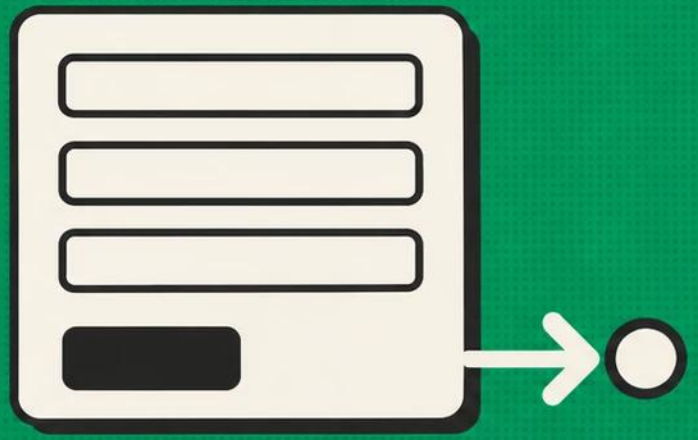
1.12

TEMPS 1 · LE PORTFOLIO · SÉANCE 3

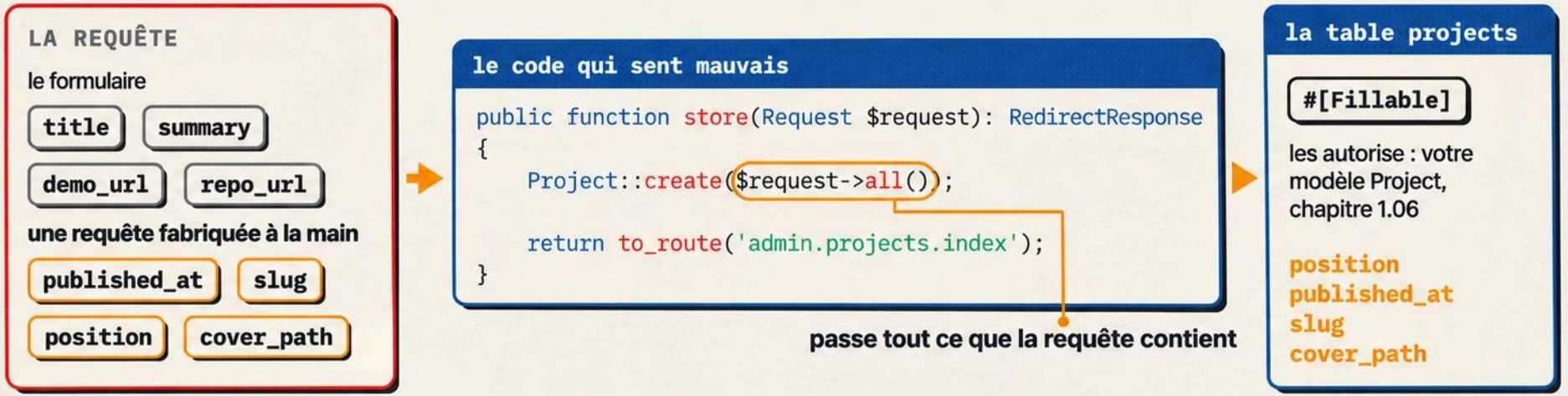
Créer un projet

Thibault Six · IFOSUP Wavre · 2026-2027



? Le formulaire a quatre champs. Que peut faire entrer $request->all() dans la table projects ?

$request->all() fait entrer tout ce que la requête contient, et rien n'est vérifié avant la base.



un titre vide → la base le refuse → 500 le visiteur reçoit une erreur 500 au lieu d'un message sous le champ

En gris, ce que le formulaire envoie ; en orange, ce qu'une requête fabriquée peut ajouter. Tout arrive dans la table sans contrôle.

1.12

? Où écrire les règles du formulaire, une seule fois pour la création et la modification ?

