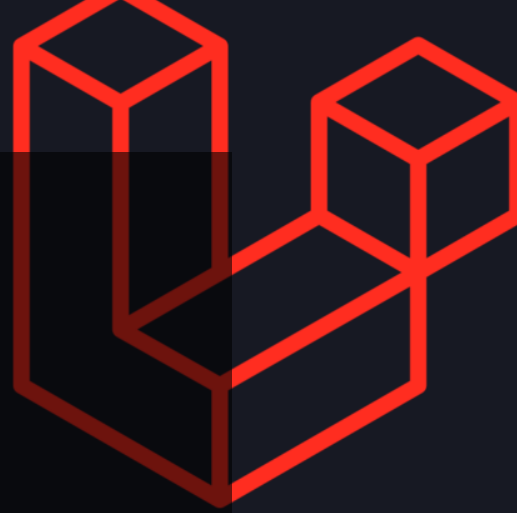


Développer en back-end



Manipulation des contrôleurs

Formatrice : Elidrissi Asmae



Plan du cour

Qu'est-ce qu'un contrôleur ? 01

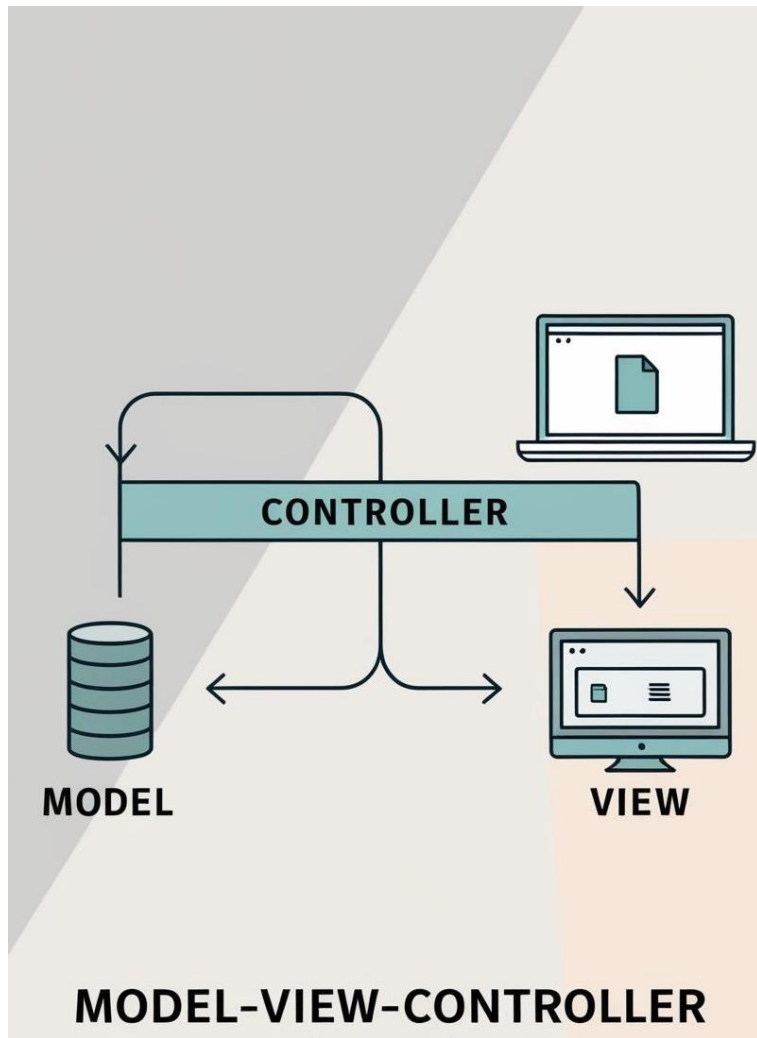
Intérêt des contrôleurs 02

Créer un contrôleur sous Laravel 03

Implémentation de contrôleur 04

Contrôleur de ressources 05

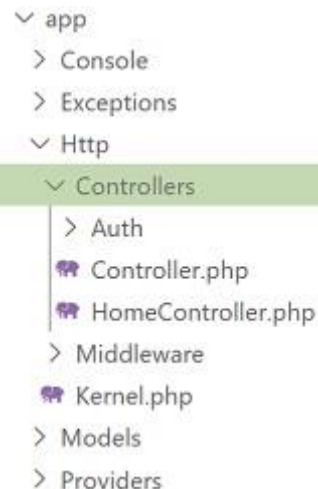




Qu'est-ce qu'un contrôleur ?

Qu'est-ce qu'un contrôleur ?

