Routes, paramètres et contrôleurs

https://github.com/heig-vd-devprodmed-course/heig-vd-devprodmed-course

Visualiser le contenu complet sur GitHub à cette adresse.

L. Delafontaine, avec l'aide de GitHub Copilot.

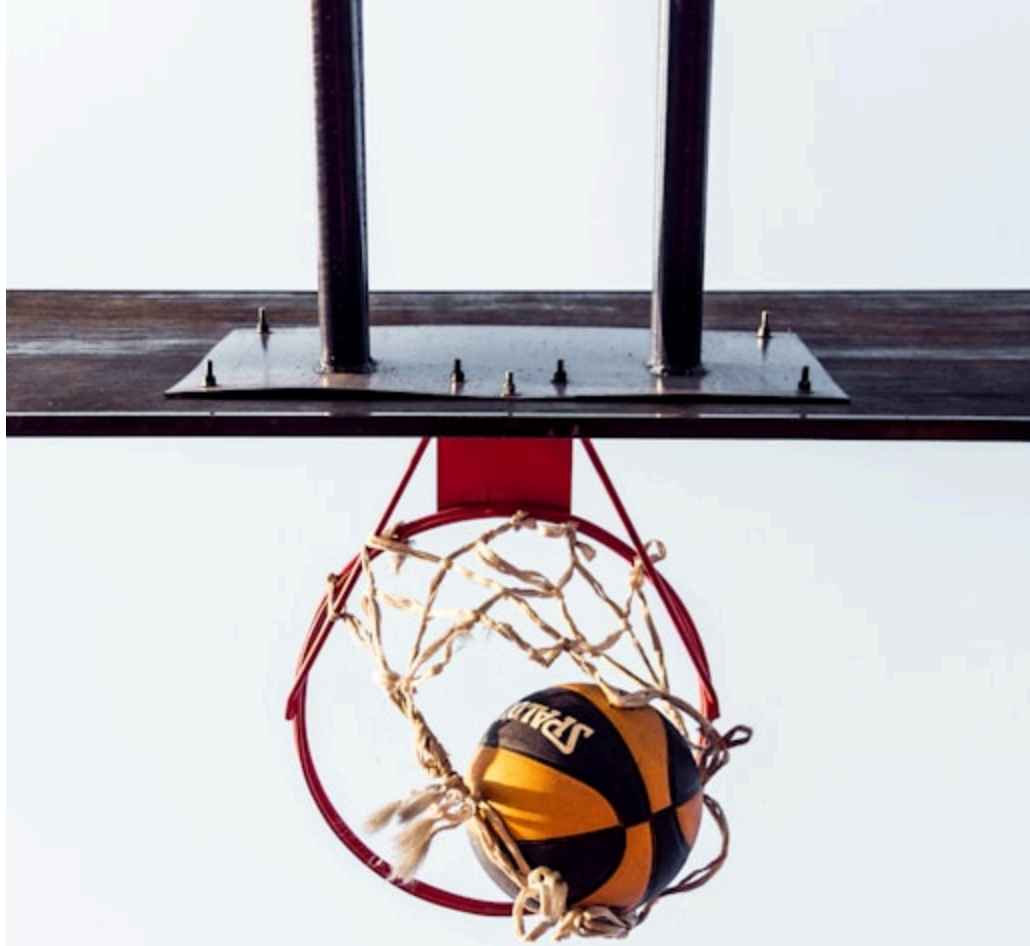
Plus de détails sur GitHub

Cette présentation est un résumé du contenu complet disponible sur GitHub.

Pour plus de détails, consulter le contenu complet sur GitHub ou en cliquant sur l'en-tête de ce document.

