Etape 1 : Créer l'entité qui représente l'utilisateur avec Symfony

Dans la console taper la commande suivante :

```
php bin/console make:user
```

Cela a pour effet de créer une Entity qui peut être modifiée comme une Entity classique en tapant dans la console

```
php bin/console make:Entity EntityName
```

Cette Entity est ensuite stockée en base de donnée avec Doctrine

Une série de questions sur le paramétrage de cette Entity est posée en console (hashage de mot de passe, propriété unique).

Une fois le processus terminé, Symfony va créer l'Entity User, son repository et modifier le fichier security.yaml en ajoutant un provider basé sur la classe User nouvellement créée comme suit :

```
# config/packages/security.yaml
    # ...

    providers:
        users:
            entity:
                # the class of the entity that represents users
                class: 'App\Entity\User'
                # the property to query by - e.g. username, email, etc
                property: 'username'
                # optional: if you're using multiple Doctrine entity
                # managers, this option defines which one to use
                # manager_name: 'customer'

    # ...
```

Ajouter la possibilité de login par d'autres attributs de l'Entity User

l'Entity provider du fichier security.yaml ne peut effectuer une requête en base de données que sur le champ spécifique défini par la clé "property"

Si on veut ajouter d'autres champs (comme un email par exemple), il faut que le UserRepository implémente l'interface Symfony/Bridge/Doctrine/security/User/UserLoaderInterface qui requiert que soit implémentée la méthode loadUserByUsername($username). Ici on prendra l'exemple du User qui souhaite s'identifier par l'email ou son username

```
// src/Repository/UserRepository.php
namespace App\Repository;

use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Symfony\Bridge\Doctrine\Security\User\UserLoaderInterface;
```

```
class UserRepository extends ServiceEntityRepository implements UserLoaderInterface
{
    // ...

    public function loadUserByUsername($usernameOrEmail)
    {
        $entityManager = $this->getEntityManager();

        return $entityManager->createQuery(
                'SELECT u
                FROM App\Entity\User u
                WHERE u.username = :query
                OR u.email = :query'
            )
            ->setParameter('query', $usernameOrEmail)
            ->getOneOrNullResult();
    }
}
```

Afin d'empêcher Symfony d'utiliser le provider créé par défaut dans security.yaml et de lui permettre d'utiliser la méthode implémentée dans UserRepository, il faut retirer la clé "property" du user provider dans security.yaml comme suit

```
# config/packages/security.yaml
security:
    # ...

    providers:
        users:
            entity:
                class: App\Entity\User
```

Pour finir, il faut modifier la méthode getUser() de UserAnthenticator en appelant la méthode implémentée dans UserRepository à la place du findOneBy() défini par défaut.

```
public function getUser($credentials, UserProviderInterface $userProvider)
    {
        $token = new CsrfToken('authenticate', $credentials['csrf_token']);
        if (!$this->csrfTokenManager->isTokenValid($token)) {
            throw new InvalidCsrfTokenException();
        }

        $user = $this->entityManager->getRepository(User::class)->loadUserByUsername($credentials['username']);

        if (!$user) {
            // fail authentication with a custom error
            throw new CustomUserMessageAuthenticationException('Username could not be found.');
        }

        return $user;
    }
```