- Dans le modèle **MVC** (modèle-vue-contrôleur), le contrôleur contient la logique concernant les actions effectuées par l'utilisateur.
 - En pratique, dans une application **Laravel**, l'utilisation de contrôleurs permet **de libérer les routes** du code qu'elles contiennent dans leurs fonctions de rappel.
 - Un **contrôleur** est matérialisé par une classe et chacune de ses méthodes représente une action. Une action correspond généralement à une route.
 - Les contrôleurs sont destinés à regrouper la logique de traitement des demandes associée dans une seule classe.
- Dans votre projet Laravel, ils sont stockés dans le répertoire **app/Http/Controllers**.

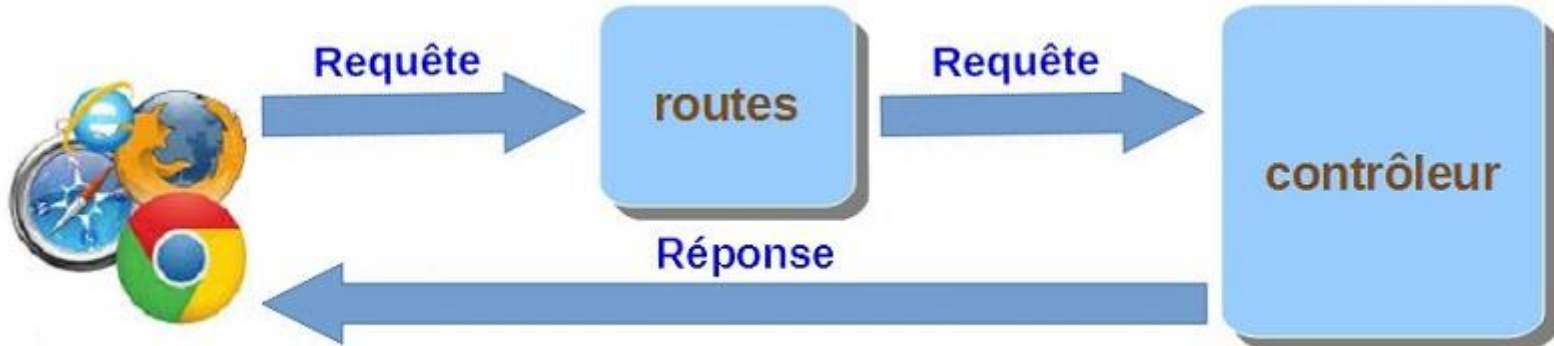


Intérêt des contrôleurs



Intérêt des contrôleurs

La tâche d'un **contrôleur** est de recevoir une requête (qui a déjà été sélectionnée par une route) et de définir la réponse appropriée, rien de moins et rien de plus. Voici une illustration du processus :



Créer un contrôleur sous Laravel



Créer un contrôleur sous Laravel

Pour créer un contrôleur à partir de la fenêtre du terminal, ouvrez le terminal et changez le répertoire vers votre dossier racine laravel. Une fois que vous y êtes, vous pouvez exécuter la commande suivante pour créer un contrôleur.

Créer un simple contrôleur vierge:

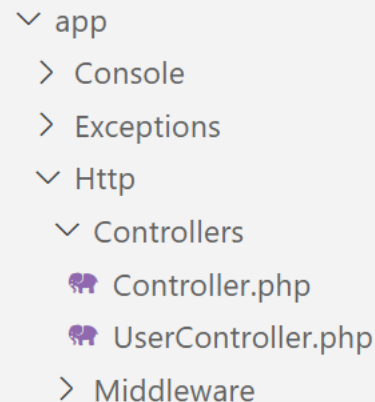
Créer un contrôleur sous sa forme la plus simple se fait grâce à la commande suivante :

php artisan make:controller *NomDeMonController*

Exemple:

php artisan make:controller UserController

➤ **Artisan**, l'outil en ligne de commande fourni avec Laravel, permet de créer rapidement un fichier contenant la structure de base d'un contrôleur.



```

  app
  > Console
  > Exceptions
  > Http
    Controllers
      Controller.php
      UserController.php
    Middleware

```


Créer un contrôleur sous Laravel

Dans l'exemple précédent nous avons créer le controller **UserController**.