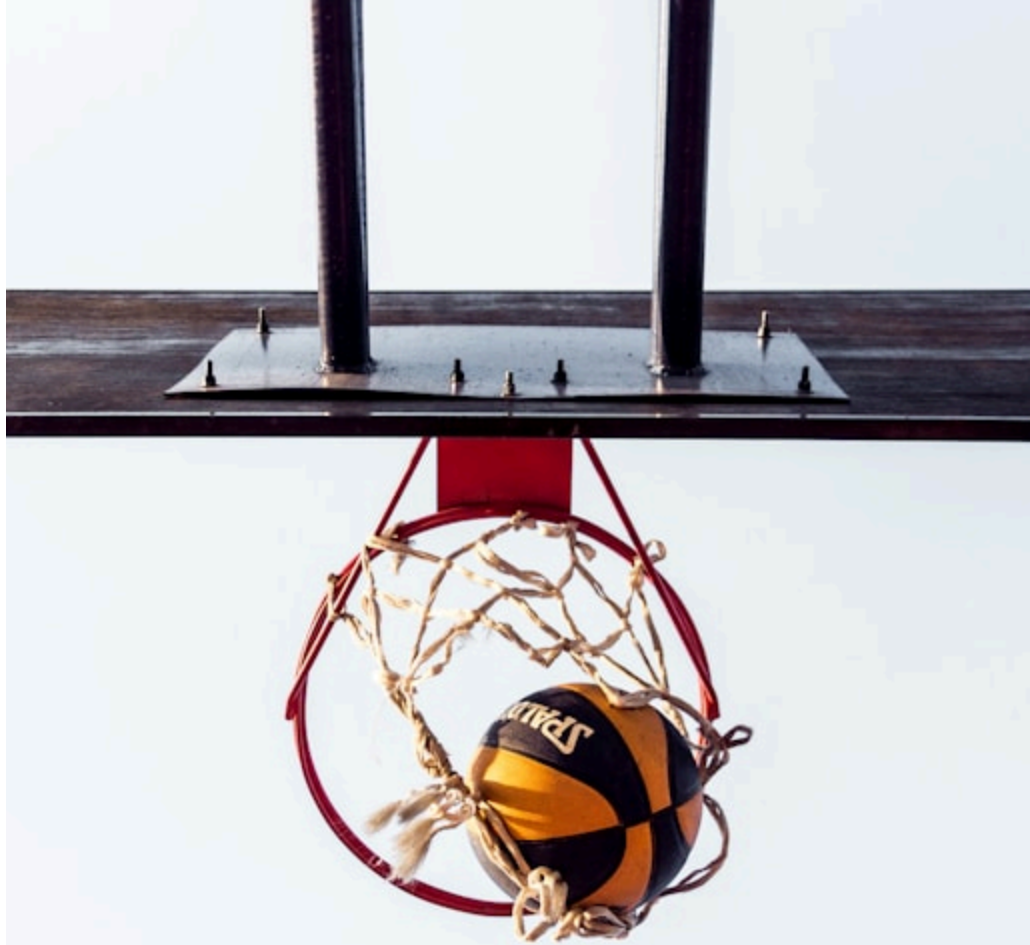
Objectifs (1)

- Décrire la partie "contrôleur" du patron de conception MVC.
- Lister les différentes méthodes HTTP et leur utilisation.
- Décrire le concept de routes dans une application Laravel.
- Définir des routes avec des paramètres dans Laravel.



Objectifs (2)

- Créer des contrôleurs dans Laravel.
- Utiliser les contrôleurs pour gérer les requêtes HTTP et retourner des réponses.
- Résumer les concepts du patron MVC et les dossiers où les trouver dans Laravel.
- Implémenter ces concepts avec Laravel pour réaliser le petit réseau social du mini-projet.



Introduction aux contrôleurs dans le patron MVC

Le patron MVC sépare une application en trois composants :

- **Modèle** : données et logique métier.
- **Vue** : présentation des données.
- **Contrôleur** : gestion des requêtes HTTP et du flux.

Le contrôleur agit comme un pont entre les vues et les modèles, gérant leurs interactions.

Rappels sur le protocole HTTP

HTTP est le protocole de communication pour les applications web.

Termes clés :

- Ressources.
- Requêtes et réponses HTTP.
- Méthodes HTTP.
- En-têtes HTTP.
- Corps de requête/réponse.

Ressources (1)

Une ressource est une entité identifiable, représentée par une URL.

`https://gaps.heig-vd.ch/consultation/fiches/uv/uv.php?id=6082`

Composants :

- Protocole (`https://`).
- Nom de domaine (`gaps.heig-vd.ch`).
- Chemin d'accès (`/consultation/fiches/uv/uv.php`).
- Paramètres de requête (`?id=6082`).

