul, nu doar numele metodei.

Exemplu de lucru

Flux: Request validat → policy → ApproveServiceRequest → tranzacție → răspuns.

Regulă: numai cererile new pot deveni approved.

Invariante: proprietarul nu se schimbă; o aprobare repetată nu creează două notificări.

Teste: stare validă, stare invalidă, acces refuzat, repetare.

Interpretarea rezultatului

Un controller scurt nu este automat o arhitectură bună dacă serviciul acceptă orice stare. Urmărește regula și efectele secundare. Pentru operații concurente, simpla verificare a statusului înaintea salvării poate necesita protecție suplimentară prin tranzacție sau actualizare condiționată.

Exercițiu de consolidare

Describe unde ai pune validarea formatului emailului și unde regula „o cerere anulată nu poate fi aprobată”.

1. Anatomia unei aplicații care crește sănătos

Describe unde ai pune validarea formatului emailului și unde regula „o cerere anulată nu poate fi aprobată”.

Verificare orientativă

Un controller scurt nu este automat o arhitectură bună dacă serviciul acceptă orice stare. Urmărește regula și efectele secundare. Pentru operații concurente, simpla verificare a statusului înaintea salvării poate necesita protecție suplimentară prin tranzacție sau actualizare condiționată.

De evitat

Nu adăuga abstracții fără o responsabilitate reală.

Serviciu cu actualizare condiționată și rezultat explicit

Pentru o operație de stare, o actualizare condiționată poate preveni suprascrierea unei schimbări concurente. Verifică numărul de rânduri afectate și întoarce un rezultat clar. Nu trimite notificarea înainte ca modificarea să fie confirmată. Exemplul izolează tranziția new→approved; autorizarea trebuie făcută înainte de apel sau inclusă într-un contract explicit al serviciului. Nu presupune că orice model primit aparține utilizatorului curent.

Exemplu de lucru

```
public function approve(int $id): void {
    $changed = ServiceRequest::whereKey($id)
        ->where('status', 'new')
        ->update(['status' => 'approved', 'updated_at' => now()]);
    if ($changed !== 1) {
        throw new DomainException('Cererea nu mai poate fi aprobată.');
```

```
}
```

```
// Notificarea se coordonează separat, după confirmarea schimbării.
```

Interpretarea rezultatului

Primul apel afectează un rând, iar al doilea nu mai găsește statusul new. Contractul alege aici să semnaleze repetarea ca imposibilitate; un alt produs poate alege idempotență cu succes fără schimbare. Definește comportamentul, nu îl lăsa accidental.

Exercițiu de consolidare

Scrie un test pentru două apeluri consecutive și verifică statusul final și numărul notificărilor.

2. Servicii pentru logica de business

Scrie un test pentru două apeluri consecutive și verifică statusul final și numărul notificărilor.

Verificare orientativă

Primul apel afectează un rând, iar al doilea nu mai găsește statusul new. Contractul alege aici să semnaleze repetarea ca imposibilitate; un alt produs poate alege idempotență cu succes fără schimbare. Definește comportamentul, nu îl lăsa accidental.

De evitat

Dependențele globale fac logica greu de reutilizat.

Validare condiționată fără a autoriza implicit

Unele câmpuri devin obligatorii doar pentru o anumită operație. Într-o cerere de schimbare a statusului, motivul poate fi cerut pentru anulare, iar statusul trebuie limitat la valorile acceptate. Form Request validează forma și poate delega accesul unei policy. Nu confunda „statusul este în listă” cu „tranziția este permisă din starea actuală”. Aceasta rămâne o regulă de business verificată aproape de actualizare.

Exemplu de lucru

```
public function rules(): array {
    return [
        'status' => ['required', 'in:approved,cancelled'],
        'reason' => ['nullable', 'string', 'max:500', 'required_if:status,cancelled'],
    ];
}
```

Teste: cancelled fără motiv → eroare; approved fără motiv → format valid; status arbitrar → eroare.

Apoi: policy și verificarea tranziției curente.

Interpretarea rezultatului

O intrare poate trece validarea și totuși să fie refuzată deoarece resursa este deja închisă. Răspunsul trebuie să explice această diferență în contractul aplicației. Nu salva un câmp suplimentar din cerere doar pentru că alte câmpuri au fost validate.