La Form Request ProjectRequest contient les règles, et Laravel les applique avant d'appeler le contrôleur.

```
$ php artisan make:request ProjectRequest
INFO  Request [app/Http/Requests/ProjectRequest.php] created successfully.
```

~~authorize()~~ elle renvoie `false` : chaque envoi recevrait une erreur 403. Supprimez-la, le middleware auth protège déjà le back-office

app/Http/Requests/ProjectRequest.php · rules() — extrait

```
'title' => ['required', 'string', 'max:255'],
'slug' => $this->isMethod('POST') ? [Rule::unique('projects', 'slug')] : ['prohibited'],
'summary' => ['required', 'string', 'max:500'],
'content' => ['nullable', 'string'],
'demo_url' => ['nullable', 'url', 'max:255'],
'repo_url' => ['nullable', 'url', 'max:255'],
'tags' => ['array'],
'tags.*' => ['integer', 'exists:tags,id'],
```

'max:500' comme `string('summary', 500)` dans la migration

'url' si le lien est rempli, une adresse complète, comme `https://exemple.be`

'exists:tags,id' chaque étiquette doit exister dans la table tags

le formulaire → **ProjectRequest** → **store(ProjectRequest $request)** quand store() s'exécute, la validation est déjà passée

Une ligne par champ, une liste de règles par ligne. La classe se place entre le formulaire et store().

1.12

? Un résumé vide donne « Le champ summary est obligatoire. ». D'où vient le mot summary ?

Laravel-Lang traduit les messages, et attributes() donne à chaque champ son nom français, écrit une seule fois.

```
$ composer require laravel-lang/common --dev
```

```
$ php artisan lang:add fr
```

```
.env  APP_LOCALE=fr
```

The title field is required.

Le champ **summary** est obligatoire.

Le champ résumé est obligatoire. ✓

les réglages du kit

Laravel-Lang : le message traduit, le nom du champ resté anglais

les traductions connaissent déjà `title`, traduit « titre »

app/Http/Requests/ProjectRequest.php · attributes()

```php
return [
    'summary' => 'résumé',
    'demo_url' => 'lien de la démo',
    'repo_url' => 'lien du code',
];
```

Suivez le message de gauche à droite : d'abord traduit, puis avec le nom français que donne attributes().

? Quand store() s'exécute, la validation est passée. Que lui reste-t-il à faire ?

store() crée le projet avec les champs validés, enregistre ses étiquettes, prépare un message et redirige.

app/Http/Controllers/Admin/ProjectController.php

```php
public function store(ProjectRequest $request): RedirectResponse
{
    $project = Project::create([
        ...$request->safe()->except('tags'),
        'position' => Project::max('position') + 1,
    ]);

    $project->tags()->sync($request->validated('tags', []));

    Inertia::flash('toast', ['type' => 'success', 'message' => 'Projet créé.']);

    return to_route('admin.projects.index');
}
```

au lieu de **Request**

1. les champs validés, sans les étiquettes ; le slug calculé en fait partie
2. place le nouveau projet après les autres
3. enregistre les étiquettes dans la table `project_tag`
4. prépare un message pour la page suivante
5. redirige vers la liste

`create()` envoie à la page la liste des étiquettes, avec deux clés : `id` et `name`

Lisez *store*() dans l'ordre des numéros : chaque ligne fait une chose, et aucune ne vérifie un champ.

1.12

? Create.vue n'écrit nulle part /admin/projets. Comment <Form> sait-il où envoyer ?

Wayfinder donne à <Form> l'adresse et la méthode de store(), et novalidate laisse toute la validation à la Form Request.

resources/js/pages/admin/projects/Create.vue extrait

```
<Form
    v-bind="ProjectController.store.form()"
    v-slot="{ errors, processing }"
    class="space-y-6"
    novalidate
>
```

`{ action: '/admin/projets', method: 'post' }`

ce que renvoie la fonction générée par Wayfinder

`<Form action="/admin/projets" method="post">`

<Form> : un composant d'Inertia qui produit un <form> et l'envoie sans recharger la page

v-bind suivi d'un objet pose chaque clé comme un attribut : aucune adresse en dur

SANS NOVALIDATE

exemple.be → Veuillez saisir une URL.

la bulle de Chrome : rien n'est envoyé

gardez `type="url"` : sur un téléphone, il affiche le clavier des adresses

AVEC NOVALIDATE

exemple.be → **ProjectRequest** → **Le format de l'URL de lien de la démo n'est pas valide.**

tous les messages viennent du même endroit

En haut, d'où vient l'adresse d'envoi ; en bas, ce que change novalidate pour un lien mal formé.

