1.20

TEMPS 1 · LE PORTFOLIO · SÉANCE 5

Contact et e-mail

Thibault Six · IFOSUP Wavre · 2026-2027



5XDYN · Projet web dynamique · 1.20

? Un visiteur vous écrit depuis le portfolio. Que doit-il se passer avant qu'il lise « Merci » ?

Le message est vérifié, enregistré, envoyé par e-mail, puis confirmé au visiteur ; la base le garde si l'e-mail se perd.



Suivez la ligne de gauche à droite : chaque station a son rôle. Un message refusé à la première ne va pas plus loin.

1.20

? Le message doit rester dans la base. Quel modèle le garde, et avec quelles colonnes ?

ContactMessage garde le nom, l'adresse et le message ; la colonne text accepte plus de 255 caractères.

```
$ php artisan make:model ContactMessage -m
```

-m : crée aussi la migration

database/migrations/…_create_contact_messages_table.php extrait

```
$table->id();
$table->string('name');
$table->string('email');
$table->text('message');
$table->timestamps();
```

les trois colonnes ajoutées entre `id()` et `timestamps()`

`text` accepte plus que les 255 caractères d'une colonne `string`

app/Models/ContactMessage.php

```
#[Fillable(['name', 'email', 'message'])]
class ContactMessage extends Model
{
    //
}
```

les colonnes remplissables, avec l'attribut du chapitre 1.06

```
$ php artisan migrate
```

→

```
2026_10_27_090831_create_contact_messages_table .. 6.68ms DONE
```

→ table `contact_messages`

La commande crée le modèle et sa migration ; la migration reçoit trois colonnes, le modèle les déclare remplissables.

? Un visiteur envoie le formulaire vide, ou tape pas-une-adresse. Où ces refus sont-ils décidés ?

ContactMessageRequest porte les règles ; attributes() donne au champ name son nom français : « nom ».

app/Http/Requests/ContactMessageRequest.php — extrait

```php
return [
    'name' => ['required', 'string', 'max:100'],
    'email' => ['required', 'email', 'max:255'],
    'message' => ['required', 'string', 'max:5000'],
];
```

'email' refuse une adresse mal formée, comme pas-une-adresse

```php
return ['name' => 'nom'];
```

attributes() : sans cette ligne, l'erreur parlerait du champ « name »

« adresse e-mail » et « message » sont déjà traduits depuis le chapitre 1.12

le formulaire envoyé vide



une erreur en français sous chaque champ

À gauche, les règles et le nom français du champ ; à droite, ce que voit le visiteur qui envoie le formulaire vide.

? Un e-mail a un objet, une adresse de réponse et un contenu. Où Laravel les décrit-il ?

Un Mailable est une classe qui décrit un e-mail ; envelope() fixe son objet et l'adresse de réponse, celle du visiteur.

```
$ php artisan make:mail ContactMessageReceived --markdown=mail.contact-message
```

`--markdown` : un template écrit en Markdown, que Laravel transforme en un e-mail mis en page

```
public function __construct(public ContactMessage $contactMessage) {}
```

le constructeur reçoit le message enregistré

app/Mail/ContactMessageReceived.php — extrait

```
return new Envelope(
    replyTo: [new Address($this->contactMessage->email, $this->contactMessage->name)],
    subject: 'Nouveau message de '.$this->contactMessage->name,
);
```

quand vous cliquez sur « Répondre », vous écrivez au visiteur, pas à vous-même

l'objet de l'e-mail

`use Queueable, SerializesModels;` et `ShouldQueue` retirés : ils servent à une file d'attente, une liste de tâches qu'un autre processus exécute plus tard. Le portfolio n'en utilise pas. `attachments()` retirée : pas de pièce jointe.

Le constructeur reçoit le message enregistré ; envelope() en tire l'objet de l'e-mail et l'adresse où répondre.