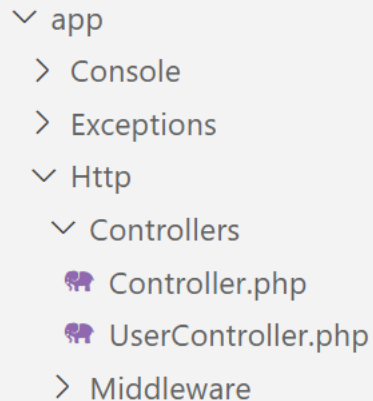
Le contrôleur crée est sous le dossier **app/Http/Controllers**

Ajoutons la méthode **index** au controlleur **UserController** :

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    //
    public function index() {
        return view("welcome");
    }
}
```



Créer un contrôleur sous Laravel

❖ Appel depuis la route:

Maintenant nous devons faire le lien entre notre fichier **web.php** et notre Controller.

Nous devons donc changer notre deuxième paramètre :

```
Route::get('/', [UserController::class, 'index']);
```

- A partir de Laravel 8.x, l'appel d'une action d'un contrôleur s'effectue en passant un tableau comme deuxième paramètre de la fonction; Le tableau comprend le nom de la classe contrôleur et le nom de la méthode à appeler.

Créer un contrôleur sous Laravel

❖ Appel depuis la route: Utilisation des paramètres

Ajoutons à notre contrôleur la méthode **get** prenant en argument un parameter \$id:

```
class UserController extends Controller
{
    public function get($id) {
        return "page $id";
    }
}
```

Ajoutons dans le fichier **web.php** la route suivante:

```
Route::get('/get/{id}', [UserController::class, 'get']);
```



Implémentation de contrôleur

Implémentation de contrôleur

Si une action de contrôleur est particulièrement complexe, vous trouverez peut-être pratique de dédier une classe de contrôleur entière à cette action unique. Pour ce faire, vous pouvez définir une seule méthode magique **__invoke** dans le contrôleur;

- Vous pouvez générer un contrôleur invocable en utilisant l'option **--invokable** de la commande Artisan **make:controller** :

```
php artisan make:controller HomeController --invokable
```

Implémentation de contrôleur

```
namespace App\Http\Controllers;

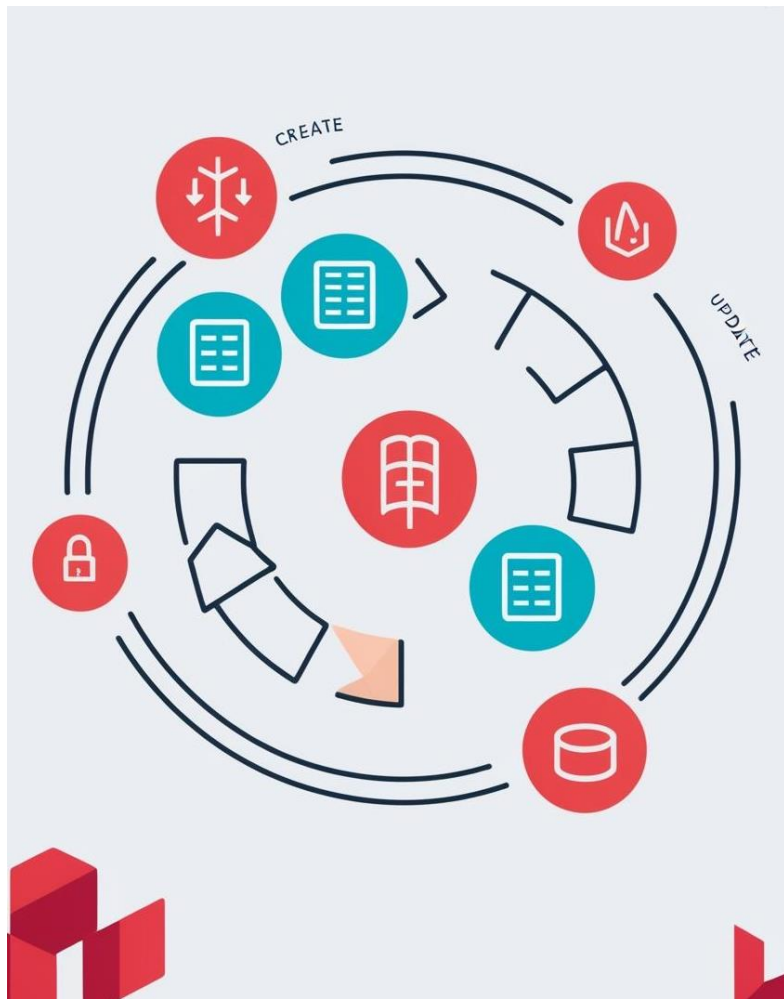
use Illuminate\Http\Request;

class HomeController extends Controller
{
    /**
     * Handle the incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function __invoke(Request $request)
    {
        return "Bienvenue à la page home";
    }
}
```

Implémentation de contrôleur

Pour appeler ce type de contrôleur via la route, il suffit d'indiquer la classe du contrôleur sans évoquer une méthode spécifique:

```
Route::get('/home', HomeController::class);
```



Contrôleur de ressources

Contrôleur de ressources

Les contrôleurs de ressources Laravel fournissent les routes CRUD (Create, Read, Update, Delete) au contrôleur en une seule ligne de code.

Un contrôleur de ressources est utilisé pour créer un contrôleur qui gère toutes les requêtes http stockées par votre application (get, post, update, delete ...).

Pour commencer, nous pouvons utiliser l' option **make:controller** de la commande Artisan **--resource** pour créer rapidement un contrôleur pour gérer ces actions :

```
php artisan make:controller PostController --resource
```

Cette commande générera un contrôleur sous **app/Http/Controllers/PostController.php**. Le contrôleur contiendra une méthode pour chacune des opérations de ressources disponibles.

Contrôleur de ressources

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PostController extends Controller
{
    public function index()
    {
        // Code pour afficher tous les posts
    }

    public function create()
    {
        // Code pour afficher le formulaire de création
    }

    public function store(Request $request)
    {
        // Code pour enregistrer un nouveau post
    }

    public function show($id)
    {
        // Code pour afficher un post spécifique
    }
}
```

```
public function edit($id)
{
    // Code pour afficher le formulaire de modification
}

public function update(Request $request, $id)
{
    // Code pour mettre à jour un post
}

public function destroy($id)
{
    // Code pour supprimer un post
}
}
```

Contrôleur de ressources

La méthode **resource** () de la classe **Route** est une fonction statique comme la méthode **get** () qui donne accès aux différentes routes que nous pouvons utiliser dans un contrôleur.

Syntaxe de la méthode resource() :

```
Route::resource("/posts", PostController::class);
```

Cette déclaration de route unique crée plusieurs routes pour gérer diverses actions sur la ressource.

Le contrôleur généré contient déjà les méthodes souhaitées pour chacune de ces actions.

N'oubliez pas que vous pouvez toujours obtenir un aperçu rapide des routes de votre application en exécutant la commande Artisan **route:list**.

En exécutant la commande ci-après, on obtient:

```
php artisan route:list --except-vendor
```

GET HEAD	posts	posts.index › PostController@index
POST	posts	posts.store › PostController@store
GET HEAD	posts/create	posts.create › PostController@create
GET HEAD	posts/{post}	posts.show › PostController@show
PUT PATCH	posts/{post}	posts.update › PostController@update
DELETE	posts/{post}	posts.destroy › PostController@destroy
GET HEAD	posts/{post}/edit	posts.edit › PostController@edit

Contrôleur de ressources

Méthode HTTP	URL	Action du contrôleur	Description
GET	/posts	index	Afficher une liste de toutes les ressources.
GET	/posts/create	create	Afficher un formulaire pour créer une nouvelle ressource.
POST	/posts	store	Traiter la requête pour enregistrer une nouvelle ressource.
GET	/posts/{post}	show	Afficher une ressource spécifique.
GET	/posts/{post}/edit	edit	Afficher un formulaire pour modifier une ressource.
PUT/PATCH	/posts/{post}	update	Traiter la requête pour mettre à jour une ressource existante.
DELETE	/posts/{post}	destroy	Supprimer une ressource spécifique.

Contrôleur de ressources

Vous pouvez même enregistrer plusieurs contrôleurs de ressources à la fois en passant un tableau à la méthode **resources** :

```
Route::resources([  
    'photos' => PhotoController::class, 'posts' => PostController::class,  
]);
```

QCM



```
if (count($workerProc) < 1) {  
    $test_files = array_merge($test_files, $sequentialTests);  
    $sequentialTests = [];  
}  
$files = [];  
$maxBatchSize = $valgrind ? 1 : ($batchSize < 100 ? $batchSize : 100);  
$averageFilesPerWorker = max(1, ($batchSize / $workerProc));  
$batchSize = min($maxBatchSize, $averageFilesPerWorker);  
while (count($files) < $batchSize && $batchSize > 0) {  
    foreach ($fileConflictsWith as $file) {  
        if (isset($activeConflicts[$fileConflictsWith[$file]]) &&  
            $waitingTests[$fileConflictsWith[$file]] > 0) {  
            continue 2;  
        }  
        $files[] = $file;  
    }  
    if ($files) {  
        foreach ($files as $file) {  
            $workerProc[] = $file;  
        }  
    }  
}
```

TP

A decorative header bar with a dark blue background. It features several 3D cubes outlined in a reddish-orange color. Some cubes are faint and in the background, while others are more prominent in the foreground.

À la prochaine !