Implementation de l'authentification

Je fais ici une presentation de la gestion de l'authentification, permettant de savoir rapidement où chercher si une modification est nécessaire. Je ne présente pas tous les cas et pour faire des modifications, il sera nécessaire de se référencer aussi à la documentation de symfony : `https://symfony.com/doc/current/security.html`

Fichiers importants

- `config/package/security.yaml` : Gestion de la configuration de l'authentification
- `src/Controller/SecurityController.php` : Gestions des routes
- `templates/app/security` : Dossier des templates d'authentification
- `src/Entity/User` : Gestion de l'entité utilisateur lié à la base de données via Doctrine

Configuration

Le fichier de configuration permet de multiples choses telles que la gestion des droits d'accès, définir qui est l'utilisateur, définir les routes de connection et de déconnexion, la gestion du hasher de mot de passe ainsi que la stratégie de droit.

Explication de la configuration actuel :

- La strategy *unanimous* définis que si l'on vérifie plusieurs conditions d'accès à un élement, toutes les conditions doivent être respectées pour y avoir accès.
- Dans la section *provider*, je définis que mon entité `App\Entity\User` est l'utilisateur "doctrine". J'utilise cet utilisateur dans mon firewall *main* afin que la plateforme utilise ma class comme référence. La valeur property cible la valeur de mon entité utilisée pour s'authentifier.
- Dans mon firewall, je définis qu'en environnement de **dev**, j'autorise l'accès aux routes du profiler ainsi qu'à mes assets. Ensuite, je définis mon provider comme expliqué plus tot ainsi que les routes d'accès aux différentes actions nécessaires à la connection, tel que logout, ou l'accès au formulaire de connection. Ces routes se trouvent dans notre **Controller**.
- Dans la section *access_control,* je définis les droits à certaines routes. Dans notre cas je dis que toutes routes qui débute par `/login` peut être accessible depuis n'importe quel utilisateur non-authentifié. Ensuite, je dis que toutes les routes qui débutent par `/` peut être accéssible seulement si l'on a le `ROLE_ADMIN` ou le `ROLE_USER`. Attention, la définition de ces droits se fait dans l'ordre du haut en bas. Si je mets l'autorisation `/login` en dernière ligne cela sera toujours refusé avant d'essayer de lire cette autorisation si je ne suis pas authentifié.
- La dernière section sert à customiser le hashage du mot de passe si on le souhaite.

Controller

Le controller permet la gestion des différentes routes liée à l'utilisateur. Actuellement les routes suivantes sont disponibles :

- `login` : C'est la route qui mène à la page de connexion.
- `logout` & `login_check` : Ce sont des routes qui ne renvoient rien, elles sont utilisé par symfony pour le control de l'authentification et la déconnexion. Ce controller servira donc à accueillir des evolutions tel qu'une gestion de mot de passe oublié où une modification de mot de passe.

Template

Les template de gestion d'authentification se range donc tous dans `templates/app/security` Actuellement seul la page de log existe, mais nous pourrions ajouter une page de modification de mot de passe ou d'oublie de mot de passe.

Entité

L'entité *User* est là pour définir tous les parameter de l'utilisateur. C'est ici que l'on va y définir ce qui est utilisé comme l'identifiant via le `getUserIdentifier`. L'entité va permettre d'enregistrer ces informations en base de donnée. Si vous souhaitez modifiant l'identifiant par l'email, il faudra ajouter un champ *email* ainsi que modifier la fonction `getUserIdentifier` par `$this->getEmail`.

Workflow

Chaque fois qu'une nouvelle fonctionnalité est ajoutée, il faut :

- Créer les tests associés.
- Respecter la documentation `AUDIT_CODE_PERFORMANCE`
- Respecter le Gitflow

Gitflow

- Pour toute implementation de nouvelles fonctionnalités, il faudra créer une nouvelle branche sur le repository en la nommant en anglais par le nom ou l'indication de la nouvelle fonctionnalité.
- Une fois que le développement est terminé, il faudra ensuite créer une pull request vers la branch `testing`. Si un lead développeur gére le projet, ce sera à lui de prendre en charge le merge de la pull request afin qu'il revérifie le code.
- Si un serveur de test est disponible pour faire du recettage le déploiement se fera depuis la branch testing. Les correctifs seront apportés sur la branch de la fonctionnalité et poussée sur la branch testing jusqu'à ce que tout soit opérationnel. La branch `testing` pourra ensuite être poussée vers la branch master pour être déployé en ligne.