? Où s'écrit le texte de l'e-mail, et comment le nom du visiteur y arrive-t-il ?

content() désigne le template Markdown : il lit la propriété publique $contactMessage et échappe le texte avec {{ }}.

app/Mail/ContactMessageReceived.php — extrait

```php
public function content(): Content
{
    return new Content(markdown: 'mail.contact-message');
}
```

```php
public ContactMessage $contactMessage
```

propriété publique : disponible dans le template

resources/views/mail/contact-message.blade.php

```
<x-mail::message>
# Nouveau message depuis le portfolio

**De :** {{ $contactMessage->name }} ({{ $contactMessage->email }})

{{ $contactMessage->message }}

Répondez directement à cet e-mail pour lui écrire.
</x-mail::message>
```

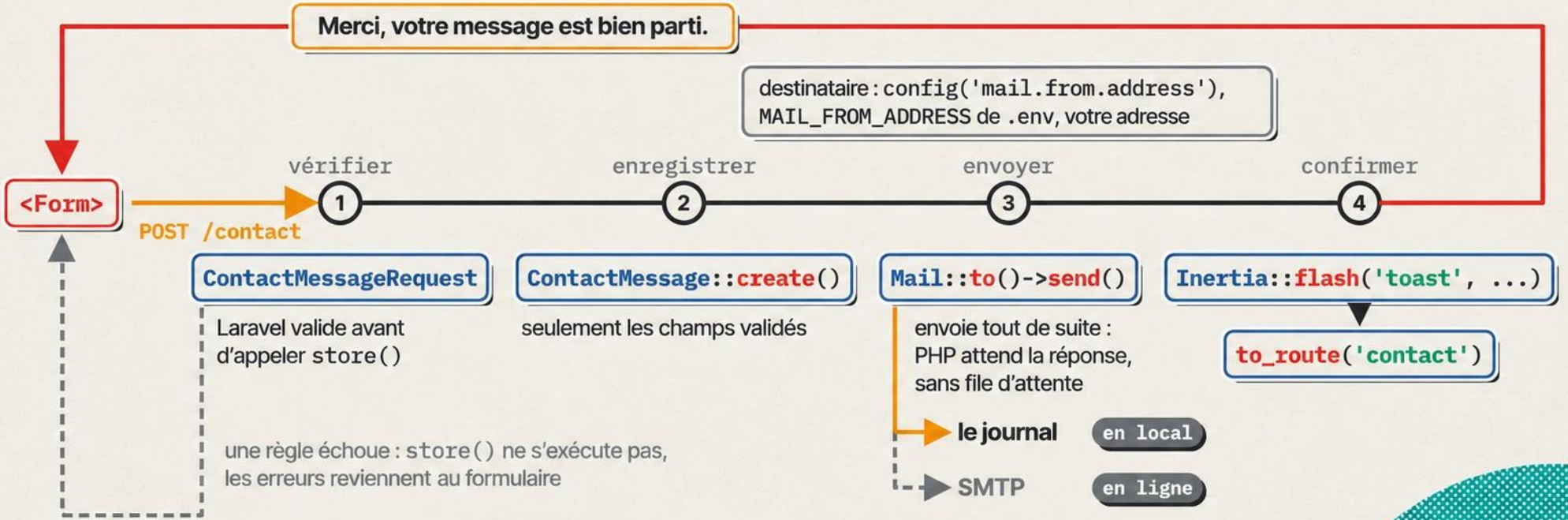
- `<x-mail::message>` : le composant de Laravel : en-tête, mise en page et pied de l'e-mail
- `# Nouveau message depuis le portfolio` et `**De :**` : un titre et du gras, comme en Markdown
- `{{ $contactMessage->message }}` : {{ }} échappe le texte du visiteur, comme dans toute vue Blade

Suivez le nom du template, de la classe au fichier ; la propriété publique du constructeur s'y lit entre {{ }}.

1.20

? Le visiteur clique sur « Envoyer ». Dans quel ordre store() traite-t-il son message ?

Laravel valide avant store(), qui enregistre, envoie l'e-mail tout de suite et redirige vers le formulaire.



Les quatre stations de la slide 1 portent maintenant leur nom dans Laravel.
Une règle qui échoue renvoie le visiteur au formulaire.

1.20

? Le formulaire s'affiche et se reçoit à la même adresse. Comment Laravel les distingue-t-il ?

GET /contact mène à create(), POST /contact à store() ; le nom de route contact ne change pas.

