

Contrôleurs

Une fois les routes créées, on peut donc utiliser les contrôleurs pour faire le lien avec les vues du projet.

Chaque contrôleur utilise un **namespace** qui correspond à un emplacement dans le système de fichiers. Cela va lui permettre de savoir de connaître les noms des routes de son dossier, et ainsi d'y faire référence. Par exemple, les contrôleurs présents dans le dossier **modules/Admin** vont indiquer être dans ce namespace, et pourront ainsi faire références aux routes de ce dossier. Cela permet de bien diviser le projet, pour éviter des manipulations entre des contrôleurs qui ne sont pas censés communiquer, et pouvoir ainsi augmenter la sécurité.

Affichage d'une vue

Les contrôleur sont des classes héritants d'un contrôleur nommé **BaseController**, héritant lui-même d'un contrôleur nommé **Controller**, qui lui provient directement de CodeIgniter. **BaseController** est une classe abstraite, qui va permettre de charger des éléments si besoin et ainsi d'améliorer les performances. Son utilisation est détaillé dans [la documentation de CodeIgniter](#) mais elle n'a pas beaucoup utilisée par Castopod.

Si on reprend l'exemple de la page d'accueil, nous avons donc besoin d'afficher une page avec la liste des podcasts disponibles, qui sont triés suivant un certain ordre. Pour cela, on va avoir besoin de savoir quelles sont les podcasts inscrits dans la base de données, et donc utilisés le modèle associé. Pour les podcasts, le modèle est **PodcastModel**, et pour savoir comment les triés, un simple paramètre dans la requête permettra de savoir comment l'utilisateur veut trier les podcasts.

Pour rappel, la route était :

```
$routes->get('/', 'HomeController', [  
    'as' => 'home',  
]);
```

Il n'y a pas de fonction particulière du contrôleur qui a été appelée par la route. Pour définir la fonction par défaut du contrôleur, il suffit de déclarer une fonction nommée *index()*. Ensuite, pour la signature de la fonction, on a plusieurs solutions, qui sont généralement :

- Un objet string avec la fonction *view()*, qui attendra le nom d'une vue

- Un objet `RedirectResponse`, avec la fonction `redirect()->route()`, qui attendra lui le nom d'une route

Les objets qui peuvent être renvoyés proviennent du dossier `CodeIgniter\HTTP`, et peuvent être importés à l'aide du mot-clé **use**.

Ici, le contrôleur renvoie deux types différents en fonction de la réponse du modèle de podcast, si un seul podcast est récupéré alors on chargera directement la page du podcast, sinon on affichera la vue nommée **home** :

```
namespace App\Controllers;

use App\Models\PodcastModel;
use CodeIgniter\HTTP\RedirectResponse;

class HomeController extends BaseController
{
    public function index(): RedirectResponse | string
    {
        // Paramètres possibles du GET
        $sortOptions = ['activity', 'created_desc', 'created_asc'];
        // Récupérer le paramètre, ou mettre par activité récente le tri
        $sortBy = in_array($this->request->getGet('sort'), $sortOptions, true) ? $this->request->getGet(
            'sort'
        ) : 'activity';

        // Récupérer les podcasts
        $allPodcasts = (new PodcastModel())->getAllPodcasts($sortBy);

        // Regarder s'il n'y a qu'un podcast, et dans ce cas renvoyé sur la page
        // du podcast directement
        if (count($allPodcasts) === 1) {
            return redirect()->route('podcast-activity', [$allPodcasts[0]->handle]);
        }

        // Création d'un objet data, qui va contenir les podcasts récupérés
        $data = [
            'podcasts' => $allPodcasts,
            'sortBy' => $sortBy,
        ];
    }
}
```

```
// Charger la vue avec l'objet data
return view('home', $data);
}
}
```

Vous remarquerez qu'un objet *data* a été créé avec deux paramètres, *podcasts* qui contient la liste des podcasts récupérés et *sortBy* qui contient la manière dont les podcasts sont triés. Dans la vue **home**, on pourra directement utiliser ces valeurs en faisant référence aux paramètres. Pour cela, on pourra directement utiliser les variables *podcasts* et *sortBy* dans la vue.

Utilisation des paramètres

Paramètres d'un groupe

Nous avons vu auparavant comment il était possible d'indiquer des paramètres à un contrôleur en prenant l'exemple d'un groupe podcast. Pour rappel, l'URL ressemblait donc à cela : *URL/@nom_du_podcast*.

Nous allons maintenant voir comment est géré ce paramètre pour le contrôleur lié au podcast. Pour cela, le contrôleur implémente une fonction nommée *_remap*. Cette fonction va permettre de récupérer les paramètres donnés dans l'URL lorsqu'une fonction de ce contrôleur est appelée.

