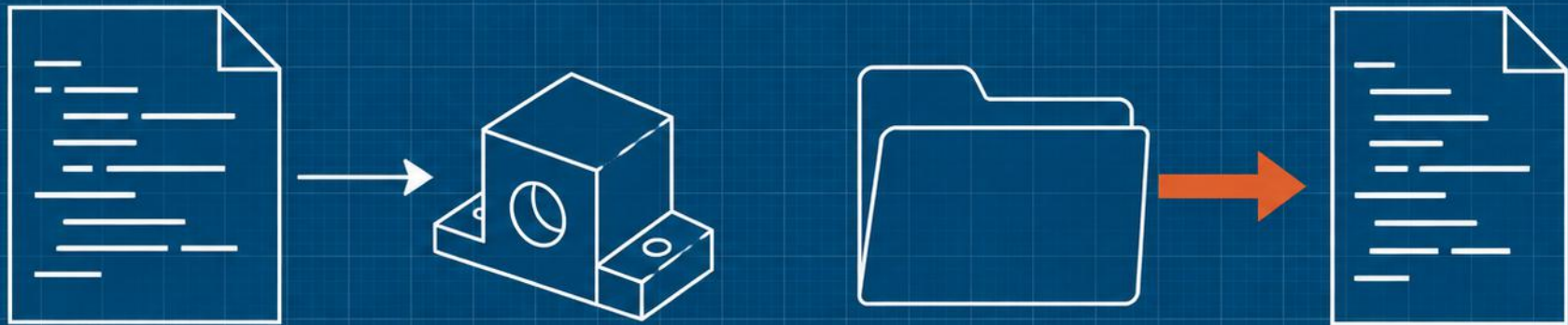


Bloc Laravel · chapitre 1

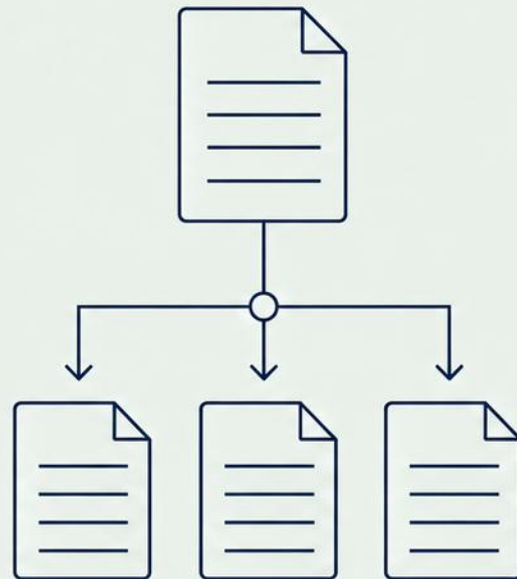
Framework et librairie

Une bibliothèque fournit des fonctions que votre programme utilise à des endroits précis.
Un framework contient le programme de démarrage de l'application.



« Quelle URL a été demandée ? » : le routage écrit à la main

```
8 // 2. Le routage : quelle URL a été demandée ?
9 $path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
10
11 if ($path === '/posts') {
12     $posts = $pdo->query('SELECT title FROM posts ORDER BY id DESC')->fetchAll();
13     echo '<h1>Articles</h1>', PHP_EOL;
14     foreach ($posts as $post) {
15         // 3. L'échappement HTML
16         echo '<p>', htmlspecialchars($post['title']), '</p>', PHP_EOL;
17     }
18 } else {
19     http_response_code(404);
20     echo 'Page introuvable', PHP_EOL;
21 }
```



Cette petite application de blog tient dans index.php. Au-dessus de cet extrait, trois instructions créent une base de données en mémoire avec deux articles.

Le routage associe l'adresse demandée au code qui doit produire la réponse. Ici, la condition `if ($path === '/posts')` réalise cette association.

