

# CAWEBI - LARAVEL



{{ 3 - Laravel }}

# Ce que l'on a vu

---

## Partie 1

1. Syntaxe PHP
2. Concept client-serveur
3. Envoi de données en GET et POST
4. Persistance des données avec les cookies et les sessions

## Partie 2

1. Fonctions
2. PHP Objet (`$this`, `__construct`)
3. Mécanisme d'exceptions : `try` / `catch` / `throw`
4. SQLite & PDO : requêtes préparées
5. Design MVC pour une application web

# Programme

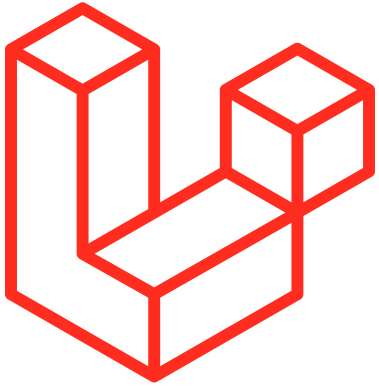
---

1. Le framework
2. Le **routage**
3. Les **contrôleurs**
4. Les **vues** avec le moteur de rendu **Blade**
5. Les **middlewares**

# Le framework Laravel

# Laravel, c'est quoi ?

---

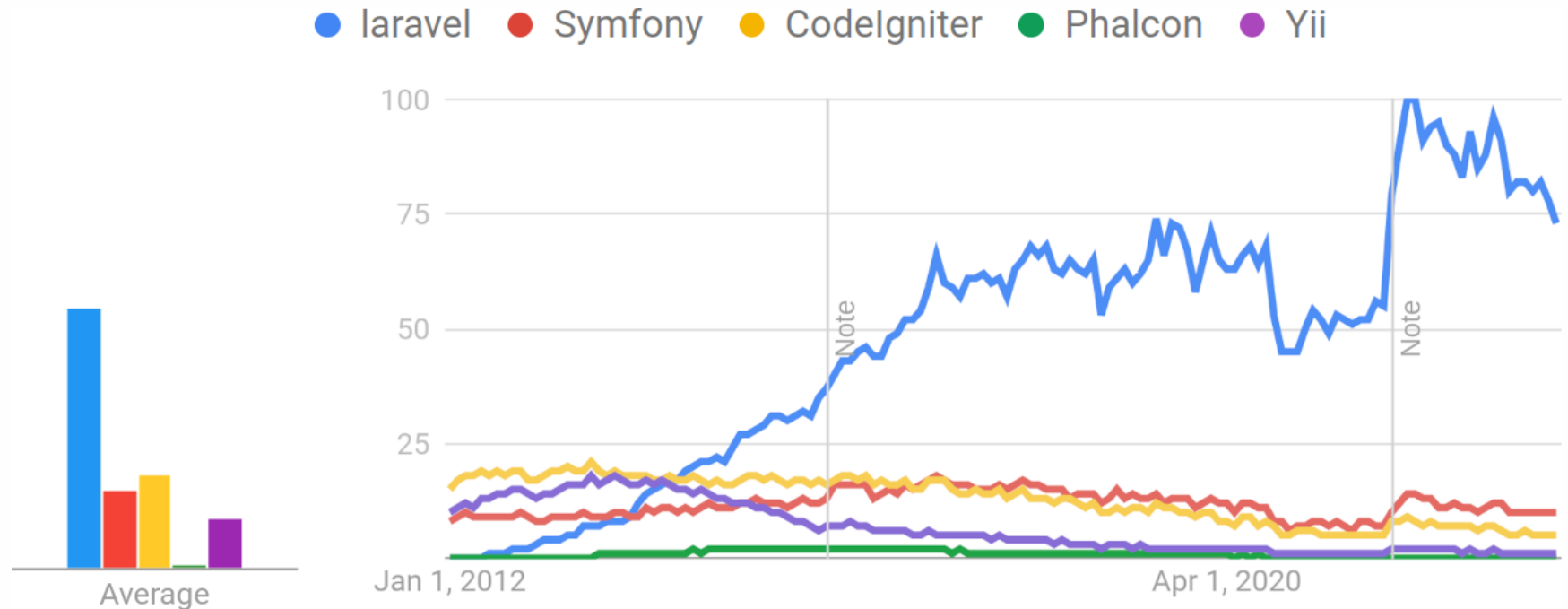


# Laravel

Laravel, c'est un framework PHP, c'est-à-dire un environnement qui facilite la création de sites et services web.

Il en existe d'autres : [Symfony](#), [CodIgniter](#), [Phalcon](#), [Yii](#), etc.