Si on reprend notre exemple, on veut donc récupérer le nom du podcast. Il n'existe pas de route qui appelle le contrôleur sans avoir le nom du podcast dans l'URL, il faudra donc lever une erreur dans le cas où aucun paramètre n'est donné.

Ensuite, on peut également vérifier si ce podcast existe dans la base de données. Pour cela, un appel au modèle peut être effectué en cherchant dedans s'il existe un podcast avec ce nom. Dans le cas où un podcast est trouvé, on peut utiliser une entité **Podcast**, qui contiendra les informations sur le podcast recherché (son identifiant, son titre, sa description, etc.)

Ce podcast pourra ensuite être enregistré dans un attribut du contrôleur, pour pouvoir par la suite manipuler ce podcast pour récupérer les différentes informations, et ainsi les afficher dans la vue.

Tout ça, c'est exactement ce que fait le contrôleur **PodcastController**, en redéfinissant la méthode *_remap()* :

```
public function _remap(string $method, string ...$params): mixed
{
    if ($params === []) {
        throw PageNotFoundException::forPageNotFound();
    }
}
```

```

if (
    ! ($podcast = (new PodcastModel())->getPodcastByHandle($params[0])) instanceof Podcast
) {
    throw PageNotFoundException::forPageNotFound();
}

$this->podcast = $podcast;

unset($params[0]);

return $this->{$method}(...$params);
}

```

Une fois cette méthode écrite, on peut aisément utiliser la variable *\$this->podcast* pour gérer le podcast sélectionné par l'utilisateur dans la suite du contrôleur.

Pour donner un exemple simplifié, pour afficher la page avec tous les épisodes d'un podcast, le contrôleur **PodcastController** utilise donc cette variable en créant une nouvelle variable *\$data* :

```

$data = [
    'podcast' => $this->podcast,
    'episodes' => (new EpisodeModel())->getPodcastEpisodes(
        $this->podcast->id,
        $this->podcast->type,
    ),
];

```

Et affiche ensuite la page avec les données chargées :

```

return view('podcast/episodes', $data);

```

Si vous allez voir le contrôleur, vous verrez que nous avons simplifié les fonctions pour présenter directement l'utilisation des paramètres. Le contrôleur ajoute en effet un système pour gérer l'enregistrement d'une requête à la page d'un podcast pour pouvoir faire de la statistique derrière (et plus précisément le non-enregistrement lorsque l'utilisateur est connecté, pour ne pas influencer ces statistiques par un administrateur), une gestion de cache pour améliorer les performances, et la gestion des années et des saisons sélectionnées par l'utilisateur.

Paramètres d'une requête

Il existe un deuxième type de paramètres, ceux qui sont liés directement à la requête. Lorsque vous faites une requête GET avec votre navigateur, vous pouvez indiquer des paramètres dans l'URL pour permettre de passer des informations à la page. Par exemple, pour la page *monsie.fr/mapage?nombre=1&lettre=a*, la page aura 2 paramètres qui sont nombre et lettre, qui seront respectivement égaux à 1 et a.

Si vous n'avez pas tout compris, nous vous conseillons de lire [cet article](#) du site IONIS qui explique les paramètres de requête.

CodeIgniter permet de récupérer ces paramètres via la méthode *getGet(nomVariable)* qui est lié à l'objet **Incoming Request**. On peut également récupérer des paramètres d'autres types de requêtes comme avec la méthode POST, avec la fonction *getPost(nomVariable)*. Comme décrit dans [la documentation de CodeIgniter](#), voici les possibilités :

```
// Tiré de la documentation de CodeIgniter

// the URI path being requested (i.e., /about)
$request->getUri()->getPath();

// Retrieve $_GET and $_POST variables
$request->getGet('foo');
$request->getPost('foo');

// Retrieve from $_REQUEST which should include
// both $_GET and $_POST contents
$request->getVar('foo');

// Retrieve JSON from AJAX calls
$request->getJSON();

// Retrieve server variables
$request->getServer('Host');

// Retrieve an HTTP Request header, with case-insensitive names
$request->header('host');
$request->header('Content-Type');

// Checks the HTTP method
$request->is('get');
$request->is('post');
```

Il existe également un service associé à la gestion des requêtes, qui permet d'utiliser la même fonction via [la fonctionnalité de Services](#) de CodeIgniter :

```
Services::request()->getGet('nomVariable')
```

Maintenant que nous avons vu comment était géré les routes par les contrôleurs, nous pouvons nous pencher sur le dernier maillon de la chaîne, c'est-à-dire [les vues](#).

Revision #17

Created 10 February 2024 16:26:36 by tanguy

Updated 11 February 2024 14:32:29 by tanguy