Fonctionnalités supplémentaires

Dans cette page, nous allons développer certaines fonctionnalités utilisées par Castopod, mais qui ne nécessite pas une page complète d'informations.

Services et Config

Services

Les services sont des fonctions qui vont pouvoir être utilisées par n'importe quelle vue ou contrôleur, et qui sont définis dans une classe ou un composant du projet. Pour les fichiers de CodeIgniter, ces services sont généralement situés dans le fichier *Config/Services.php*. C'est par exemple le cas de Vite, qui permet l'intégration de Tailwind à l'aide de son service (situé dans *App/Libraries/Vite/*) avec ce code présent sur toutes les vues, qui permet de récupérer les fichiers CSS et JS du projet :

```
<?= service('vite')
    ->asset('styles/index.css', 'css') ?>
<?= service('vite')
    ->asset('js/app.ts', 'js') ?>
```

Un autre exemple de service, avec le service **settings**, qui permet d'accéder entre autre aux valeurs du fichier de configuration général *App/Config/App.php*. C'est avec ce service par exemple que l'on peut connaître la couleur de thème voulue par l'utilisateur.
Pour cela, deux fonctions sont proposées par ce service, la fonction *get* et *set*. Dans le fichier de configuration, la couleur du thème est dans la variable *theme*, ainsi les fonctions seront :

```
// Pour récupérer la couleur du thème
service('settings')
  ->get('App.theme')

// Pour mettre la variable $theme comme couleur de thème
service('settings')
```

```
    ->set('App.theme', $theme);
```

Pour connaitre la liste des variables que l'on peut paramétrer via le service settings, nous vous invitons à lire la page correspondant à la configuration de Castopod dans la partie **App**.

Config

A l'instar des services, si vous fouillez dans les fichiers de Castopod, vous pourrez parfois tomber sur des lignes PHP qui ressemble à cela :

```
namespace Modules\Admin\Config;
...
config(Admin::class)->gateway
```

Cela permet de réutiliser un attribut commun, ici par exemple l'emplacement URL des pages d'administrations. CodeIgniter va ici remplacer cette partie de code par l'attribut *gateway* de la classe Admin du dossier **Config**.

Si nous allons à cette emplacement, c'est-à-dire *modules/Admin/Config/Admin.php*, nous avons ce fichier :

```
class Admin extends BaseConfig
{
    // Attribut gateway de la classe Admin
    public string $gateway = 'cp-admin';
    ...
}
```

CodeIgniter va donc remplir *config(Admin::class)->gateway* par *cp-admin*.

Pour conclure, Config est un mot-clé qui va plus être utilisé pour accéder à des fonctions ou variables liées à la configuration initiale du projet, tandis que Services est liée à des objets qui vont permettre d'utiliser des fonctionnalités spécifiques de l'application.

Helper

Les **Helpers** sont des fonctions qui ressemblent beaucoup à ce que peut proposer les Services. La différence est que les Helpers vont permettre l'utilisation de fonctions globales tandis que les Services sont des composants spécifiques préchargés qui encapsulent des fonctionnalités avancées pour des parties spécifiques de l'application dans CodeIgniter.

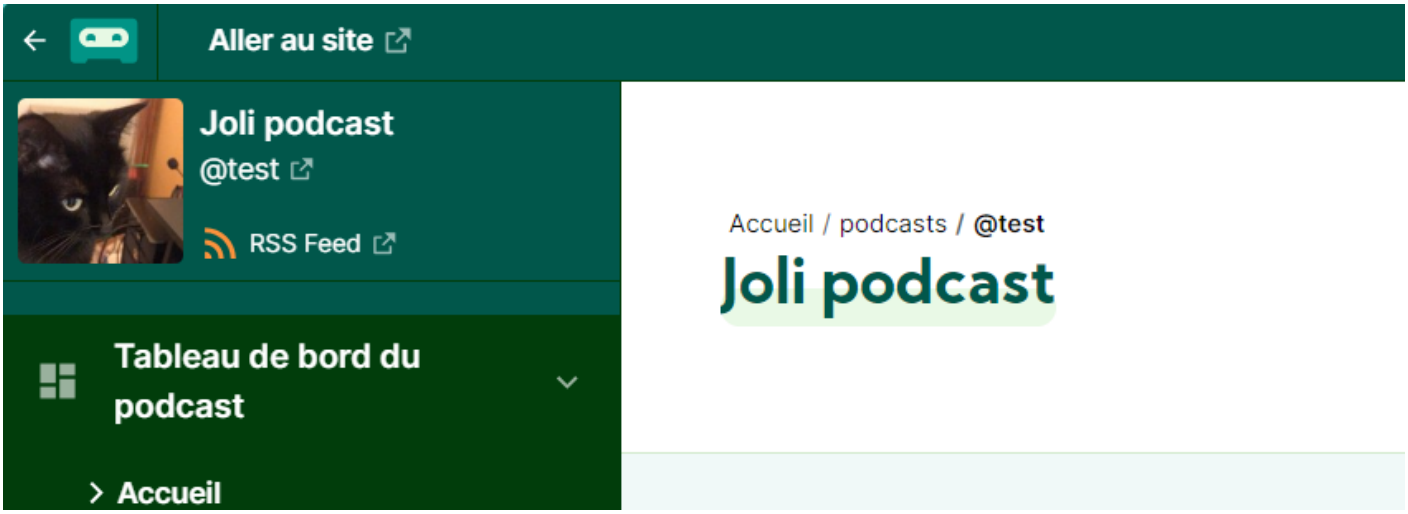
Les Helpers qui peuvent être utilisés sont écrits dans le fichier *App/Config/Autoload.php*, dans la variable **$helpers**. On y retrouve par exemple pour Castopod un Helper pour la gestion des authentifications. Mais on peut également intégrer un Helper directement dans un contrôleur, à l'aide de la fonction *helper()*, dans laquelle on va indiquer les Helpers que l'on souhaite utiliser.