# Pourquoi choisir Laravel ?



*Recherches Google des principaux framework PHP dans le monde.*

- ✓ Laravel est le framework PHP le plus utilisé
- ✓ Laravel est open-source, sous license MIT
- ✓ Laravel utilise astucieusement plusieurs bibliothèques PHP qui ont fait leurs preuves : symfony, monolog, etc.

# Pré-requis

---

## Version de Laravel

Ce cours se base entièrement sur Laravel 11, version sortie le 14 février 2023.

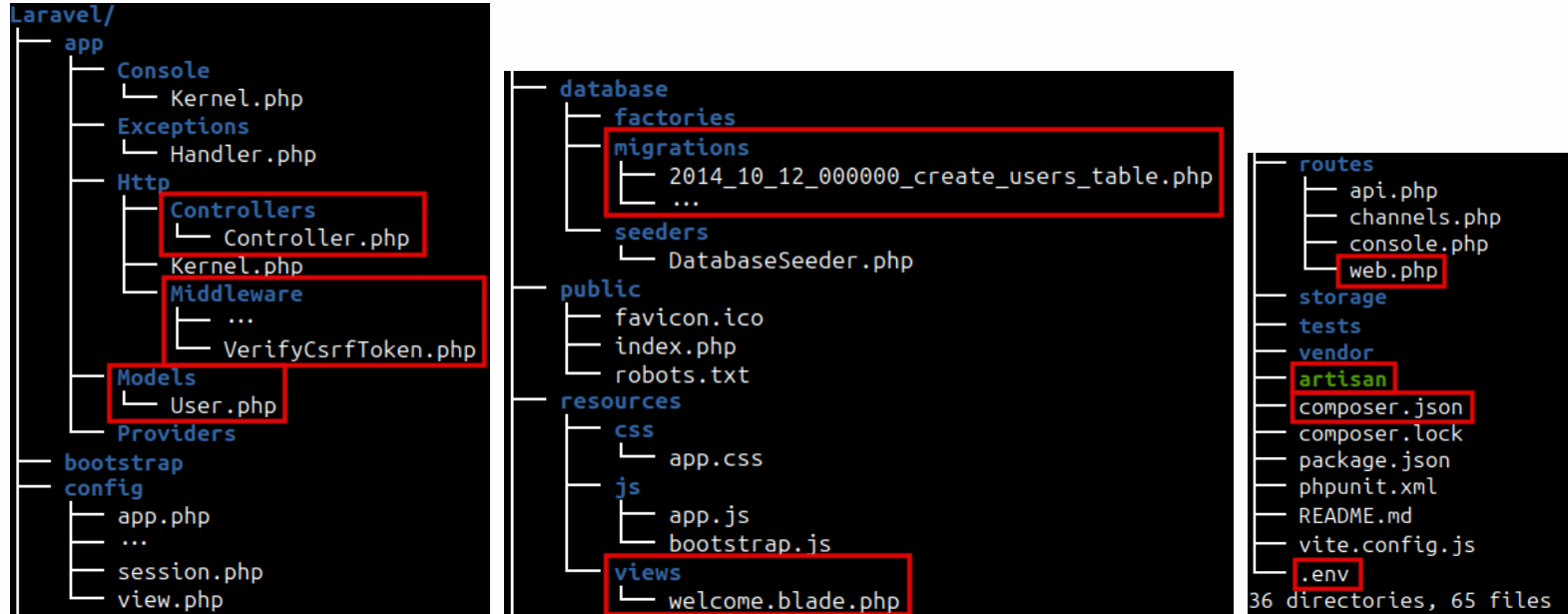


- ✓ Laravel 11 nécessite une version de PHP  $\geq 8.2$
- ✓ Utilisez au minimum Laravel 10 qui nécessite PHP  $\geq 8.1$

## Rappel doc & stackoverflow-like

- ✓ Ne recopiez pas bêtement des solutions proposées par des gens que vous ne connaissez pas
- ✓ La seule aide fiable est la doc officielle sur [php.net](https://php.net)... et [Laravel.com](https://laravel.com).