Le fichier fonctionne tant que le site ne contient qu'une page

```
1 php -S 127.0.0.1:8080 index.php
```

```
1 $ curl http://127.0.0.1:8080/posts
2 <h1>Articles</h1>
3 <p>&lt;script>alert(1)&lt;/script></p>
4 <p>Bienvenue sur le blog</p>
```

```
5 $ curl -i http://127.0.0.1:8080/contact
6 HTTP/1.1 404 Not Found
7 Host: 127.0.0.1:8080
8 Date: Sat, 26 Sep 2026 14:18:07 GMT
9 Connection: close
10 X-Powered-By: PHP/8.4.21
11 Content-type: text/html; charset=UTF-8
12
13 Page introuvable
14
15
```

○○○

L'échappement transforme `<script>` en `<script>`. Le navigateur affiche donc ce texte au lieu de l'exécuter comme une balise.

Avec vingt pages, la condition devient une suite de `elseif` à modifier pour chaque nouvelle adresse.

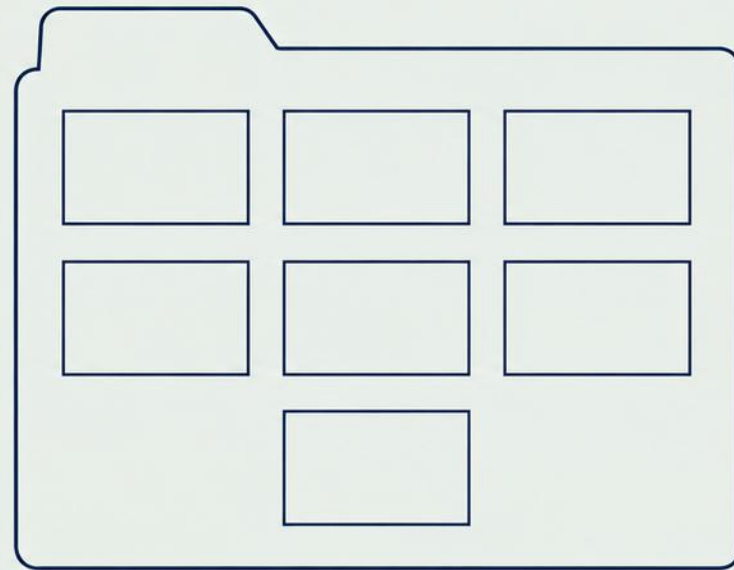
Une session enregistre sur le serveur l'identité de la personne connectée.

Vous devez aussi refuser les formulaires envoyés depuis un autre site et préparer l'envoi des courriels. Sans outil commun, vous remettez en place ces mécanismes dans chaque projet.

Une bibliothèque fournit du code que votre programme utilise

```
composer require nesbot/carbon
```

```
./composer.json has been created
Running composer update nesbot/carbon
Loading composer repositories with package information
Updating dependencies
Lock file operations: 7 installs, 0 updates, 0 removals
  - Locking carbonphp/carbon-doctrine-types (3.2.1)
  - Locking nesbot/carbon (3.14.0)
...
Generating autoload files
6 packages you are using are looking for funding.
Use the composer fund command to find out more!
No security vulnerability advisories found.
Using version ^3.14 for nesbot/carbon
```



Une bibliothèque rassemble des classes et des fonctions écrites par d'autres personnes. Votre programme utilise seulement celles dont il a besoin.

Le mot « librairie » est aussi employé, par calque de l'anglais *library*.

Le signe ... remplace les lignes intermédiaires. Composer installe sept paquets, car Carbon en utilise cinq et l'un de ces paquets en utilise un autre.

Un paquet nécessaire au fonctionnement d'un programme s'appelle une dépendance.

Avec une bibliothèque, c'est votre code qui appelle

```
1 <?php
2
3 require __DIR__ . '/vendor/autoload.php';
4
5 use Carbon\Carbon;
6
7 $publishedAt = Carbon::parse('2026-09-23 09:00');
8
9 echo $publishedAt->locale('fr_BE')->isoFormat('dddd D MMMM YYYY'), PHP_EOL;
10 echo $publishedAt->locale('fr_BE')->diffForHumans(), PHP_EOL;
```

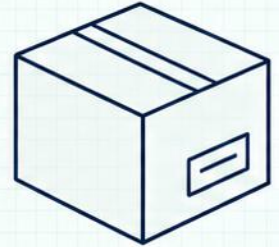
```
$ php dates.php
```

```
mercredi 23 septembre 2026
```

```
il y a 3 jours
```



dates.php



Carbon

PHP exécute dates.php de haut en bas. À la ligne `Carbon::parse()`, PHP utilise la méthode fournie par Carbon et place la date obtenue dans `$publishedAt`.

