Code de MyCentraleAssos

Le code de MyCentraleAssos est ouvert uniquement à l'équipe travaillant sur l'appli. Ce code est disponible sur la forge de l'école :
https://forge.centrale-marseille.fr/projects/plateform-assos/repository

Technologies utilisées

MyCentraleAssos utilise la norme de développement du GInfo avec les technologies suivantes :

- Framework Symfony 4 avec les modules usuels :
  - Twig pour les templates
  - Doctrine pour l'ORM
  - FOSUserBundle pour la gestion des utilisateurs
  - Webpack Encore pour la gestion des assets (CSS & JS)
- Des modules un peu moins usuels pour certains besoins spécifiques :
  - FOSRestBundle pour l'API
  - FOSOAuthServerBundle pour gérer les accès à l'API et la connexion à des applications externes.
  - TCPDF avec un bundle Symfony pour la génération de PDFs (bulletins de cotisation)

La liste complète ainsi que les versions sont disponibles sur le composer.json du projet.

L'application est conçue pour tourner avec un serveur Apache 2 et un serveur MySQL pour la base de données.

Les technologies utilisées pour le développement sont :

- Composer pour la gestion de dépendances back (PHP)
- NPM pour la gestion de dépendances front (Modules JS etc...)

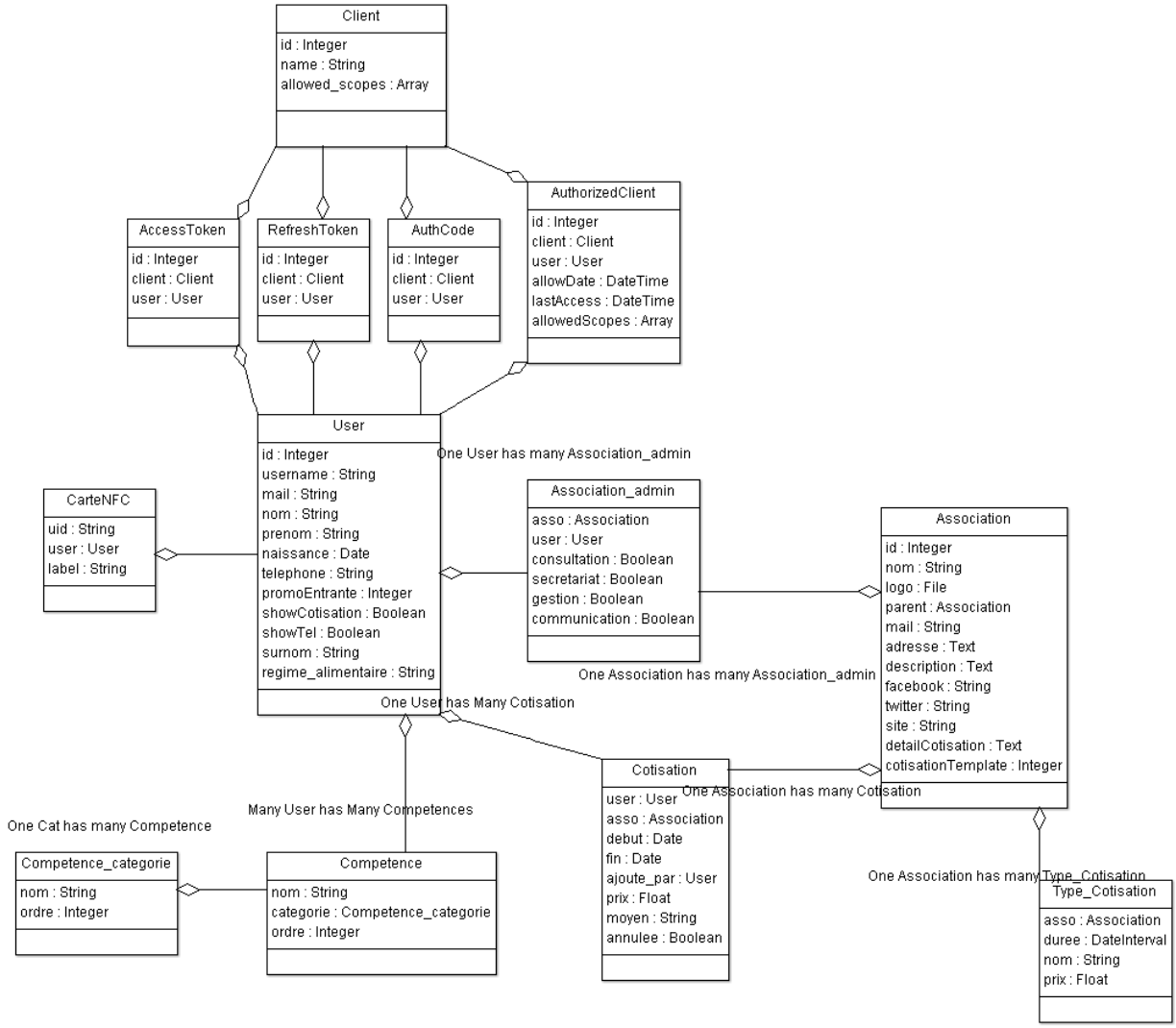
De plus diverses libraries sont utilisées en front :

- AdminBSB Material Design qui donne le thème général du site, basé sur **BOOTSTRAP 3 (ET NON 4)**
- FontAwesome & Material Icons pour les icônes
- JQuery
- DataTables pour la génération de tableaux dynamiques

La liste complète ainsi que les versions sont disponibles sur le package.json du projet.