# L'arborescence de Laravel



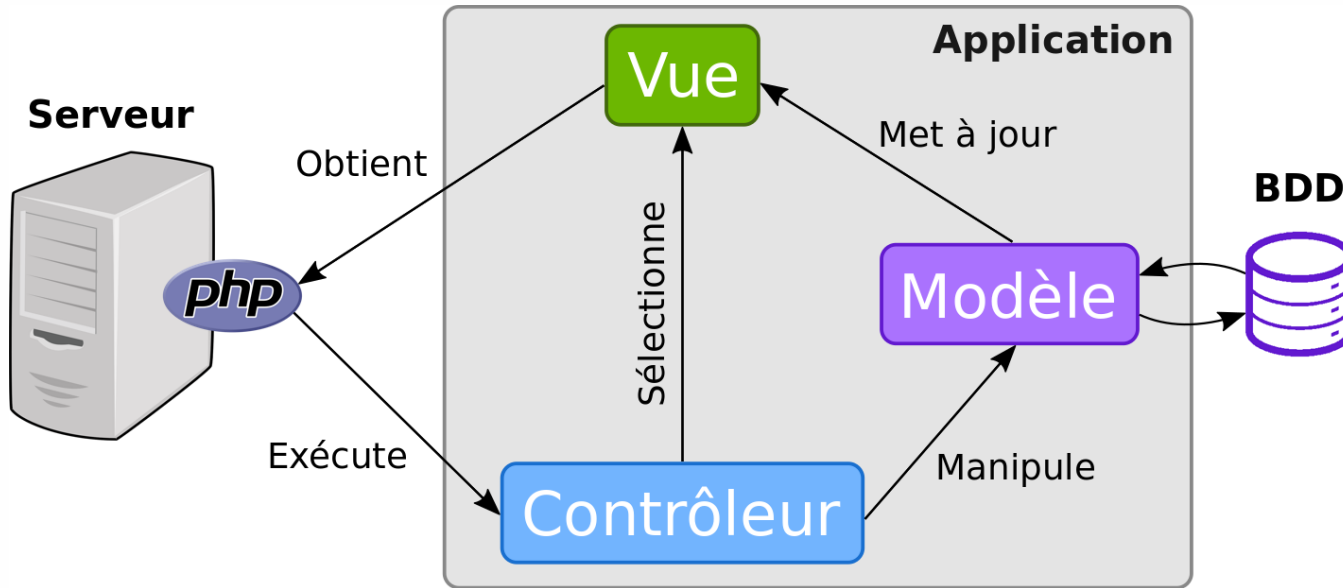
Les fichiers essentiels :

- ✓ `artisan` : script pour la configuration de Laravel
- ✓ `web.php` : fichier de routage pour un site web
- ✓ `composer.json` : liste des dépendances (installées dans `vendor`)
- ✓ `.env` : le fichier de configuration des variables d'environnement, notamment celles de la base de données.



# Laravel est MVC

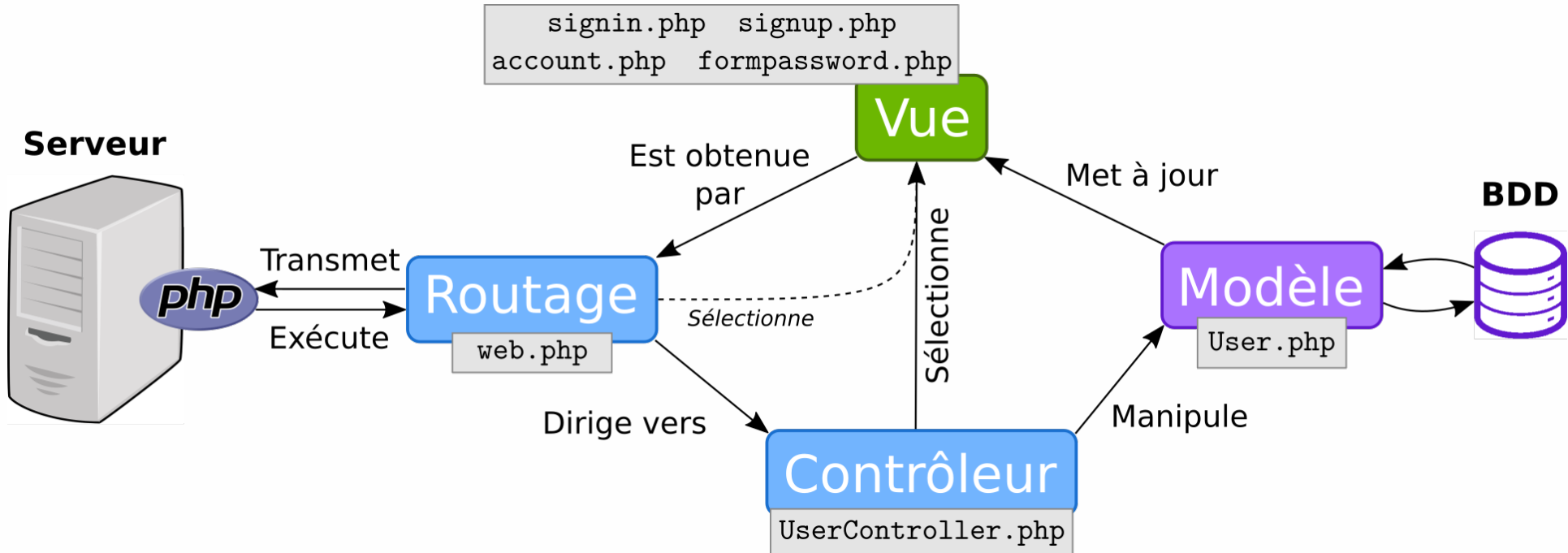
# MVC, encore et toujours



MVC (*Modèle-Vue-Contrôleur*) est une architecture logicielle qui propose un découpage en trois entités :