~~`Route::inertia('contact', 'Contact')`~~ la ligne du chapitre 1.08, remplacée

routes/web.php

```php
Route::get('contact', [ContactController::class, 'create'])->name('contact');
Route::post('contact', [ContactController::class, 'store'])->name('contact.store');
```

le nom ne change pas : le lien du menu fonctionne toujours



Même adresse, deux routes : GET /contact mène à create(), POST /contact mène à store().

1.20

Après l'envoi, Laravel renvoie la même page. Comment le visiteur sait-il que c'est parti ?

:reset-on-success vide les trois champs, et le message flash s'affiche grâce au <Toaster /> ajouté au layout public.

Wayfinder : l'adresse /contact et la méthode post

resources/js/pages/Contact.vue — extrait

```
<Form
    v-bind="ContactController.store.form()"
    :reset-on-success="['name', 'email', 'message']"
    v-slot="{ errors, processing }"
    class="space-y-6"
>
```

SANS :RESET-ON-SUCCESS

le message resterait dans les champs, et le visiteur ne saurait pas s'il est parti

après un envoi réussi

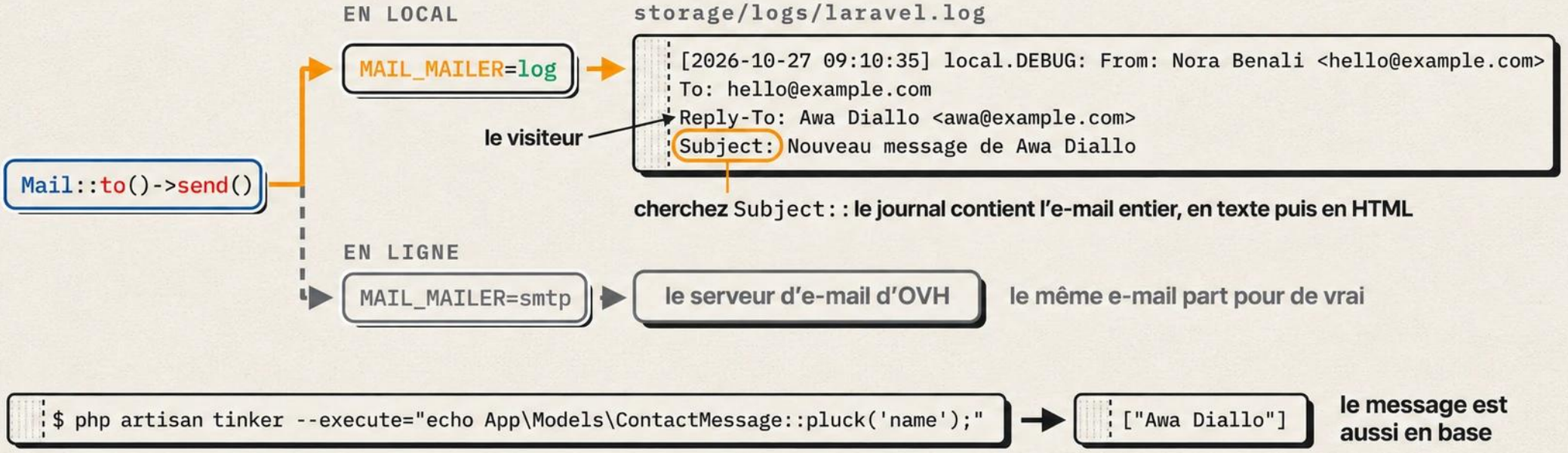


<Toaster /> ajouté au layout public au chapitre 1.18 : il affiche le message flash

À gauche, l'option qui vide les champs ; à droite, la page après l'envoi, avec le message de confirmation en bas.

? En local, aucun e-mail ne doit partir pour de vrai. Où Laravel met-il celui du visiteur ?

En local, MAIL_MAILER=log écrit l'e-mail dans laravel.log ; en ligne, smtp le fera partir pour de vrai.




```
$ php artisan tinker --execute="echo App\Models\ContactMessage::pluck('name');"
```

→

```
["Awa Diallo"]
```

le message est aussi en base

Le même appel, deux destinations selon MAIL_MAILER : le journal en local, le serveur d'e-mail d'OVH en ligne.

? Un robot envoie votre formulaire des centaines de fois. Comment le freiner, et que voit-il ?