1.12

? Les champs n'ont aucun v-model. Comment <Form> lit-il leurs valeurs, et où vont les erreurs ?

<Form> lit chaque champ par son name, et son slot fournit errors et processing aux messages et au bouton.

resources/js/pages/admin/projects/Create.vue — extrait

```
<div class="grid gap-2">
    <Label for="title">Titre</Label>
    <Input id="title" name="title" />
    <InputError :message="errors.title || errors.slug" />
</div>
```

CE QUI PART

`title` `summary` `demo_url` `repo_url` `tags[]`

un formulaire non contrôlé : Vue ne suit pas les valeurs, <Form> les lit par leur name au moment de l'envoi

CE QUE LE SLOT REND

`errors` → un message par champ refusé, vide tant que rien n'est refusé → `InputError` affiche le message en rouge, rien s'il est vide

pendant l'envoi : le bouton grisé, la roue de `<Spinner v-if="processing" />`

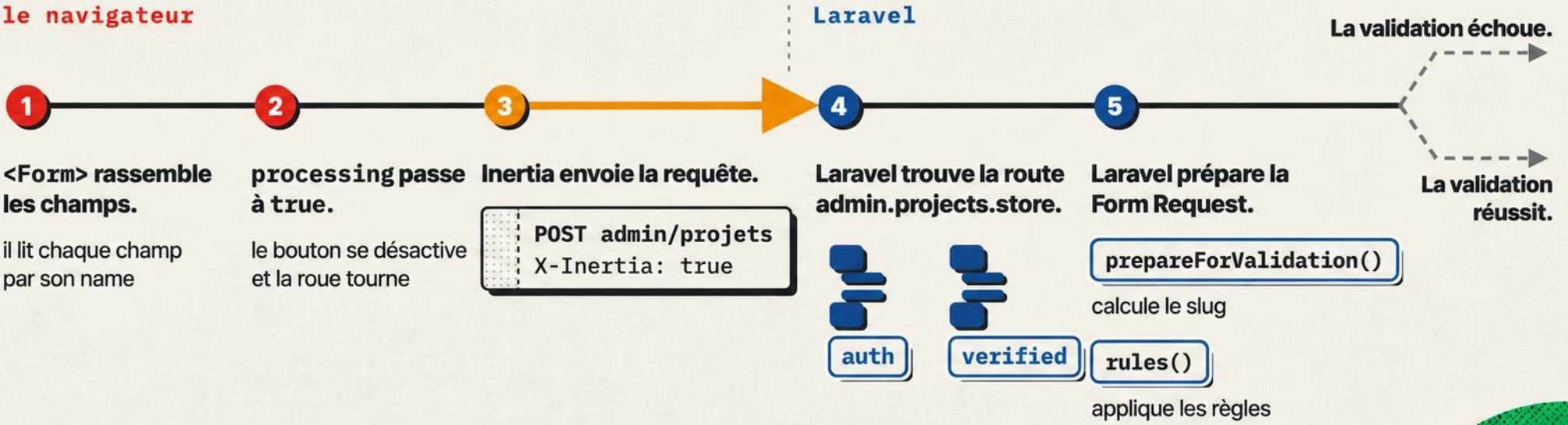
`processing` → true du clic jusqu'à la réponse du serveur → `<Button type="submit" :disabled="processing">` → Créer le projet

un double clic n'envoie pas deux fois le même projet

À gauche, ce qui part : chaque champ par son name. À droite, ce que le slot rend à la page pendant l'envoi.

? Vous cliquez sur « Créer le projet ». Que se passe-t-il avant que store() soit appelé, ou pas ?