- ✓ le **Modèle** qui représente les données,
- ✓ la **Vue** qui définit l'affichage du modèle,
- ✓ le **Contrôleur** qui contient la logique de l'application et répond aux requêtes utilisateur en coordonnant les modèles et les vues.

# Le MVC avec Laravel



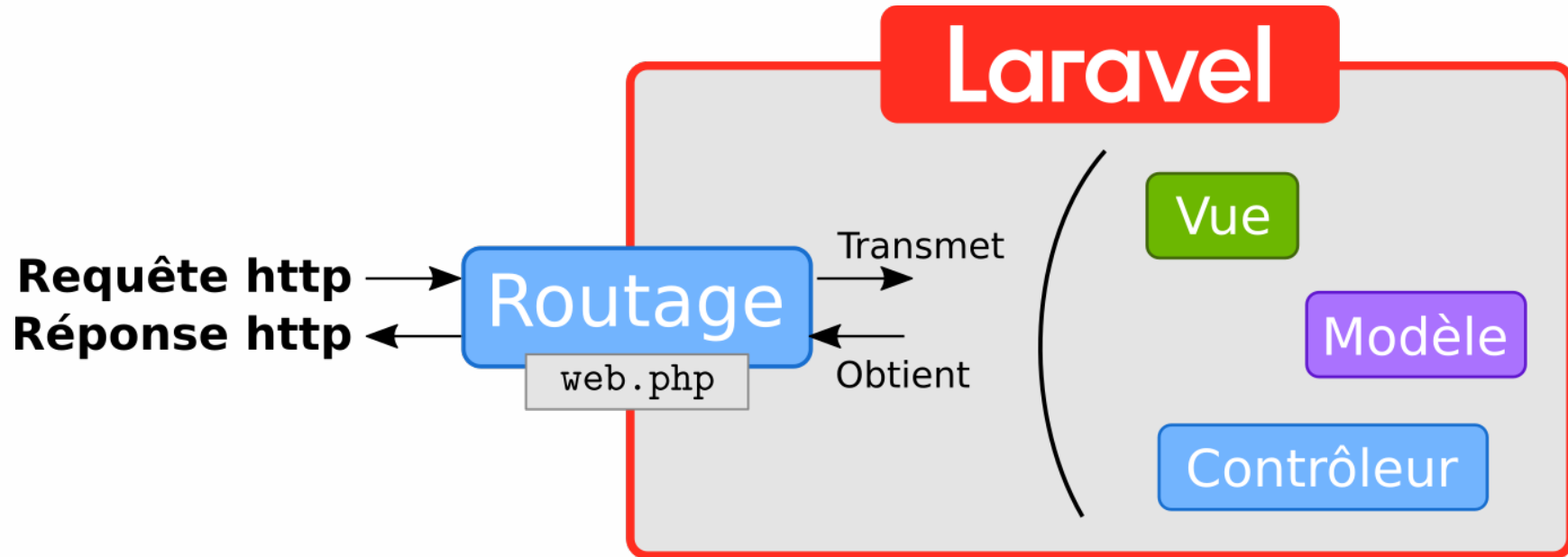
# Vue

- ✓ Les vues contiennent le code HTML des réponses aux requêtes.
- ✓ Elles peuvent contenir du code PHP pour personnaliser l'affichage de chaque client (notamment à travers les sessions).

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

# Le routage

# Qu'est-ce que c'est ?



Le routage est :

- ✓ un mécanisme de contrôle exécuté à la réception d'une requête.
- ✓ l'unique interface entre la requête et le reste de l'application.

# Les routes simples

```
use Illuminate\Support\Facades\Route;

Route::get ( 'index', function() { return "GET"; } );
Route::post( 'index', function() { return "POST"; } );
Route::any ( 'index', function() { return "ANY"; } );
```

*Extrait du fichier* routes/web.php *du site* http://site.fr

Spécifier une route, c'est indiquer la fonction à exécuter à la réception d'une requête HTTP avec une URL et une méthode spécifiques.

- ✓ Le routage est spécifié dans le fichier routes/web.php
- ✓ Le routage est effectué via la facade Route
- ✓ Les routes non spécifiées produiront une réponse http 404.