Les deux lignes suivantes utilisent encore Carbon pour mettre cette date en forme. Si vous retirez ces appels, aucune méthode de Carbon n'est exécutée.

Une bibliothèque fournit donc du code que votre programme utilise aux endroits choisis. La seconde ligne change selon le jour où vous lancez le script.

Votre projet ne contient ni new PostController() ni ->index()

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Models\Post;
6 use Illuminate\View\View;
7
8 class PostController extends Controller
9 {
10     public function index(): View
11     {
12         $posts = Post::latest()->get();
13
14         return view('posts.index', ['posts' => $posts]);
15     }
16 }
```

```
$ curl http://premier-test-laravel.test/posts
```

```
<h1>Articles</h1>
<p>&lt;script&gt;alert(1)&lt;/script&gt;</p>
<p>Bienvenue sur le blog</p>
```

Un contrôleur est une classe dont les méthodes préparent la réponse à une adresse. Vous avez déjà rencontré ce contrôleur dans le bloc POO.

Aucun fichier de votre projet ne contient new PostController() suivi de ->index().

Le modèle Post est une classe qui représente les articles enregistrés dans la table posts. Une vue est un fichier qui produit le HTML envoyé au navigateur. Dans cette vue, les doubles accolades {{ }} échappent chaque titre.

La réponse contient le même HTML qu'avec index.php. Votre code ne contient pourtant ni condition sur l'adresse ni création d'un objet PDO.



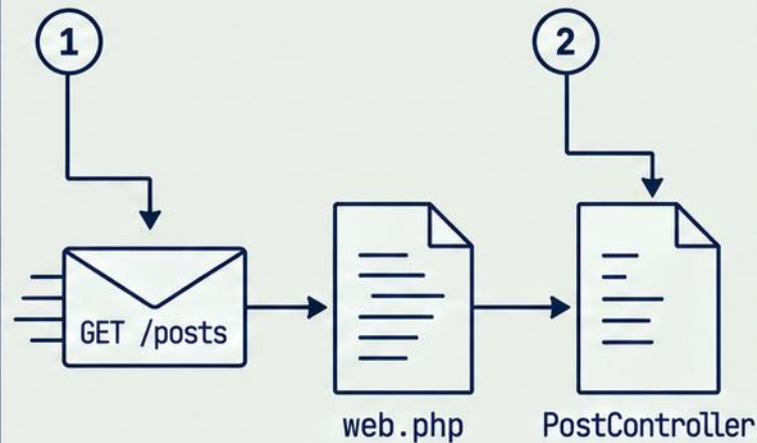
La route associe /posts à la méthode index()

```
1 use App\Http\Controllers\PostController;  
2  
3 Route::get('/posts', [PostController::class, 'index']);  
4  
5
```

```
$ php artisan route:list --path=posts
```

```
GET|HEAD      posts ..... PostController@index
```

```
Showing [1] routes
```



Quand Laravel reçoit une requête pour /posts, il cherche ce chemin dans les routes enregistrées. La route contient `PostController::class` et `index`.

Laravel crée donc un objet `PostController`, exécute sa méthode `index()` et transforme la valeur renvoyée en réponse pour le navigateur.

run() cherche une route, crée le contrôleur et appelle sa méthode

```
1  <?php
2
3  // Le « framework » : écrit une fois, par quelqu'un d'autre
4  function run(array $routes, string $path): void
5  {
6      if (!isset($routes[$path])) {
7          echo 'Page introuvable', PHP_EOL;
8          return;
9      }
10
11     [$class, $method] = $routes[$path];
12     $controller = new $class();
13     echo $controller->$method(), PHP_EOL;
14 }
```

framework.php



écrit une fois

[\$class, \$method] copie les deux valeurs de la route dans deux variables.

new \$class() crée un objet de la classe nommée dans \$class. \$controller->\$method() exécute sur cet objet la méthode nommée dans \$method.

Si le chemin ne figure pas dans \$routes, elle affiche Page introuvable et s'arrête.

app.php contient le contrôleur et la liste des routes