Entités

Le schéma relationnel de MyCA peut être résumé par le diagramme UML suivant :



Le schéma est correct à quelques champs près pour quelques entités.

Utilisateurs

Détail

Associations

Détail

Cotisations

Détails

Contrôleurs

les controllers sont répartis suivant une logique fonctionnelle dans divers dossiers :

- **/src/Controller/Admin/** Les controllers servant à l'administration de la plateforme
  - *AdminController.php* : Stats administrateurs, quelques fonctions comme l'envoi de mails de relance
- **/src/Controller/API/** Tous les controllers relatifs à l'API.
  - *AssosController.php* Récupération d'informations sur les associations, les rôles. Les cotisations font l'objet de leur propre controller.
  - *CardController.php* Recherche sur les UID de cartes
  - *CotisationController.php* Récupération des données de cotisation d'une association.
  - *UserController.php* Récupération de données sur l'utilisateur courant. (Infos de base, rôles et cotisations en cours)
  - *UsersController.php* Récupération de données sur les utilisateurs du site en général. (Liste)
- **/src/Controller/Association/** Gestion de toutes les actions relatives à la gestion associative, annuaire des associations.
  - *AssoAdminController.php* Gestion des rôles dans une association
  - *AssociationController.php* Gestion d'une association en général, page de description de celle-ci
  - *CotisationController.php* Gestion de toutes les actions relatives aux cotisations, tant du coté asso que du coté user (bulletins)
- **/src/Controller/User/** Gestion des actions relatives aux utilisateurs, éditions de profil, annuaire, cartes NFC.
- **/src/Controller/IndexController.php** Page d'accueil du site pour les non connectés.

Particularités de code de MyCentraleAssos

Envoi de mails

Pour envoyer des mails facilement, on peut utiliser le service Mailer écrit pour My. Pour l'utiliser il suffit de le passer en argument de l'action, comme ceci :

```
    /**
     * @Route("/addMember/{id}", name="groupe_add_member")
     * @Security("is_granted('ROLE_USER')")
     */
    public function addMember(Groupe $groupe, Request $request, Mailer
$mailer)
    {
```

Celui-ci permet d'envoyer des mails en html et la version txt en utilisant Twig. Il faut créer des templates correspondant dans **templates/Mail/**

Utilisation :

```
public function sendMail($destinataire, $titre, $params, $html_template,
```

```
$plain_template=null, $piece_jointe=false)
```

- **Destinataire :** array [email ⇒ nom]
- **Titre :** string (titre du mail)
- **params :** array, paramètres passés au render du twig
- **html_template :** chemin du template html twig
- **plain_template :** chemin du template txt twig (sans html)
- **pice_jointe :** array

Exemple d'utilisation :

```
            $mailer->sendMail(
                [$user->getEmail() => $user->getFullName()],
                'Vous avez été ajouté au groupe '.$groupe->getNom(),
                ['gm' => $newgm],
                'Mail/Groupe/addMember.html.twig',
                'Mail/Groupe/addMember.txt.twig'
            );
```

Pour un exemple avec pièces jointes voir le **CotisationController.php**

Champ de recherche d'utilisateur

Pour mettre un champ de recherche d'utilisateur, il suffit d'avoir un select ayant la class *userseach* et d'inclure le fichier **userSearch.js**.

Exemple :

```
<select name="addUser" id="addUser" class="usersearch form-
control"></select>
```

ou un form avec twig :

```
{{ form_row(form.user, {'attr': {'class': 'usersearch'}}) }}
```

Import du fichier JS :

```
<script src="{{ asset('build/userSearch.js') }}"></script>
```

Très important : lors de l'utilisation avec un form, on envoie le select vide (recherche via ajax) mais il faut rajouter les choix sinon le form passe pas la validation. Cela se fait comme ceci (exemple formulaire cotisation) :

```
ublic function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            [...]
            ->add('user', EntityType::class, [
                'class' => 'App\Entity\User',
```

```
                'required' => true,
                'choices' => [],
                'choice_label' => function(User $user){ return
$user->getNom().' '.$user->getPrenom(); }
            ])
            [...]
        ;

        // Comme pas de choix disponibles sur le User (autocomplete), il
faut rajouter le choix de l'utilisateur aux possibilités
        $builder->addEventListener(FormEvents::PRE_SUBMIT,
function(FormEvent $event){
            $data = $event->getData();

            if(!$data || empty($data['user'])) return;

            $event->getForm()->add('user', EntityType::class,
                [
                    'class' => 'App\Entity\User',
                    'required' => true,
                    'query_builder' => function(EntityRepository $repo) use
($data){
                        return $repo->createQueryBuilder('u')->where('u.id =
:id')->setParameter('id', $data['user']);
                    },
                    'choice_label' => function(User $user){ return
$user->getNom().' '.$user->getPrenom(); }
                ]);
        });
    }
```

Cas particulier : Non validation du formulaire (sans rafraîchissement) on peut réinitialiser le champ via le code suivant (exemple ajout utilisateur dans groupe sans rechargement) :

```
$('#addUser').change(function(){
    let uid = $(this).val(),
    gid = $('#groupe_id').val(),
    csrf = $('#admin_csrf').val();

    if(!uid)
        return;

$(this).val('').trigger('change.abs.preserveSelected').selectpicker('refresh
');

    [...]
```

DataTable