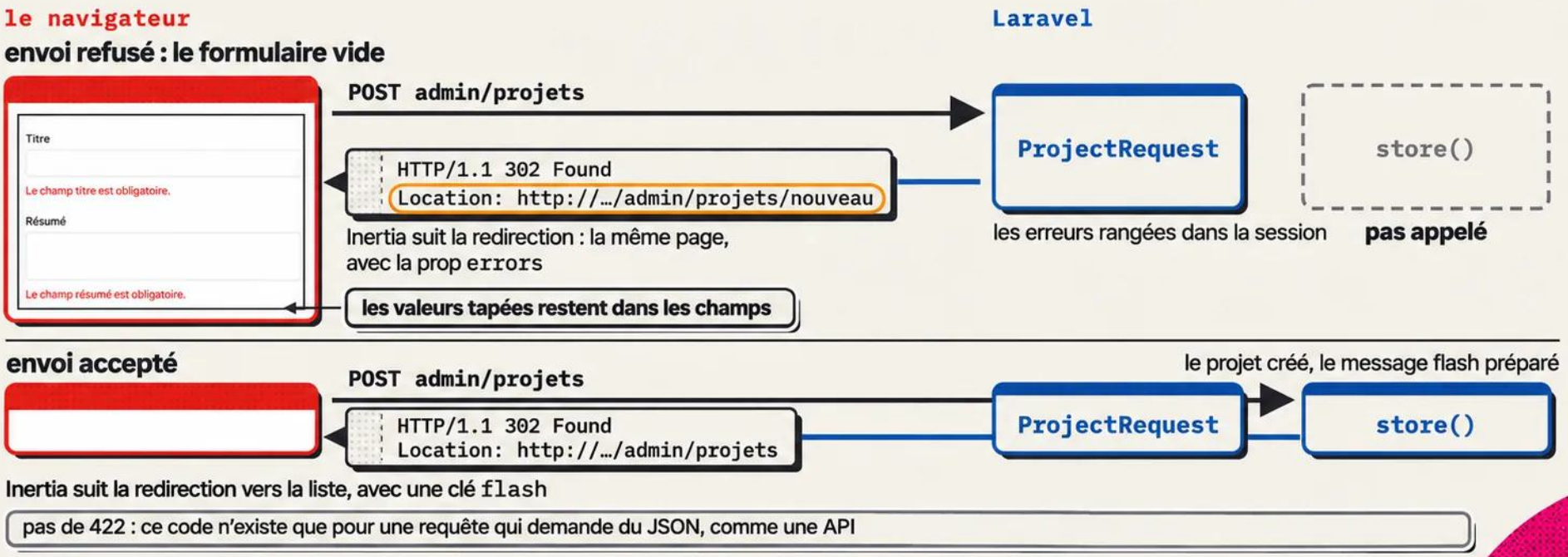
Avant store(), Inertia envoie un POST, Laravel passe les middlewares, puis la Form Request calcule le slug et valide.



Les cinq premières étapes sont les mêmes pour tout envoi. Le trajet ne se sépare qu'à la validation.

? Un envoi refusé reçoit-il un code 422 ? Qu'est-ce qui distingue sa réponse d'un envoi accepté ?

Refusé ou accepté, le serveur répond 302 : seule l'adresse de la redirection change, et la page reçoit errors ou flash.



Même requête, deux trajets. En haut, la Form Request refuse et revient au formulaire ; en bas, store() crée le projet et renvoie à la liste.

1.12

? store() a préparé « Projet créé. » sur le serveur. Qui l'affiche dans le navigateur ?

Inertia::flash() ajoute une clé flash à la réponse ; flashToast.ts l'écoute et affiche la notification.

```php
Inertia::flash('toast', ['type' => 'success', 'message' => 'Projet créé.']);
```

```
{"toast":{"type":"success","message":"Projet créé."}}
```

la clé flash, à côté des props de la page suivante

resources/js/lib/flashToast.ts

```ts
export function initializeFlashToast(): void {
    router.on('flash', (event) => {
        const flash = (event as CustomEvent).detail?.flash;
        const data = flash?.toast as FlashToast | undefined;

        if (!data) {
            return;
        }

        toast[data.type](data.message);
    });
}
```

ici `toast.success('Projet créé.')`, du paquet vue-sonner

appelée chaque fois qu'une réponse apporte un message flash

`<Toaster />` dans le layout du back-office, chapitre 1.11



la notification, dans un coin de l'écran

le message vaut pour la page suivante seulement : un rechargement le fait disparaître

Suivez le message : préparé par le serveur, porté par la réponse, lu par flashToast.ts, affiché par le Toaster.