# Les routes paramétrées

```
Route::get (
    'module/{var1}/{var2}',
    function( string $var1, int $var2 ) : string {
        return "GET avec ".$var1." et ".$var2;
    }
);
```

routes/web.php

http://site.fr/module/M2/WEB

GET avec M2 et WEB

http://site.fr/module/M/W

GET avec M et W

*Routage des URL de la forme* http://site.fr/ module/[text]/[integer]

- ✓ Une route peut déclarer des paramètres avec des accolades.
- ✓ Chaque paramètre déclaré devient un argument de la fonction à exécuter.



# Les groupes de routes préfixés

```
Route::prefix('admin')->group( function () {  
  
    Route::get('/', function() { return "GET admin"; });  
    Route::get('module', function() { return "GET admin/module"; });  
    Route::get('stats', function() { return "GET admin/stats"; });  
  
});
```

routes/web.php

http://site.fr/admin

GET admin

http://site.fr/admin/module

GET admin/module

http://site.fr/admin/stats

GET admin/stats

## Les groupes préfixés

- ✓ permettent d'organiser les routes.
- ✓ définissent un début d'URL commun à toutes les routes du groupe.

# Routage et vues

# Afficher une vue

```
Route::get( 'home', function () {  
    return view('index');  
});
```

```
// Alternative  
Route::view( 'home', 'index' );
```

routes/web.php

```
<!doctype html>  
<html lang="fr">  
    <head>  
        <meta charset="utf-8">  
        <title>Page d'accueil</title>  
    </head>  
    <body>Bonjour !</body>  
</html>
```

resources/views/index.php

*Exemple d'une requête vers* `http://site.fr/home`

Pour ne pas écrire le code HTML au niveau des routes, Laravel propose un mécanisme de vues

- ✓ Les vues doivent être placées dans le répertoire `resources/views/`
- ✓ Le paramètre de la directive `view()` doit être le nom du fichier sans extension

# Transmettre des données à une vue

```
Route::get( 'home', function () {  
    return view('index', ['name'=>'CAWEBI']);  
});
```

```
// Alternative  
Route::view(  
    'home', 'index', ['name'=>'CAWEBI']  
);
```

routes/web.php

```
<!doctype html>  
<html lang="fr">  
    <head>  
        <meta charset="utf-8">  
        <title>Page d'accueil</title>  
    </head>  
    <body>  
        Bonjour <?= $name ?> !  
    </body>  
</html>
```

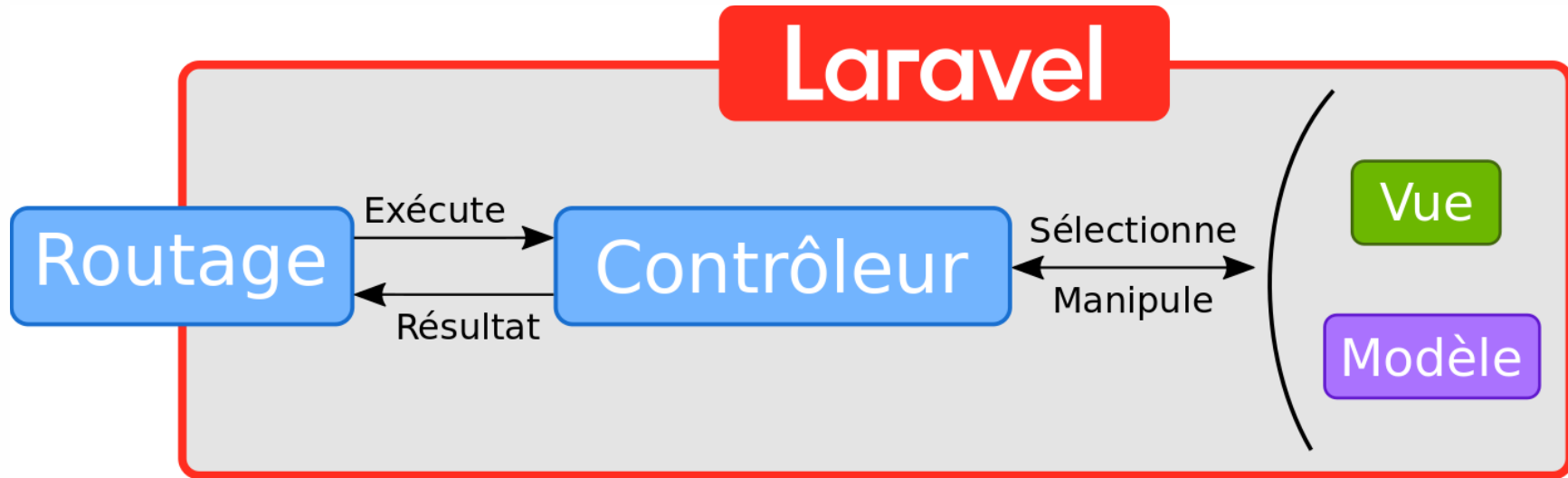
resources/views/index.php

*Exemple d'une requête vers* <http://site.fr/home>

- ✓ Les données sont transmises dans un tableau dont chaque entrée est convertie en variable dans la vue.
- ✓ La méthode `Route::view()` correspond au protocole GET et s'utilise pour afficher une vue sans contrôle.