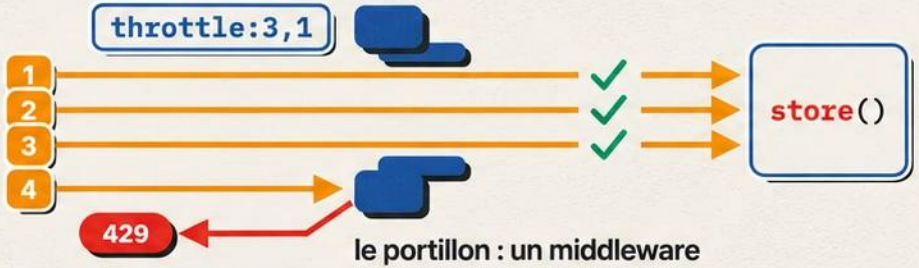
throttle:3,1 accepte trois envois par minute et par adresse IP ; le quatrième reçoit 429, dans la page d'erreur du site.

routes/web.php

```php
Route::post('contact', [ContactController::class, 'store'])
    ->middleware('throttle:3,1')
    ->name('contact.store');
```

quatre envois dans la même minute

les envois refusés par la validation comptent aussi



le portillon : un middleware

SANS RIEN DE PLUS

la page d'erreur de Laravel, « 429 | Trop de requêtes », dans une fenêtre posée par-dessus le site

AVEC 429 DANS LA LISTE

app/Providers/AppServiceProvider.php — extrait

```php
if (in_array($response->statusCode(), [403, 404, 429])) {
```

ErrorPage.vue : une table des titres et une table des messages, un par code

le quatrième envoi, dans votre layout public

Nora Benali

429 · Trop d'envois

Réessayez dans une minute.

Retour à l'accueil

En haut, le portillon compte les envois ; en bas, ce que voit le quatrième, avant et après la ligne d'AppServiceProvider.

? Comment un test vérifie-t-il l'e-mail sans qu'il parte ni s'écrive dans le journal ?

Mail::fake() garde les e-mails en mémoire ; assertSent() vérifie la classe envoyée et son adresse de réponse.



tests/Feature/ContactTest.php

```
Mail::fake();
```

première ligne de chaque test

```
Mail::assertSent(ContactMessageReceived::class, fn ($mail) => $mail->hasReplyTo('awa@example.com'));
```

un e-mail de cette classe est parti, avec la bonne adresse de réponse

```
Mail::assertNothingSent();
```

le formulaire refusé : aucun e-mail

```
 PASS  Tests\Feature\ContactTest
✓ a visitor can send a message
✓ the contact form is validated
✓ the contact form is throttled

Tests:    3 passed (21 assertions)
```

Le trajet est celui de la slide 6 ; seul l'envoi change : le test interroge les e-mails gardés en mémoire.

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

À vous, vingt minutes : un formulaire de contact qui valide, enregistre, vous envoie l'e-mail et confirme l'envoi.

20 min

À FAIRE

1. Le modèle `ContactMessage`, sa migration, `php artisan migrate`.
2. `ContactMessageRequest` : `rules()` et `attributes()`.
3. Le Mailable `ContactMessageReceived` et son template Markdown.
4. `ContactController`, ses deux routes, et `Contact.vue`.
5. Un envoi depuis `portfolio.test/contact`, puis `laravel.log`.

À VÉRIFIER

- [ ] Envoyé vide, le formulaire affiche une erreur en français sous chaque champ.
- [ ] Une adresse comme `pas-une-adresse` est refusée.
- [ ] Après un envoi valide, les champs se vident et le message de confirmation s'affiche.
- [ ] Tinker trouve le message dans la table `contact_messages`.
- [ ] `storage/logs/laravel.log` contient l'e-mail, avec `Subject: Nouveau message` de suivi du nom et le `Reply-To` du visiteur.
- [ ] Les deux premiers tests de `ContactTest.php` sont verts, et `php artisan test` aussi.
- [ ] Le quatrième envoi dans la minute affiche la page « 429 · Trop d'envois » du site. **bonus**

Cinq étapes à gauche, sept vérifications à droite ; la dernière n'est demandée qu'à ceux qui font le bonus.