Ressources (2)

Dans une application de réseau social :

- Ressources : utilisateur.trices, publications, commentaires, etc.
- Organisation hiérarchique : `/posts/123`.
- Structure conçue à partir de la base de données et des modèles Laravel.

Les ressources sont parfois appelées "endpoints" ou "routes", mais elles font référence aux entités manipulées par l'application.

Requêtes et réponses HTTP

HTTP utilise un modèle requête-réponse :

1. Un client (votre navigateur) envoie une requête au serveur.
2. Le serveur traite et répond avec une réponse.

Exemple : création d'une publication.

1. Requête `POST` avec données dans le corps.
2. Réponse : page HTML, JSON, redirection, etc.

Le contrôleur traite la requête et détermine la réponse appropriée.

Méthodes HTTP

Les méthodes HTTP indiquent l'action à effectuer :

- `GET` : récupérer une ressource ou collection.
- `POST` : créer une nouvelle ressource.
- `PUT` / `PATCH` : mettre à jour une ressource.
- `DELETE` : supprimer une ressource.

Un navigateur envoie `GET` pour les liens et `POST` pour les formulaires par défaut.

En-têtes HTTP

Paires clé-valeur fournissant des informations supplémentaires.

Exemples :

- Type de contenu.
- Informations d'authentification.
- Préférences de langue.

Exemple : `Content-Type: application/json` indique un corps au format JSON.

Corps de requête/réponse

Contient les données envoyées ou reçue par le client ou le serveur.

Exemple : lors de la création d'une publication, le corps de la requête `POST` contient les données de la publication (JSON ou données de formulaire).

En retour, l'application peut répondre avec une page HTML, un JSON, ou une redirection.

Lors de chaque requête ou réponse, des données sont transférées pour permettre l'interaction entre le client et le serveur.

Routes (1)

Les routes sont les chemins d'URL permettant d'accéder aux ressources.

Dans Laravel :

- `routes/web.php` : routes web.
- `routes/api.php` : routes d'API (pour une future séance).

Les routes associent des URL à des actions via les méthodes HTTP.

Routes (2)

```php
Route::get('/profile', function () {
    $user = User::where('username', 'janedoe')->first();

    $posts = Post::where('user_id', $user->id)
        ->orderBy('created_at', 'desc')
        ->with(['user', 'likes'])
        ->get();

    return view('profile', ['user' => $user, 'posts' => $posts]);
});
```

Répond à une requête GET sur `/profile` et retourne une vue.

Paramètres de route (1)

Les routes peuvent inclure des paramètres dynamiques :

```php
Route::get('/profile/{username}', function ($username) {
    $user = User::where('username', $username)->first();

    $posts = Post::where('user_id', $user->id)
        ->orderBy('created_at', 'desc')
        ->with(['user', 'likes'])
        ->get();

    return view('profile', ['user' => $user, 'posts' => $posts]);
});
```

Paramètres de route (2)

La syntaxe `/{username}` capture la partie de l'URL et la passe à la fonction.

Validation des paramètres avec des contraintes :

```
Route::get('/profile/{username}', function ($username) {
    // ...
})->where('username', '[A-Za-z0-9\-]+');
```

Si l'URL ne correspond pas, Laravel retourne une erreur 404. Permet de s'assurer que les données reçues sont valides avant traitement.

Contrôleurs (1)

Gérer toutes les routes dans le fichier `routes/web.php` avec des fonctions anonymes devient difficile à maintenir.

Solution : déléguer la logique à des contrôleurs dédiés.

Avantages :

- Meilleure organisation du code.
- Respect du patron MVC.
- Facilite la maintenance.

Contrôleurs (2)

Chaque contrôleur est responsable d'un ensemble de fonctionnalités liées à une ressource :

- `PostController` : gérer les publications.
- `UserController` : gérer les utilisateur.trices.
- `LikeController` : gérer les likes.
- `MyProfileController` : gérer le profil de la personne connectée.
- etc.

➡️ Facilite la compréhension et la maintenance du code.