# Les Contrôleurs

# Qu'est-ce que c'est ?



- ✓ Un contrôleur contient la logique applicative pour répondre aux requêtes.
- ✓ Un contrôleur manipule les modèles et les vues avec pour objectif de construire le résultat.
- ✓ Dans Laravel, c'est au niveau du routage qu'est décidé quelle méthode de quel contrôleur appeler.

# Les classes de contrôleurs

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class MyController extends Controller
{
    public function __construct() { ... }
    public function myMethod( Request $request ) { ... }
}
```

app/Http/Controllers/MyController.php

## Un contrôleur

- ✓ est une classe qui étend la classe `Controller` de Laravel
- ✓ doit être placé dans le répertoire `app/Http/Controllers`

**namespace** : espace de noms virtuel du contrôleur (≈ `package` de Java)

**use** : import d'une classe ou d'un namespace (≈ `import` de Java)

# Routage vers un contrôleur

```
Route::get( 'home', [MyController::class, 'show'] );
```

routes/web.php



Bonjour CAWEBI !

http://site.fr/home

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Page d'accueil</title>
  </head>
  <body>
    Bonjour <?= $name ?> !
  </body>
</html>
```

resources/views/index.php



```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class MyController extends Controller
{
    public function show(Request $request) {
        return view('index', ['name'=>'CAWEBI']);
    }
}
```

app/Http/Controllers/MyController.php

Une méthode publique d'un contrôleur peut être utilisée comme fonction à exécuter pour une route avec cette syntaxe :

```
[NomDeLaClasse::class, 'nomDeLaMéthode']
```



# Un bon contrôleur

---

Un contrôleur Laravel bien construit doit :

- ✓ répondre aux normes de la POO,
- ✓ contenir la logique de traitement de requêtes,
- ✓ manipuler un ou plusieurs modèles,
- ✓ ne contenir aucune instruction d'accès direct à une BDD.

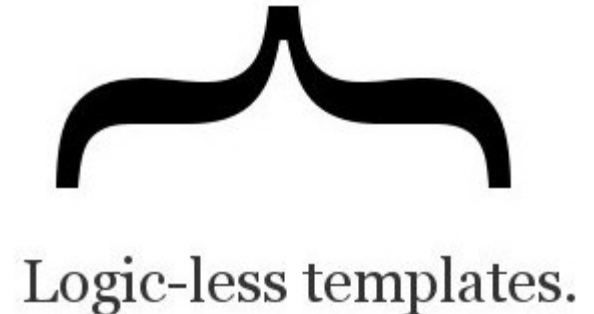
Une règle essentielle :

**Low coupling, High cohesion**

# Blade, le moteur de vues

# Qu'est-ce qu'un moteur de vues ?

---



## Moteur de vues

(*Template Engine en anglais.*) Mécanisme permettant d'écrire des vues de façon plus souple en intégrant par exemple de l'héritage ou des structures de contrôle.

- ✓ C'est un meta-langage qui génère du code pour un langage cible.
- ✓ Il existe une multitude de moteurs de vues comme par exemple **Smarty**, **Twig**, **Mustache**, etc.

# Blade, le moteur de vues de Laravel

---



Laravel utilise son propre moteur de vues : Blade.

## Avantage

Les vues Blade sont pré-compilées en PHP. Il n'y a donc pas de processus supplémentaire lié à Blade. Lorsqu'on accède à une vue, elle aura déjà été transformée en code PHP.

*Note : Les vues Blade doivent avoir pour extension `.blade.php`.*

# Définir un layout

Un layout est une page modèle qui peut être étendue.

```
<html>
  <head><title>@yield('tit
  <body>
    @yield('header')
    @section('content')
      <nav>...</nav>
    @show
  </body>
</html>
```

resources/views/layouts

```
<html>
  <head>
    <title>Mon titre</title>
  </head>
  <body>
    <h1>Ma page</h1>
    <nav>...</nav>
    <p>Contenu de ma page.</p>
  </body>
</html>
```

*Résultat.*

```
layouts.myLayout')
title', 'Mon titre')
header')
page</h1>
n
content')
t
tenu de ma page.</p>
n
```

resources/views/myPage.blade.php

@yield('name')

déclare une section `name` qui sera définie dans une page enfant.

