Authentifications et autorisations

La partie gestion des connexions et déconnexions des utilisateurs à la page d'administration est gérée à l'aide du *Shield* d'authentification de CodeIgniter. Ce *shield* est un framework développé par CodeIgniter pour l'authentification et l'administration sur un projet CodeIgniter. Pour avoir plus de détail, vous pouvez vous rendre sur la page de documentation de CodeIgniter.

Modèles associés

Pour commencer, voyons comment sont enregistrés les utilisateurs et leurs informations dans la base de données.

CodeIgniter divise la gestion des données en deux tables :

- Une table *User*, gérée par le modèle *UserModel*, qui va contenir les informations liés sur l'activité du compte (sa dernière connexion, quand est-ce qu'il a été créé, etc.)
- Une table *Auth_identities* qui va enregistrer les identifiants de connexion des comptes, en hashant le mot de passe. Cette table est gérée par le framework directement, ainsi que le liant entre les deux tables (lorsqu'un utilisateur est créé sur *User*, le framework va créé cette utilisateur dans cette table également).

Pour la gestion des authorisations, la table *Auth_groups_users* va liée l'identifiant des utilisateurs à leurs droits sur le projet. Pour Castopod, il existe quatre groupes :

- *podcaster* : le groupe par défaut lorsqu'un utilisateur est créé
- *superadmin* : le groupe possédant tous les droits sur le projet
- *guest* : identique à *podcaster*, mais qui n'a pas accès au panneau de configuration
- *admin* : le groupe possédant tous les droits sur un podcast

Tous ces groupes sont gérés dans le fichier *Modules/Auth/Config/AuthGroups.php*.

Configuration du framework

Castopod utilise principalement la connexion à l'aide d'un email et d'un mot de passe. Cette connexion est gérée par des identifiants sessions. Lorsque l'utilisateur va se connecter à Castopod, si les données indiquées sont bonnes, Castopod va créer un identifiant unique et va le sauvegarder pendant un certain temps, et cette variable s'appelle une session. Castopod va ensuite renvoyer l'identifiant à l'utilisateur, qui va être enregistré dans un cookie sur son ordinateur, et qui va être utilisé pour se connecter à la page voulue.

Pour empêcher un visiteur non connecté à accéder à certaines pages, on va s'appuyer sur ce principe de session, en utilisant le fichier *App/Config/Filters.php*. Ce fichier est utilisé pour pouvoir modifier la demande d'un utilisateur en fonction de certains paramètres. Par exemple, pour gérer la connexion à la page d'administration, nous avons ceci :

```
public function __construct()
  {
      parent::__construct();


      $this->filters = [
          'session' => [
              'before' => [config('Admin')->gateway . '*'],
          ],
      ];
  }
```

On peut voir que lorsque l'utilisateur souhaite accéder à une page d'administration, identifier par *config('Admin')->gateway .* '*', s'il n'est pas connecté, il sera redirigé vers une page de connexion.

Dans le fichier *Modules/Auth/Config/Auth.php*, vous allez pouvoir configurer les liens de redirection lorsque l'utilisateur se sera connecté, à l'aide de la variable *$redirects* :

```
public array $redirects = [
    'register'          => '/',
    'login'             => '/',
    'logout'            => 'login',
    'force_reset'       => '/',
    'permission_denied' => '/',
    'group_denied'      => '/',
];
```

Vous pouvez également configurer la reconnexion lorsque l'utilisateur est déjà connecté dans le constructeur de l'objet, dans la variable du même nom.