Castopod utilise un framework de CodeIgniter pour gérer l'authentification et les autorisations (accessible dans le dossier *vendor/codeigniter4/shield/src/Authentication*) et qui propose un Helper pour savoir si un utilisateur est authentifier. On retrouve une fonction *loggedIn* d'ailleurs dans la page principale du site, qui permet de savoir si la vue doit afficher un bandeau d'administration :

```
<?php if (auth()->loggedIn()): ?>
    <?= $this->include('_admin_navbar') ?>
<?php endif; ?>
```

Breadcrumb

Breadcrumb (ou en français un Fil d'Ariane) est un utilitaire qui va permettre d'afficher la hiérarchie de la page actuelle :



Ici par exemple, le breadcrumb est *Accueil/Podcasts/@test*.

Pour gérer cela, Castopod utilise une bibliothèque écrite par **Ad Aures** qui permet de générer cette liste. Pour cela, la vue utilise une fonction *render_breadcrumb()* qui va générer la liste en fonction de l'emplacement de la page en utilisant la fonction *render()* de la classe **App/Librairies/Breadcrumb**.
Le texte qui sera affiché, correspondant à la page, sera celui indiqué dans le dossier **Language** dans le fichier **Breadcrump.php** de la langue correspondant à l'utilisateur.

Cette bibliothèque utilise deux fichiers :

- Le fichier helper : App/Helper/breadcrumb_helper.php

- Le fichier de la classe : App/Librairies/Breadcrumb.php

Et c'est le service *breadcrumb* qui va permettre de relier les deux.

Rules

Lorsque des données sont transmises via un formulaire, les données vont être vérifiées pour qu'elles respectent certaines règles. Par exemple, lors du téléversement d'un épisode, le fichier audio doit avoir une extension mp3 ou m4a, et la couverture de l'épisode doit être carré, avec une dimension minimale de 1400*1400px.

C'est le contrôleur associé à la création d'un épisode qui va vérifier si ces conditions ont bien été respectées. Cela va pouvoir être fait grâce à la méthode *validate()*, qui va avoir en paramètre la liste des règles qui doivent être respectées, qui va renvoyer un booléen en fonction du respect des règles et un *validator* qui va contenir le message d'erreur.

Si on reprend notre exemple, cela ressemblera à ça :

```
$rules = [
    'audio_file'          => 'uploaded[audio_file]|ext_in[audio_file,mp3,m4a]';,
    'cover'          =>
'is_image[cover]|ext_in[cover,jpg,jpeg,png]|min_dims[cover,1400,1400]|is_image_ratio[cover,1,1]',
];

// On revient en arrière avec un message d'erreur
if (! $this->validate($rules)) {
    return redirect()
        ->back()
        ->withInput()
        ->with('errors', $this->validator->getErrors());
}
```

Ce concept est expliqué plus en profondeur dans la documentation de CodeIgniter.

Components

En regardant les formulaires dans les vues de Castopod, vous remarquerez peut-être une manière un peu particulière d'écrire ces formulaires. Par exemple avec la page de connexion au site :

```
...

<?= $this->section('content') ?>

<form actions="<?= url_to('login') ?>" method="POST" class="flex flex-col w-full gap-y-4">
    <?= csrf_field() ?>

    <Forms.Field
        name="email"
        label="<?= lang('Auth.email') ?>"
        required="true"
        type="email"
        inputmode="email"
        autocomplete="username"
        autofocus="autofocus"
    />

    <Forms.Field
        name="password"
        label="<?= lang('Auth.password') ?>"
        type="password"
        inputmode="text"
        autocomplete="current-password"
        required="true" />

    ...

    <Button variant="primary" type="submit" class="self-end"><?= lang('Auth.login') ?></Button>
</form>

<?= $this->endSection() ?>

...
```

Vous aurez remarqué que les différents entrées du formulaire sont sous la forme *Forms.Field*, et qu'à la fin il y a un composant nommé *Button*. Cela est possible grâce à l'utilisation de Composants, appelés **Components** dans CodeIgniter.

Tous les composants HTLM que vous pouvez utiliser dans une vue CodeIgniter sont décrits dans le dossier **App/Views/Components**. Cela peut être pratique pour rendre vos page plus lisible et plus simple.

Vous pourrez également retrouver dans ce dossier un sous-dossier **errors** contenants les pages d'erreurs lorsqu'une page est inaccessible ou qu'un utilisateur n'a pas les droits nécessaires pour y accéder.

Revision #10
Created 11 February 2024 11:24:27 by tanguy
Updated 11 February 2024 17:00:00 by tanguy