@section('name')

- ✓ (layout) définit une section qui peut être surchargée.
- ✓ (enfant) définit une section qui a été déclarée dans le layout.

# Transmettre des données

```
Route::get('myName', function () {  
    return view('vue', ['name'=>'Bond']);  
});
```

routes/web.php

```
Mon nom est {{ $name }},  
James {{ $name }}.
```

vue.blade.php

Mon nom est Bond,  
James Bond.

http://.../myName

Les données :

- ✓ sont transmises à la vue via un tableau en argument de `view()`
- ✓ sont accessibles dans la vue avec des doubles accolades `{{ }}`.
- ✓ sont protégées des attaques XSS.

Attention : tout se passe côté serveur, entre le contrôleur et la vue.  
Il ne s'agit pas de transmission de données depuis/vers le client.

# Condition If

```
@if ($name === "Bond")
    Mon nom est {{ $name }},
    James {{ $name }}
@elseif ($name === "Pile")
    Mon nom est {{ $name }},
    Jean {{ $name }}.
@else
    Mon nom est {{ $name }}.
@endif
```

vue.blade.php

```
return view('vue', ['name'=>'Bond']);
```

Mon nom est Bond, James Bond.

```
return view('vue', ['name'=>'Pile']);
```

Mon nom est Pile, Jean Pile.

```
return view('vue', ['name'=>'Personne']);
```

Mon nom est Personne.

Les conditions peuvent concerner n'importe quelle variable PHP accessible depuis la vue :

- ✓ des variables transmises par le contrôleur
- ✓ des variables de session
- ✓ ...

# Conditions isset, empty et switch

## Instructions duales `isset` et `empty`

```
@isset( $name )  
    Mon nom est {{ $name }}.  
@endisset  
@empty( $name )  
    Mon nom est Tu-Sais-Qui.  
@endempty
```

vue.blade.php

```
return view('vue', ['name'=>'Bond']);
```

Mon nom est Bond.

```
return view('vue');
```

Mon nom est Tu-Sais-Qui.

## L'instruction `switch`

```
@switch( $name )  
    @case('Bond')  
        Mon nom est {{ $name }},  
        James {{ $name }}.  
    @default  
        Mon nom est Tu-Sais-Qui.  
@endswitch
```

vue.blade.php

```
return view('vue', ['name'=>'Bond']);
```

Mon nom est Bond, James Bond.

```
return view('vue');
```

Mon nom est Tu-Sais-Qui.



# Boucles for et foreach

## La boucle `for`

```
@for( $i=1 ; $i<=$buts ; $i++ )  
    Et {{ $i }}  
@endfor  
Zéro.
```

1998.blade.php

```
return view('1998',['buts' => 3]);
```

Et 1, Et 2, Et 3 Zéro.

## La boucle `foreach`

```
@foreach( $buts as $i )  
    Et {{ $i }},  
@endforeach  
Zéro.
```

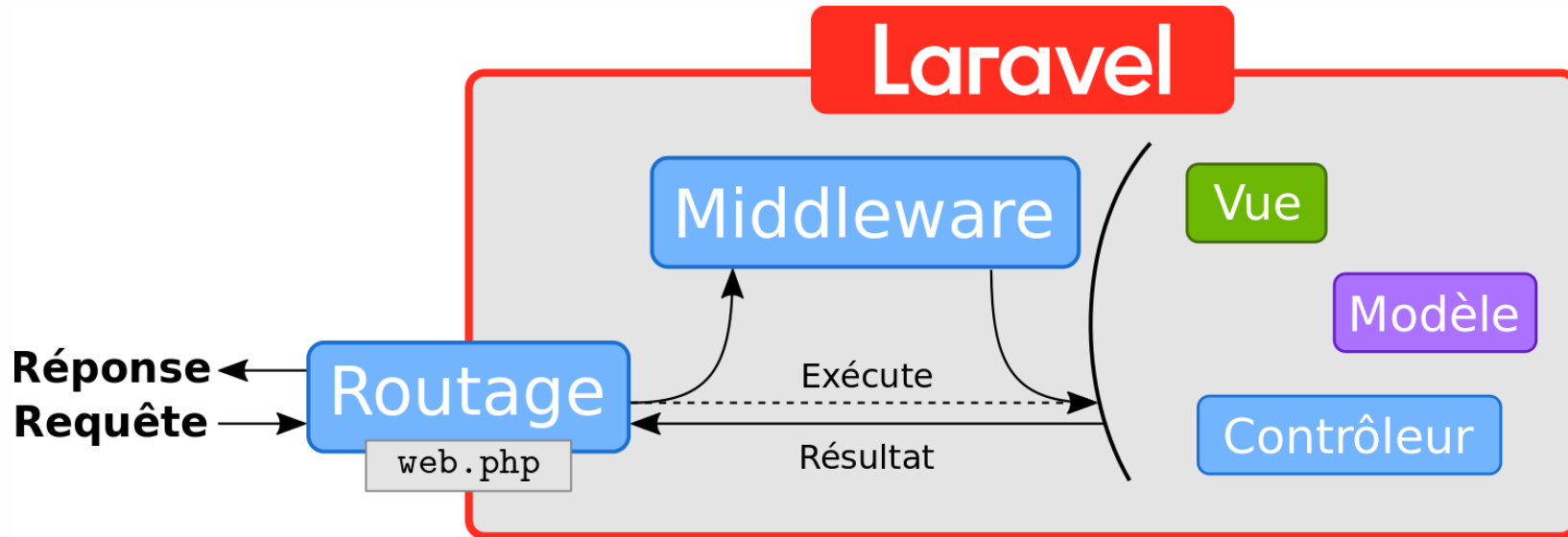
1998v2.blade.php

```
view('1998v2', ['buts' => ['un','deux','trois']]);
```