Vosu allez pouvoir configurer les vues associées à chaque page du framework dans la variable *views* du même fichier. Vous pouvez également indiquer des routes qui seront reliées aux

contrôleurs associés à l'aide du fichier *AuthRoutes.php*.
Par exemple, pour le projet Castopod, nous avons ceci :

```
public array $views = [
    'login'                     => 'login',
    'register'                   => 'register',
    'layout'                    => '_layout',
    'action_email_2fa'           => 'email_2fa_show',
    'action_email_2fa_verify'    => 'email_2fa_verify',
    'action_email_2fa_email'     => 'emails/email_2fa_email',
    'action_email_activate_show'  => 'email_activate_show',
    'action_email_activate_email' => 'emails/email_activate_email',
    'magic-link-login'           => 'magic_link_form',
    'magic-link-message'         => 'magic_link_message',
    'magic-link-email'           => 'emails/magic_link_email',
    'magic-link-set-password'    => 'magic_link_set_password',
    'welcome-email'              => 'emails/welcome_email',
  ];
```

Nous voyons que pour la page de connexion, c'est la route *login* qui est associée. Et si nous allons voir dans le fichier *AuthRoutes.php*, nous retrouvons le contrôleur associés :

```
class AuthRoutes extends ShieldAuthRoutes
{
    public array $routes = [
        'login' => [
            ['get', 'login', 'LoginController::loginView', 'login'],
            ['post', 'login', 'LoginController::loginAction'],
        ],
        ...
    ];
}
```

C'est le contrôleur *LoginController* qui permet d'obtenir la vue associée via la méthode *loginView()*

Enfin, il est possible dans le fichier *Modules/Auth/Config/Auth.php* de paramétrer le cookie qui va enregistrer l'identifiant de l'utilisateur, en modifiant la variable *sessionConfig*. Pour l'instant, cette variable n'a pas été modifiée, donc sa configuration est celle par défaut dans le fichier *Vendor/CodeIgniter4/shield/src/Auth.php* :

```
public array $sessionConfig = [
    'field'            => 'user',
    'allowRemembering'   => true,
    'rememberCookieName' => 'remember',
    'rememberLength'     => 30 * DAY,
];
```

Lorsque la configuration est celle par défaut, les variables possèdent les valeurs du fichier présent dans ce dossier. Mais pour modifier ces valeurs, il suffit alors de paramétrer les fichiers du dossier *Modules* dans le projet de Castopod.

D'autres options de configuration sont possibles sur ce framework, pour en savoir plus nous vous conseillons à nouveau de vous rendre sur la documentation officielle.

Fonctions du framework

Une fois la configuration terminée, nous allons pouvoir nous attaquer aux fonctions proposées par ce framework. Ces fonctions vont utiliser la classe **Auth**, que nous allons pouvoir appeler à l'aide de la fonction *auth()*.

Voici les différentes fonctions présentes dans la classe **Auth** :

- *attempt()* : Elle va permettre de se connecter, en passant en paramètre un email et un mot de passe (ou d'autres champs si vous avez modifier les champs nécessaires à la connexion). Cette fonction renvoie un objet **Response**, sur lequel on va pouvoir utiliser la fonction *isOK()* pour savoir si la connexion s'est bien passée.
- *check()* : Cette fonction ressemble à la fonction précédente, car elle va permettre de vérifier si les champs indiquées en paramètres correspondent à un utilisateur. La différence est simplement que cette fonction ne va pas connecter l'utilisateur derrière, elle permet simplement de vérifier les informations.
- *loggedIn()* : Elle permet de vérifier si un utilisateur est connecté
- *logout()* : Cette fonction va déconnecter l'utilisateur, et supprimer l'identifiant unique sur le serveur lié à cette utilisateur.
- *forget()* : Cette fonction permet de supprimer tous les identifiants uniques, ce qui amène tous les utilisateurs à se reconnecter.

La classe **Auth** apporte des fonctions supplémentaires liés à la session actuelle, qui sont décrite comme ceci dans la documentation :

```
// Récupérer l'utilisateur actuel
auth()->user();
```

```
// Récupérer l'identifiant de l'utilisateur actuel
auth()->id();
// ou
user_id();

// Récupérer le 'User Provider' (UserModel par défaut)
auth()->getProvider();
```

Le *User Provider* correspond aux informations de l'utilisateur stockées avec le modèle *UserModel* par défaut comme indiqué.

Revision #5
Created 10 February 2024 15:36:34 by tanguy
Updated 16 February 2024 12:39:44 by tanguy