Exercițiu de consolidare

Adaugă un test cu reason prea lung și unul cu un status valid, dar nepermis pentru starea resursei.

3. Form Requests avansate

Adaugă un test cu reason prea lung și unul cu un status valid, dar nepermis pentru starea resursei.

Verificare orientativă

O intrare poate trece validarea și totuși să fie refuzată deoarece resursa este deja închisă. Răspunsul trebuie să explice această diferență în contractul aplicației. Nu salva un câmp suplimentar din cerere doar pentru că alte câmpuri au fost validate.

De evitat

Validarea doar a primului element lasă restul necontrolat.

Policy care compară identitatea și proprietarul

O policy exprimă dreptul asupra unei resurse. Pentru laborator, proprietarul poate actualiza cererea, iar alt utilizator nu. Nu primi identitatea dintr-un câmp ascuns al formularului; folosește utilizatorul autentificat. Tipurile trebuie comparate consecvent, iar rolurile administrative se adaugă explicit dacă produsul le cere. Route model binding găsește modelul, dar nu este singur un control de autorizare.

Exemplu de lucru

```
public function update(User $user, ServiceRequest $request): bool {
    return $request->user_id === $user->id;
}
```

În controller:

```
Gate::authorize('update', $serviceRequest);
```

```
$serviceRequest->update($validated);
```

Teste: proprietar → permis; alt utilizator → 403; vizitator → autentificare cerută.

Interpretarea rezultatului

Exemplul presupune că `user_id` și `id` sunt reprezentate compatibil de modele. Verifică și endpointurile alternative, nu doar cel principal. Dacă lista este filtrată după proprietar, dar detaliul nu este autorizat, un identificator ghicit poate expune date.

Exercițiu de consolidare

Adaugă un rol operator și definește dacă poate citi toate cererile sau și modifica, fără a combina automat cele două permisiuni.

4. Policies și Gates

Adaugă un rol operator și definește dacă poate citi toate cererile sau și modifica, fără a combina automat cele două permisiuni.

Verificare orientativă

Exemplul presupune că `user_id` și `id` sunt reprezentate compatibil de modele. Verifică și endpointurile alternative, nu doar cel principal. Dacă lista este filtrată după proprietar, dar detaliul nu este autorizat, un identificator ghicit poate expune date.

De evitat

Un id greu de ghicit nu înlocuiește autorizarea.

Job reluabil fără efecte duplicate

Un job poate fi executat de mai multe ori. Proiectează-l astfel încât repetarea să nu dubleze operația logică sau să fie detectată. Nu serializa secrete sau documente complete dacă poate primi un identificator. Datele pot fi schimbate ori șterse între dispatch și execuție, deci jobul trebuie să trateze acest caz. Pentru expediere externă, păstrează identificatorul și statusul necesar reconcilierii, nu doar un flag setat prea devreme.

Exemplu de lucru

Job: `SendRequestSummary(requestId)`.

La execuție: citește resursa; verifică statusul; obține cheia operației; expediază prin mecanism controlat; salvează dovada.

Cazuri: resursă lipsă; furnizor indisponibil; răspuns pierdut după succes; job repetat.

Nu presupune: „coada execută exact o singură dată”.

Interpretarea rezultatului

Retry-ul după timeout poate produce duplicare dacă furnizorul a acceptat deja mesajul. Folosește contractul de idempotență al furnizorului sau un mecanism de reconciliere potrivit. O tranzacție DB nu anulează automat un email deja trimis.

Exercițiu de consolidare

Scrie un test cu furnizor fals care acceptă prima cerere, apoi simulează pierderea răspunsului.

5. Jobs și cozi

Scrie un test cu furnizor fals care acceptă prima cerere, apoi simulează pierderea răspunsului.

Verificare orientativă

Retry-ul după timeout poate produce duplicare dacă furnizorul a acceptat deja mesajul. Folosește contractul de idempotență al furnizorului sau un mecanism de reconciliere potrivit. O tranzacție DB nu anulează automat un email deja trimis.

De evitat

Un job poate rula înainte ca datele să fie confirmate.

Eveniment după o schimbare confirmată

Un eveniment descrie ceva ce s-a întâmplat, nu doar o intenție. Dacă o cerere este aprobată într-o tranzacție, notificarea trebuie coordonată cu confirmarea acesteia. Un listener nu trebuie să depindă de o stare necomisă pe care un alt proces nu o poate citi. Păstrează evenimentul mic și clar, apoi verifică ce se întâmplă la eșecul listenerului. Operația principală și notificarea pot avea politici diferite de reluare.