1.12

? Le formulaire n'a pas de champ slug. D'où vient l'adresse publique du nouveau projet ?

prepareForValidation() calcule le slug à partir du titre, avant la validation et seulement à la création, en POST.

app/Http/Requests/ProjectRequest.php · prepareForValidation() extrait

```php
if ($this->isMethod('POST')) {
    $this->merge([
        'slug' => Str::slug((string) $this->input('title')),
    ]);
}
```

ajoute une valeur à la requête, comme si le formulaire l'avait envoyée

Météo Wavre → `Str::slug()` → `meteo-wavre`

les accents disparaissent, les espaces deviennent des tirets, les majuscules passent en minuscules

```
$ php artisan tinker --execute="echo Illuminate\Support\Str::slug('Météo Wavre');"
meteo-wavre
```

```php
'slug' => $this->isMethod('POST') ? [Rule::unique('projects', 'slug')] : ['prohibited'],
```

POST : la création → `Rule::unique('projects', 'slug')` → **Un projet porte déjà ce titre.**

deux projets nommés « Météo Wavre » auraient la même adresse

PUT : la modification, chapitre 1.13 → `prohibited` → un projet garde l'adresse reçue à sa création : un lien partagé reste bon

En haut, le slug calculé avant la validation. En bas, la règle suit la même condition : unique en POST, interdit en PUT.

? Deux cases cochées sur trois. Comment arrivent-elles dans la table project_tag ?

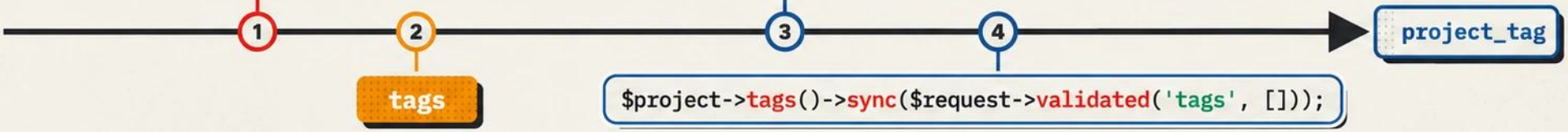
Les crochets de tags [] rassemblent les cases cochées en un tableau, et sync() rend project_tag identique à cette liste.

la case de shadcn-vue ajoute une vraie case HTML, cachée, qui porte le name et la valeur

```
<Checkbox name="tags[]" :value="String(tag.id)" />
```

chaque élément : un entier, présent dans la colonne id de la table tags

```
'tags.*' => ['integer', 'exists:tags,id'],
```



un seul tableau : deux cases cochées, deux numéros

aucune case cochée : un tableau vide

vérifiez avec le slug de votre nouveau projet

```
$ php artisan tinker --execute="echo App\Models\Project::firstWhere('slug', 'quiz-des-capitales')->tags->pluck('name');"
["Vue.js","JavaScript"]
```

Suivez les cases cochées : un tableau dans la requête, chaque numéro vérifié, puis sync() qui écrit dans project_tag.

? Qu'est-ce qui doit être vrai à la fin ?

À vous, trente-cinq minutes : un formulaire qui refuse en français, crée le projet et affiche « Projet créé. ».

À FAIRE

1. `ProjectRequest` sans `authorize()`, avec ses règles, `prepareForValidation()`, `messages()` et `attributes()`.
2. Laravel-Lang installé, `lang:add fr`, puis `APP_LOCALE=fr` dans `.env`.
3. `create()` et `store()` dans `Admin\ProjectController`.
4. La page `admin/projects/Create.vue`.
5. Puis créez un vrai projet : un quatrième projet de votre parcours, avec son titre, son résumé, ses liens et ses étiquettes.

tapez le code, ne le collez pas

À VÉRIFIER

35 min

- [ ] Le bouton « Nouveau projet » de la liste ouvre le formulaire, avec le fil d'Ariane « Projets › Nouveau projet ».
- [ ] Envoyé vide, le formulaire affiche un message en français sous le titre et sous le résumé, et aucun projet n'est créé.
- [ ] Un lien de démo sans `https://` affiche « Le format de l'URL de lien de la démo n'est pas valide. », et les autres champs gardent leur valeur.
- [ ] Un titre déjà utilisé affiche « Un projet porte déjà ce titre. » sous le titre.
- [ ] Un envoi valide ramène à la liste, avec le nouveau projet en dernière ligne et la notification « Projet créé. ».

Cinq étapes à gauche, cinq vérifications à droite ; finissez par les étiquettes du nouveau projet, dans Tinker.

1.12