Utiliser un AD pour se connecter à un projet Symfony

Sur un projet Symfony

composer require symfony/ldap

```
lea@MacBook-Air-de-Lea notes-de-frais % composer require symfony/ldap
./composer.json has been updated
Running composer update symfony/ldap
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking symfony/ldap (v8.0.6)
Writing lock file
Installing dependencies from lock file (including require-dev)
```

Dans le .env.local (ou .env en prod), configurer les informations de l'AD
Remplacer le mot de passe par le vrai mot de passe du compte administrateur

```
LDAP_HOST=192.168.107.226
LDAP_PORT=389
LDAP_DN=CN=Administrateur,CN=Users,DC=gsb,DC=local
LDAP_PASSWORD=TonMotDePasse
LDAP_BASE_DN=DC=gsb,DC=local
```

Ajouter la configuration dans config/services.yaml

```yaml
services:
    _defaults:
        autowire: true
        autoconfigure: true

    App\:
        resource: '../src/'

    Symfony\Component\Ldap\Adapter\ExtLdap\Adapter:
        arguments:
            - host: '%env(LDAP_HOST)%'
              port: '%env(int:LDAP_PORT)%'

    Symfony\Component\Ldap\Ldap:
        arguments: ['@Symfony\Component\Ldap\Adapter\ExtLdap\Adapter']
        tags:
            - { name: ldap }

    App\Service\LdapService:
        arguments:
            $ldapDn: '%env(LDAP_DN)%'
            $ldapPassword: '%env(LDAP_PASSWORD)%'
            $ldapBaseDn: '%env(LDAP_BASE_DN)%'
```

Créer un service LDAP dans le projet Symfony qui va permettre de récupérer les utilisateurs de l'AD. Le service est divisé en plusieurs étapes :

```
/**
* Mapping des groupes AD vers les groupes en base de données
symfony, association via nom du groupe (AD) et code unique
(Symfony)
**/
private const GROUP_MAPPING = [
   'Visiteur' ⇒ 'VISITEUR',
   'Comptable' ⇒ 'COMPTABLE',
   'Administrateur Technique' ⇒ 'ADMIN_TECH',
   'Administrateur Fonctionnel' ⇒ 'ADMIN_FONC',
];
```

Ensuite :

```
/**
* Authentifie un utilisateur via LDAP et synchronise avec son
compte de l'application.
*
* Étapes :
* 1. Vérification que le "username" et le mot de passe sont
existants
* 2. Connexion LDAP avec le compte fourni dans le fichier
.env.local
* 3. Recherche l'utilisateur dans l'annuaire avec son
"sAMAccountName"
* 4. Essai de connexion au serveur LDAP avec l'id du compte
pour vérifier son mot de passe
* 5. Récupération les groupes AD de l'utilisateur
* 6. Mapping des groupes AD vers les groupes de l'application
web
* 7. Création de l'utilisateur en bdd si inexistant ou met à
jour ses informations
*
*/
public function authenticate(string $username, string
$password): ?User
{
```

```
    $username = trim($username);
    $password = trim($password);

    if ($username === '' || $password === '') {
        return null;
    }

    try {
        $this->ldap->bind($this->ldapDn, $this->ldapPassword);

        $query = $this->ldap->query(
            $this->ldapBaseDn,
            sprintf('(sAMAccountName=%s)',
$this->escapeLdapValue($username))
        );

        $results = $query->execute();

        if (count($results) === 0) {
            return null;
        }

        /** @var Entry $ldapUser */
        $ldapUser = $results[0];
        $userDn = $ldapUser->getDn();

        if (!$userDn) {
            return null;
        }

        try {
            $this->ldap->bind($userDn, $password);
        } catch (ConnectionException) {
            return null;
        }

        $this->ldap->bind($this->ldapDn, $this->ldapPassword);
```

```
        $adGroups = $this->getGroupes($ldapUser);
        $groupeCode = $this->mapGroupeToCode($adGroups);

        if ($groupeCode === null) {
            return null;
        }

        $email = $this->getFirstAttribute($ldapUser, 'mail') ??
sprintf('%s@gsb.local', $username);

        $user = $this->userRepository->findOneBy(['email' =>
$email]);

        if ($user === null) {
            $user = $this->createUser($ldapUser, $username,
$email, $groupeCode);
        } else {
            $this->updateUserFromLdap($user, $ldapUser,
$groupeCode);
        }

        return $user;
    } catch (\Throwable $e) {
        return null;
    }
}
```

Il faut ensuite gérer les groupes :

```
/**
* Extrait la liste des groupes Active Directory d'un
utilisateur LDAP.
* on récupère les groupes LDAP de l'utilisateur, on garde
seulement leur nom.
*/
private function getGroupes(Entry $ldapUser): array
{
    $groupes = [];
    $memberOf = $ldapUser->getAttribute('memberOf') ?? [];
```

```
    foreach ($memberOf as $groupDn) {
        if (preg_match('/CN=([^,]+)/i', $groupDn, $matches)) {
            $groupes[] = trim($matches[1]);
        }
    }

    return array_unique($groupes);
}
```

```
/**
* Utilise la constante GROUP_MAPPING pour associer un groupe
AD
* au code utilisé dans l'application.
*/
private function mapGroupeToCode(array $adGroups): ?string
{
    foreach ($adGroups as $group) {
        if (isset(self::GROUP_MAPPING[$group])) {
            return self::GROUP_MAPPING[$group];
        }
    }

    return null;
}
```