Exemplu de lucru

Eveniment: `ServiceRequestApproved(requestId)`.
Emiterea: numai după tranziție reușită și commit, conform mecanismului aplicației.
Listener A: audit intern.
Listener B: notificare în coadă.
Teste: rollback → fără notificare; commit → o operație logică; retry → fără duplicare.

Interpretarea rezultatului

Emiterea înainte de commit poate produce o notificare pentru o aprobare care este ulterior anulată de rollback. Pentru listeners queued, verifică opțiunile after-commit ale versiunii Laravel utilizate. Nu testa doar că `event()` a fost apelat; verifică momentul și efectul.

Exercițiu de consolidare

Definește un caz în care notificarea eșuează, dar aprobarea trebuie să rămână validă și vizibilă operatorului.

6. Events și listeners

Definește un caz în care notificarea eșuează, dar aprobarea trebuie să rămână validă și vizibilă operatorului.

Verificare orientativă

Emiterea înainte de commit poate produce o notificare pentru o aprobare care este ulterior anulată de rollback. Pentru listeners queued, verifică opțiunile after-commit ale versiunii Laravel utilizate. Nu testa doar că `event()` a fost apelat; verifică momentul și efectul.

De evitat

Events nu trebuie să ascundă regula principală de business.

Organizare pe domenii fără dependențe circulare

Folderele pe domenii ajută când grupează reguli care se schimbă împreună. Nu muta toate clasele doar pentru un aspect mai ordonat dacă importurile devin circulare. În proiectul de service, cererile și notificările pot fi separate, iar domeniul nu ar trebui să citească direct Request HTTP. Păstrează convențiile frameworkului pentru punctele de intrare și testează rezolvarea claselor după mutare. Organizarea se evaluează prin claritatea dependențelor.

Exemplu de lucru

```
app/Domain/Requests/ApproveRequest.php
app/Domain/Requests/RequestStatus.php
app/Http/Controllers/OperatorRequestController.php
app/Http/Requests/ChangeRequestStatus.php
app/Policies/ServiceRequestPolicy.php
```

Dependență permisă: controller → regulă de domeniu.
De evitat: regulă de domeniu → controller.

Interpretarea rezultatului

Un folder Domain nu face automat codul independent. Verifică importurile și efectele secundare. Dacă mutarea schimbă nume de evenimente, bindings sau descoperirea policy-urilor, actualizează și verificările aferente. Nu combina reorganizarea cu schimbarea regulilor de status.

Exercițiu de consolidare

Desenează printr-o listă dependențele a cinci clase și identifică una care ar trebui inversată sau eliminată.

7. Foldere pe domenii

Desenează printr-o listă dependențele a cinci clase și identifică una care ar trebui inversată sau eliminată.

Verificare orientativă

Un folder Domain nu face automat codul independent. Verifică importurile și efectele secundare. Dacă mutarea schimbă nume de evenimente, bindings sau descoperirea policy-urilor, actualizează și verificările aferente. Nu combina reorganizarea cu schimbarea regulilor de status.

De evitat

Mutarea fișierelor fără reguli de dependență nu îmbunătățește arhitectura.

Atelier final: Comandă și notificare după tranzacție

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

Comanda fictivă are două poziții: 2 × 1.500 bani și 1 × 2.000 bani. Notificarea trebuie planificată doar dacă salvarea comenzii reușește. Simulează o excepție la a doua poziție.

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

Separă validarea, autorizarea și serviciul de creare. Testează succesul și rollbackul. Documentează momentul în care jobul devine eligibil pentru rulare.

3. Variație și verificare — 15 minute

Rulează de două ori aceeași intenție de creare și definește comportamentul așteptat. Arată ce control suplimentar ar fi necesar pentru idempotență.

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

Totalul este 5.000 bani. La excepție nu rămâne o comandă parțială și nu se trimite notificarea de succes. Testul verifică și baza, și coada de joburi.

Ce predai

- Hartă de flux
- Serviciu testabil
- Policy
- Job după commit
- Teste de regresie

Criterii de acceptare

- Refactorizarea păstrează comportamentul
- Notificarea nu pleacă la rollback
- Accesul este verificat server-side

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>