DOCUMENTATION TECHNIQUE

Sommaire :

1. User Entity

L'utilisateur est représenté par l'entité User (src/Entity/User). Cette classe implémente le UserInterface qui est indispensable pour la gestion d'une classe utilisateur et toutes les fonctions liées à cette dernière.

```
/**
 * @ORM\Entity(repositoryClass=UserRepository::class)
 * @ORM\HasLifecycleCallbacks()
 * @UniqueEntity(fields={"email"}, message="Il existe déjà un compte avec cet e-mail")
 */
class User implements UserInterface
```

Les attributs de cette classe sont l'identifiant de l'utilisateur (auto-incrémenté), son nom d'utilisateur, son mot de passe (encrypté), son email et ses rôles

2. Security Component

a. Providers

Le providers permet de définir quelle entité représentera la classe utilisateur, il permet également de choisir quelle propriété de la classe utilisateur servira d'identifiant pour l'identification de l'utilisateur

```
providers:
    users_in_memory: { memory: null }
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

b. Firewalls

Le firewalls permet de réguler l'accès ou non à certaines pages du site.
Régler les paramètres du firewall va vous permettre de sécuriser le site. En effet, vous pouvez restreindre l'accès à certaines parties du site uniquement aux visiteurs qui sont membres. Autrement dit, il faudra que le visiteur soit authentifié pour que le firewall l'autorise à passer.

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
```

Le firewall permet de vérifier l'identité de l'utilisateur. Dans ce cas, nous sommes sur la configuration dev qui désactive la sécurité pour certaines URL. En étant réglée sur false elle nous permet d'autoriser le chargement des pages d'assets ainsi que le profiler de Symfony.

```yaml
main:
    anonymous: lazy
    provider: users_in_memory
    guard:
        authenticators:
            - App\Security\LoginFormAuthenticator
    logout:
        path: app_logout
        # where to redirect after logout
        target: login
    access_denied_handler: App\Security\AccessDeniedHandler
```

La config main permet de définir la classe qui gère l'identification et la vérification des données soumises au moment de la connexion utilisateur (App/Security/LoginFormAuthenticator).
Elle permet également de définir la route de déconnexion et la route vers laquelle il faut rediriger l'utilisateur après sa déconnexion.
On peut également définir la classe qui gérera les exceptions liées aux accès refusés.

c. Access control

```yaml
# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
    # - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }
```

Dans cette partie vous pouvez restreindre l'accès à certaines routes à des rôles spécifiques. Cette restriction est assez générale et si vous avez des routes de type **/admin/articles/list**, il vous suffit de juste de restreindre l'accès à la route **/admin** pour que la route précédente soit également inaccessible.

3. Login form

```php
class LoginFormAuthenticator extends AbstractFormLoginAuthenticator implements PasswordAuthenticatedInterface
{
    use TargetPathTrait;

    public const LOGIN_ROUTE = 'login';

    private $entityManager;
    private $urlGenerator;
    private $csrfTokenManager;
    private $passwordEncoder;
```

Toutes les informations transmises sur la page de connexion sont traitées dans la classe LoginFormAuthenticator.
Cette classe contient plusieurs fonctions :

- La fonction supports qui est appelée au début de chaque requête et qui permet de vérifier que la route visitée correspond bien à la route de login définit dans notre composant Security
- La fonction getCredentials qui permet de récupérer les valeurs de chaque champ du formulaire de connexion
- La fonction getUser permet de vérifier que l'identifiant rentré correspond bien à un utilisateur présent dans la base de données et il vérifie également la validité du token d'authentification généré lors de l'envoi.
- La fonction checkCredentials vérifie la validité du mot de passe rentré.
- La fonction getPassword permet de re-encrypter le mot de passe automatiquement dans le temps
- La fonction onAuthenticationSuccess permet de gérer l'action à effectuer dès lors que l'on sait que l'authentification de l'utilisateur s'est bien passée
- La fonction getLoginUrl retourne la route de connexion.