Et un, Et deux, Et trois Zéro.

# Les Middleware

# Qu'est-ce que c'est ?



- ✓ Dans le contexte de Laravel, un middleware est un contrôleur spécifique dont le but est de filtrer des requêtes
- ✓ Exemples : authentification, journalisation, vérification des données, etc.
- ✓ Laravel permet de mettre en place facilement des middlewares pour une route, un groupe de routes ou un contrôleur.

# Les classes de Middleware

```
namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class MyMiddleware {
    public function handle( Request $request, Closure $next ) {
        // Code à exécuter AVANT le traitement de la requête $request
        $res = $next($request);
        // Code à exécuter APRÈS le traitement de la requête $request
        return $res;
    }
}
```

## Un middleware

- ✓ est une classe sans aucun héritage avec une méthode `handle(Request $request, Closure $next)`
- ✓ décore l'exécution "normale" de requêtes
- ✓ mutualise du code de contrôle récurrent
- ✓ se place dans le répertoire `app/Http/Middleware`

# Middleware global

```
return Application::configure(basePath: dirname(__DIR__))
-> ...
->withMiddleware(function (Middleware $middleware) {
    $middleware->append(\App\Http\Middleware\MyMiddleware::class);
})
```

bootstrap/app.php

- ✓ Pour faire exécuter un middleware pour toutes les requêtes HTTP, on le déclare dans la méthode `withMiddleware` de `bootstrap/app.php`.
- ✓ Le nom à indiquer doit être `\Namespace\De\La\Classe\NomDeLaClasse::class`
- ✓ On peut le placer en tête (`prepend`) ou en fin (`append`) des middlewares globaux

# Middleware sur une route

---

```
use App\Http\Middleware\MyMiddleware;  
  
Route::get('home', ... )->middleware(MyMiddleware::class);
```

ou

```
use App\Http\Middleware\MyMiddleware;  
use App\Http\Middleware\MySecondMiddleware;  
  
Route::get( 'home', ... )->middleware([MyMiddleware::class, MySecondMiddleware::class]);
```

routes/web.php

Un middleware peut être ajouté :

- ✓ aux routes de routes/web.php
- ✓ individuellement ou dans un tableau
- ✓ en paramètre la méthode ->middleware().

# Les groupes de middleware

```
return Application::configure(basePath: dirname(__DIR__))
-> ...
->withMiddleware(function (Middleware $middleware) {
    $middleware->appendToGroup("myMidGroup", [
        \App\Http\Middleware\MyMiddleware::class,
        \App\Http\Middleware\MySecondMiddleware::class
    ]);
})
```

bootstrap/app.php

```
Route::get( 'home', ... )->middleware('myMidGroup');
Route::middleware(['myMidGroup'])->group( ... );
```

routes/web.php

## Un groupe de middleware

- ✓ se déclare dans la méthode `withMiddleware` de `bootstrap/app.php` avec la méthode `appendToGroup`,
- ✓ s'utilise dans `routes/web.php`,
- ✓ permet d'associer plusieurs middlewares d'un coup à une route ou un groupe de routes.

# Combo : middleware sur route préfixée

```
Route::prefix('admin')->middleware('myMidGroup')->group(  
    function () {  
        Route::get ('home', ... );  
        Route::post( ... );  
    }  
);
```

routes/web.php

Associer un middleware à un groupe préfixé permet d'organiser ses routes par préfixe avec une sémantique claire.

C'est à vous, développeurs, d'utiliser à bon escient les préfixes et middlewares pour organiser et mutualiser votre code.