```
1 <?php
2
3 require __DIR__ . '/framework.php';
4
5 // Votre code : une classe, une méthode, une table des routes
6 class PostController
7 {
8     public function index(): string
9     {
10         return 'Liste des articles';
11     }
12 }
13
14 $routes = ['/posts' => [PostController::class, 'index']];
15
16 run($routes, $argv[1]);
```

```
$ php app.php /posts
Liste des articles

$ php app.php /contact
Page introuvable
```

app.php

```
<?php

class PostController
{
    public function index(): string
    {
        return 'Liste des articles';
    }
}

$routes = ['/posts' =>
    [PostController::class, 'index']];

run($routes, $argv[1]);
```

framework.php

```
<?php

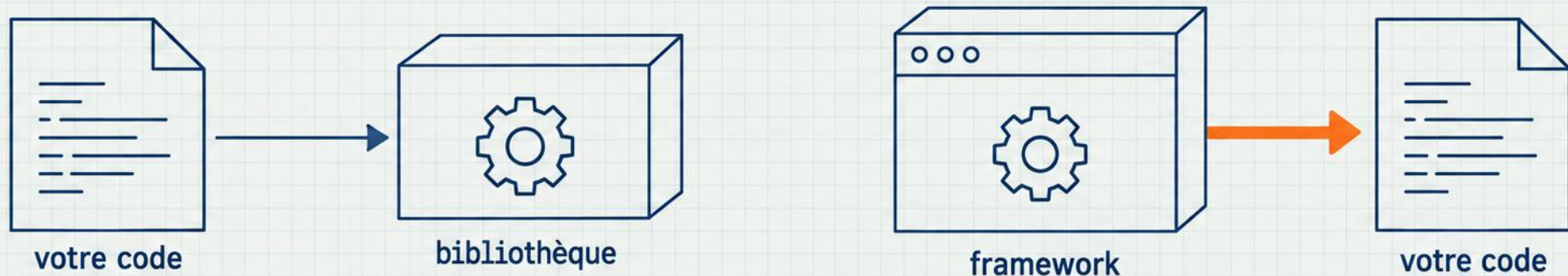
function run(array $routes,
    string $uri): void
{
    // ...
}
```

\$argv[1] contient le premier argument écrit après **php app.php**.

Votre fichier déclare **PostController**, sa méthode **index()** et la route associée à **/posts**. La fonction **run()** reçoit ensuite le chemin demandé.

Pour **/posts**, **run()** crée un **PostController** et appelle **index()**. L'instruction qui lance la méthode se trouve donc dans **framework.php**.

Vous appelez la bibliothèque. Le framework vous appelle.



1. Avec une bibliothèque, votre script contient une instruction que vous avez écrite, comme `Carbon::parse()`.
2. Un framework contient le programme de démarrage de l'application. Quand le navigateur demande une page, ce programme lit l'adresse et exécute la méthode enregistrée.
3. Vous écrivez la route et la méthode, mais pas l'instruction qui **appelle** `index()`. Cette organisation s'appelle l'inversion de contrôle.
4. À l'intérieur de `index()`, votre code utilise ensuite des fonctions de Laravel, comme `view()`, de la même manière qu'il utiliserait une bibliothèque.
5. En 1985, Richard Sweet décrit ce renversement par une formule de Hollywood :
« Ne nous appelez pas, c'est nous qui vous appellerons ».

Les conventions fixent les noms et les dossiers habituels

Les conventions sont des règles qui fixent à l'avance les noms et les dossiers habituels.

Laravel place les contrôleurs dans `app/Http/Controllers`, les modèles dans `app/Models` et les vues dans `resources/views`. Il associe le modèle `Post` à la table `posts` avec une règle de pluriel anglais.

Le framework fournit les mécanismes communs. Ses classes gèrent notamment le routage, les sessions, la protection des formulaires et l'envoi de courriels.

Les projets emploient les mêmes dossiers. Dans un autre projet Laravel, vous chercherez encore les contrôleurs dans `app/Http/Controllers`.