Créer un contrôleur

Commande de base :

```
php artisan make:controller NomDuContrôleurController
```

Convention : suffixe `Controller` (`UserController`, `PostController`, etc.).

Crée un fichier dans `app/Http/Controllers/` avec une classe de base vide.

Créer la méthode d'un contrôleur (1)

Convention pour les méthodes d'un contrôleur de ressource :

- `index()` : affiche une liste de ressources.
- `create()` : affiche un formulaire de création.
- `store(Request $request)` : traite le formulaire de création.
- `show($id)` : affiche une ressource spécifique.
- `edit($id)` : affiche un formulaire d'édition.
- `update(Request $request, $id)` : traite le formulaire d'édition.
- `destroy($id)` : supprime une ressource.

Créer la méthode d'un contrôleur (2)

Exemple de méthodes :

```
class PostController extends Controller
{
    public function index()
    {
        $posts = Post::orderBy('created_at', 'desc')
            ->with('user')->with('likes')->get();

        return view('posts.index', ['posts' => $posts]);
    }
```

```
    public function show(string $id)
    {
        $post = Post::with('user')->with('likes')->findOrFail($id);

        return view('posts.show', ['post' => $post]);
    }
}
```

Associer une route à une méthode du contrôleur (1)

Associer les routes aux méthodes du contrôleur :

```php
use App\Http\Controllers\PostController;

Route::get('/posts', [PostController::class, 'index']);
Route::get('/posts/{id}', [PostController::class, 'show']);
```

Lorsqu'une requête est envoyée, la méthode correspondante est exécutée.

Associer une route à une méthode du contrôleur (2)

Définir des routes pour d'autres méthodes :

```
Route::post('/posts', [PostController::class, 'store']);
Route::put('/posts/{id}', [PostController::class, 'update']);
Route::delete('/posts/{id}', [PostController::class, 'destroy']);
```

Utiliser les méthodes HTTP appropriées (`POST` , `PUT` , `DELETE` , etc.).

Créer un contrôleur de ressource

Laravel permet de créer un contrôleur complet :

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

Génère toutes les méthodes standard (`index` , `create` , `store` , `show` , `edit` , `update` , `destroy`).

```
Route::resource('posts', PostController::class);
```

Une seule ligne génère toutes les routes pour les actions CRUD.

Gérer les erreurs dans les contrôleurs

Laravel fournit des mécanismes pour gérer les exceptions.

Lorsqu'une ressource n'est pas trouvée par exemple (avec `findOrFail()` par exemple), Laravel retourne une erreur 404.

Laravel gère automatiquement plusieurs types d'erreurs que nous étudierons plus en détail dans une future séance.

La documentation offre des ressources précieuses : https://laravel.com/docs/12.x/errors.

Le patron MVC : récapitulatif (1)

Le patron MVC sépare l'application en trois composants :

- **Modèle** : données et logique métier (base de données, validations, relations).
- **Vue** : présentation des données (interface utilisateur).
- **Contrôleur** : gestion des requêtes HTTP et du flux (pont entre modèles et vues).

Ces composants interagissent pour créer une application web structurée et maintenable.

Le patron MVC : récapitulatif (2)

Organisation dans Laravel :

- **Modèles** : `app/Models/` et migrations dans `database/migrations/`.
- **Vues** : `app/View/` (classes) et `resources/views/` (fichiers Blade).
- **Contrôleurs** : `app/Http/Controllers/`.

Le patron MVC est extrêmement courant dans le développement web et est utilisé par de nombreux frameworks.

Une bonne compréhension de ce model vous sera très utile le futur.

Conclusion

Les routes et contrôleurs permettent de gérer les requêtes HTTP de manière structurée.

Le patron MVC sépare clairement les responsabilités.

Vous avez maintenant les bases pour développer des applications avec Laravel en utilisant le patron MVC.