Ensuite on gère les utilisateurs (création et modifications):

```
/**
* Création de l'utilisateur si inexistant dans la base à
partir des infos AD.
*
*
* - email (username AD)
* - nom
* - prénom
* - Compte Actif
* - Demande de consentement des données à false pour la
première connexion
```

```
*
* Puis association au(x) groupe(s) de l'application.
*/
private function createUser(Entry $ldapUser, string $username,
string $email, string $groupeCode): User
{
    $user = new User();
    $user->setEmail($email);
    $user->setPassword('ldap_user');
    $user->setNom($this->getFirstAttribute($ldapUser, 'sn') ??
$username);
    $user->setPrenom($this->getFirstAttribute($ldapUser,
'givenName') ?? '');
    $user->setIsActive(true);
    $user->setConsentData(false);

    $this->syncUserGroup($user, $groupeCode);

    $this->em->persist($user);
    $this->em->flush();

    return $user;
}
```

```
/**
* Met à jour un utilisateur existant avec les informations
LDAP.
* - nom
* - prénom
* - Compte actif
* - Groupe Droit de l'application
*/
private function updateUserFromLdap(User $user, Entry
$ldapUser, string $groupeCode): void
{
    $user->setNom($this->getFirstAttribute($ldapUser, 'sn') ??
$user->getNom());
```

```
    $user->setPrenom($this->getFirstAttribute($ldapUser,
'givenName') ?? $user->getPrenom());
    $user->setIsActive(true);

    $this->syncUserGroup($user, $groupeCode);

    $this->em->flush();
}
```

```
/**
* Synchronisation Utilisateur AD > Utilisateur application web
*/
private function syncUserGroup(User $user, string
$groupeCode): void
{
    $groupeDroit =
$this->groupeDroitRepository->findOneBy(['code' =>
$groupeCode]);

    if ($groupeDroit === null) {
        return;
    }

    foreach ($user->getGroupeDroits() as $existingGroup) {
        $user->removeGroupeDroit($existingGroup);
    }

    $user->addGroupeDroit($groupeDroit);
}
```

Enfin,
On gère les récupérations des informations de l'AD et les échappements de caractères pour n'avoir aucun conflit entre l'Active Directory et l'application

```
private function getFirstAttribute(Entry $entry, string
$attribute): ?string
{
    $values = $entry->getAttribute($attribute);
```

```
    if (!is_array($values) || empty($values)) {
        return null;
    }

    $value = $values[0];

    return is_string($value) && trim($value) !== '' ?
trim($value) : null;
}

private function escapeLdapValue(string $value): string
{
    return str_replace(
        ['\\', '*', '(', ')', "\0"],
        ['\5c', '\2a', '\28', '\29', '\00'],
        $value
    );
}
```

Dernière étape : On modifie l'AuthentificationAuthenticator pour prendre en compte l'Active Directory lors de la connexion

```
class AuthentificationAuthenticator extends
AbstractLoginFormAuthenticator
{
    use TargetPathTrait;

    public const LOGIN_ROUTE = 'app_login';

    public function __construct(
        private UrlGeneratorInterface $urlGenerator,
        private LdapService $ldapService,
    ) {}

    public function authenticate(Request $request): Passport
    {
        $username =
$request->getPayload()->getString('username');
```

```
        $password =
$request->getPayload()->getString('password');


$request->getSession()->set(SecurityRequestAttributes::LAST_US
ERNAME, $username);

        return new Passport(
            new UserBadge($username, function() use ($username,
$password) {
                $user =
$this->ldapService->authenticate($username, $password);
                if ($user === null) {
                    throw new
CustomUserMessageAuthenticationException(
                        'Identifiants invalides ou compte non
autorisé.'
                    );
                }
                return $user;
            }),
            new CustomCredentials(fn() => true, $password),
            [
                new CsrfTokenBadge('authenticate',
$request->getPayload()->getString('_csrf_token')),
                new RememberMeBadge(),
            ]
        );
    }

    public function onAuthenticationSuccess(Request $request,
TokenInterface $token, string $firewallName): ?Response
    {
        if ($targetPath =
$this->getTargetPath($request->getSession(), $firewallName)) {
            return new RedirectResponse($targetPath);
        }
```

```
        return new
RedirectResponse($this->urlGenerator->generate('app_home'));
    }

    protected function getLoginUrl(Request $request): string
    {
        return
$this->urlGenerator->generate(self::LOGIN_ROUTE);
    }
}
```

Si utilisation d'un container tel que Docker qui ne peut pas accéder à l'Active Directory, bien penser à créer une configuration uniquement en environnement de développement permettant d'accéder à l'application avec un compte sans AD (users créées en base de données)

```
if ($this->kernelEnvironment === 'dev') {

    $localUser = $this->userRepository->findOneBy([
        'email' => $username
    ]);

    if ($localUser &&
$this->passwordHasher->isPasswordValid($localUser, $password))
{
        return $localUser;
    }
}
```

On peut alors essayer de se connecter à l'application grâce à un identifiant Active Directory