Certaines protections s'appliquent automatiquement. Les doubles accolades de la vue échappent le HTML sans appel direct à `htmlspecialchars()`.



`app/Http/Controllers`



`app/Models`



`resources/views`

`posts`

Ce que vous perdez avec un framework

Le déroulement est plus difficile à suivre.

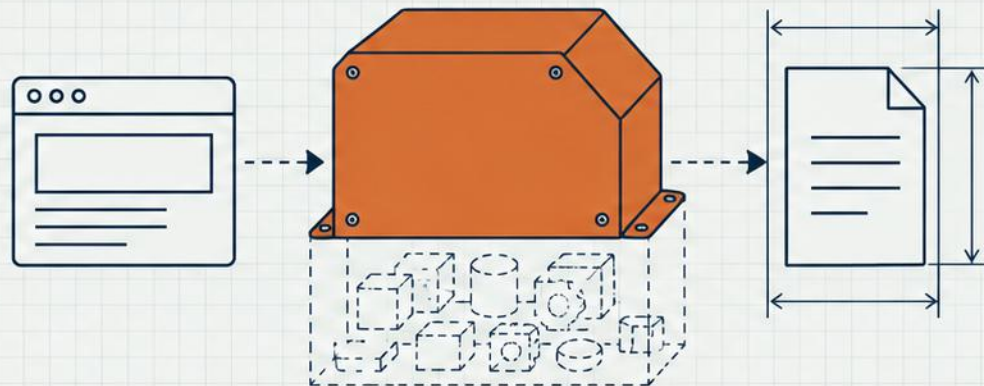
Dans `index.php`, vous lisez toutes les instructions dans leur ordre. Pour afficher `/posts`, Laravel charge environ trois cents classes, dont quatre seulement se trouvent dans vos fichiers.

Votre code doit respecter les conventions.

Si le nom ou le dossier d'un fichier ne correspond pas à la règle attendue, Laravel ne charge pas l'élément concerné. Le message d'erreur n'indique pas toujours quelle convention a été enfreinte.

Le projet doit suivre les versions du framework.

Laravel publie une version majeure par an et fournit les correctifs de sécurité pendant deux ans. Vous devez donc adapter le projet à une nouvelle version au moins tous les deux ans.



Plusieurs langages proposent des frameworks web

Côté serveur, ils reçoivent la requête HTTP et produisent la réponse.

Nom	Langage	Depuis
Laravel	PHP	2011
Symfony	PHP	2005
Django	Python	2005
Ruby on Rails	Ruby	2004
Express	JavaScript (Node.js)	2010
Next.js	JavaScript (Node.js)	2016
Spring	Java	2002
ASP.NET Core	C#	2016

Laravel fournit de nombreux réglages par défaut.

Ruby on Rails a popularisé les noms et les dossiers fixés par défaut plutôt que leur configuration séparée.

React se présente comme une bibliothèque

Un **composant** est une partie d'interface réutilisable, comme un bouton ou une carte d'article. Dans le navigateur, ces outils exécutent le code des composants pour construire l'interface.

Nom	Langage	Depuis
React	JavaScript	2013
Vue	JavaScript	2014
Angular	TypeScript	2016
Svelte	JavaScript	2016

React exécute le code de vos composants, mais il n'organise pas à lui seul toute l'application. Next.js ajoute les routes, le démarrage du serveur et des règles pour organiser les fichiers. Il remplit donc le rôle d'un framework.

Laravel utilise aussi Carbon, onze paquets Symfony et trois **polyfills**. Ces paquets apportent à une ancienne version de PHP une fonction plus récente.

À vous : qui appelle qui ?

1. **Bibliothèque ou framework ?** Sur papier, classez huit outils, de Carbon à React. Pour chacun, nommez le code exécuté et l'instruction qui déclenche son exécution.
2. **Qui écrit, qui appelle ?** Pour cinq instructions de `PostController` et de `routes/web.php`, indiquez qui les écrit. Indiquez ensuite quel programme les exécute et à quel moment.
3. **Cinq questions de révision,** en cartes à retourner sur le site du cours.