Questions

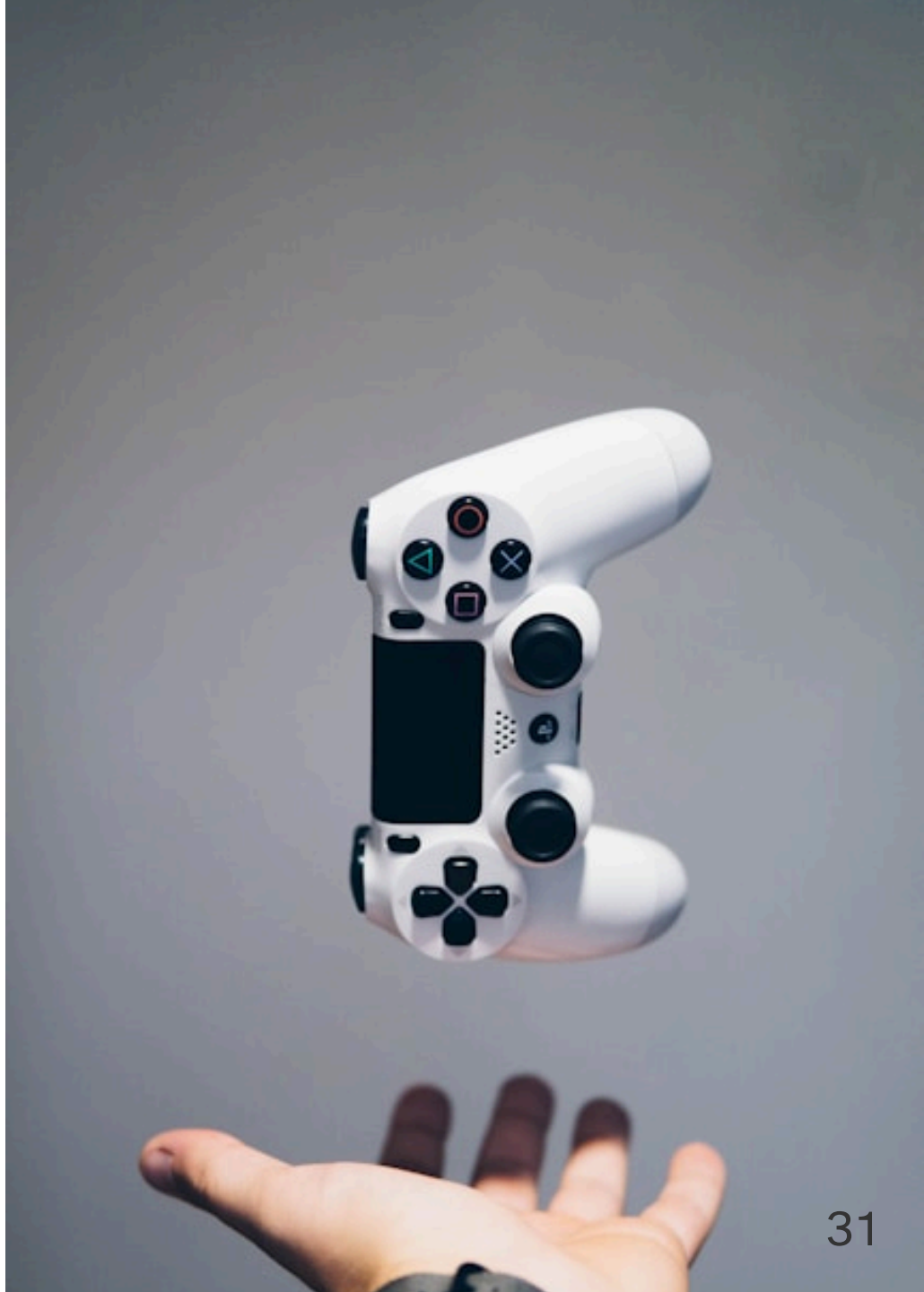
Est-ce que vous avez des questions ?

À vous de jouer !

- (Re)lire le contenu de cours.
- Faire les exercices.
- Faire le mini-projet.
- Poser des questions si nécessaire.

➡️ Visualiser le contenu complet sur GitHub.

N'hésitez pas à vous entraidez si vous avez des difficultés !



Sources

- Illustration principale par Richard Jacobs sur Unsplash
- Illustration par Aline de Nadai sur Unsplash
- Illustration par Nikita Kachanovsky sur Unsplash