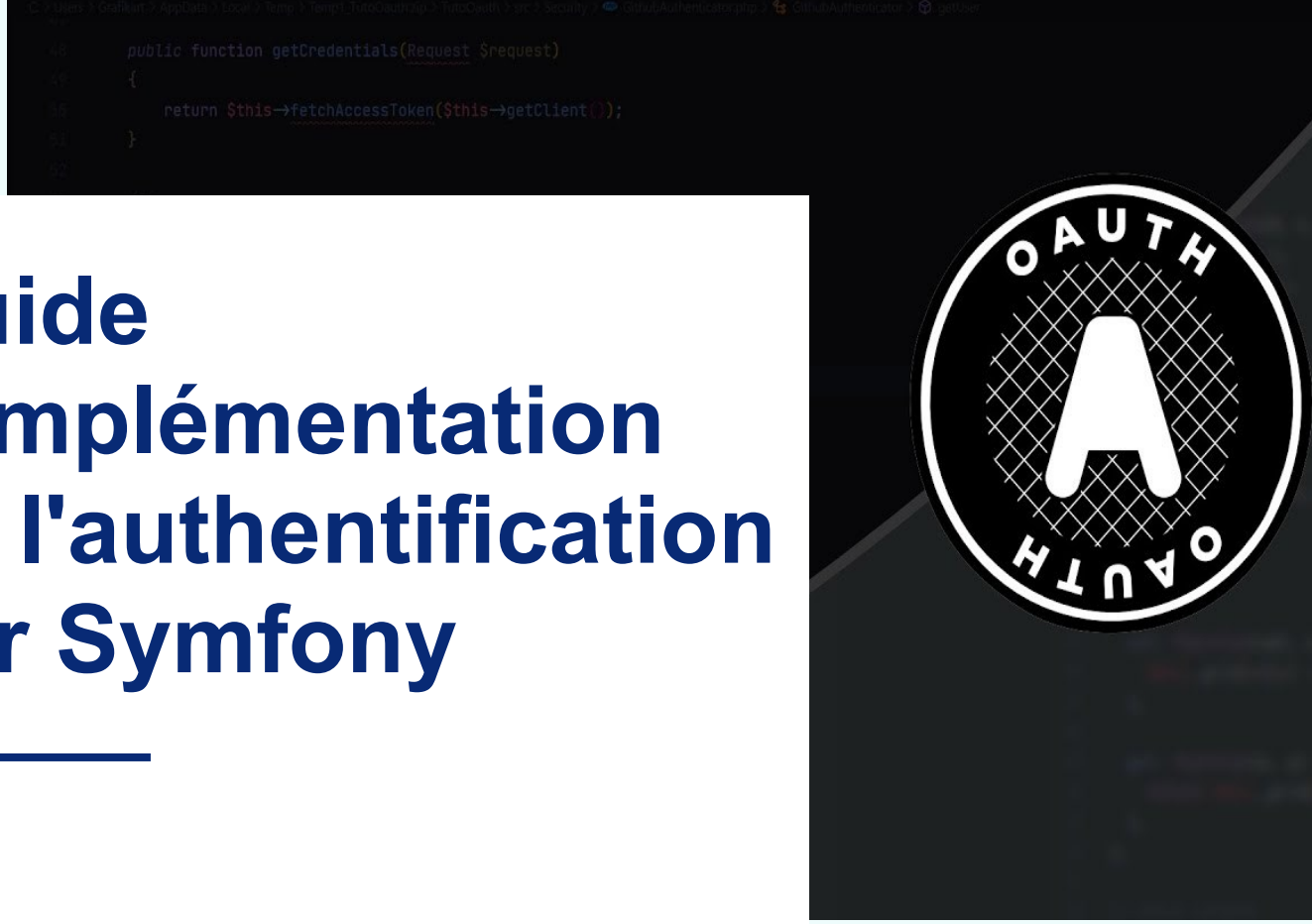


Guide d'implémentation de l'authentification sur Symfony

26 SEPTEMBRE

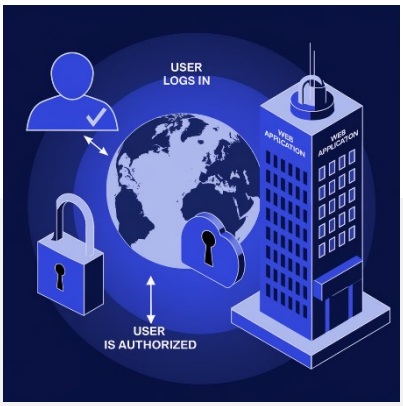
Getssone

Créé par : Solis Gaëtan

Getssone Certified

Introduction

L'authentification est le processus par lequel un utilisateur est vérifié et autorisé à accéder à une application web.



Comprendre l'authentification sur Symfony

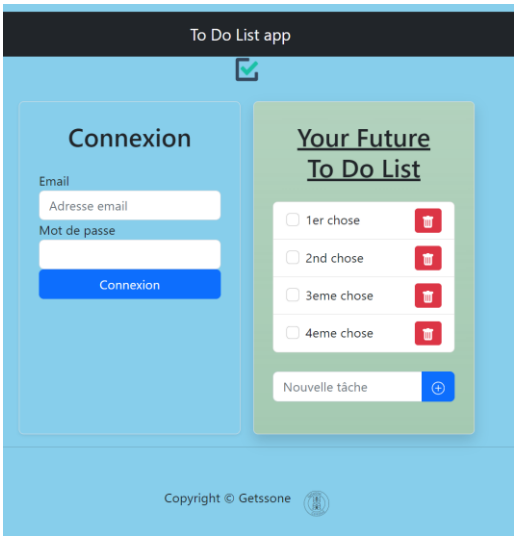
Dans le contexte de Symfony, l'authentification est gérée par le composant de sécurité, qui fournit une infrastructure pour gérer l'authentification, l'autorisation et la gestion des utilisateurs

Il fonctionne ainsi :

1. L'utilisateur soumet un formulaire de connexion avec ses informations d'identification (nom d'utilisateur et mot de passe). (cf.screen 1)
2. Le contrôleur d'authentification reçoit les informations d'identification et les transmet au gestionnaire d'authentification. (cf.screen 2)
3. Le gestionnaire d'authentification vérifie les informations d'identification de l'utilisateur en les comparant aux informations d'identification stockées dans la base de données. (cf.screen 3)
4. Si les informations d'identification sont valides, le gestionnaire d'authentification crée un objet utilisateur authentifié et le transmet au gestionnaire de sécurité. (cf.screen 3)

Le gestionnaire de sécurité stocke l'objet utilisateur authentifié dans la session de l'utilisateur et autorise l'accès à l'application.

Screen 1



Screen 2

```php
#[Route(path: '/login', name: 'login', methods: ['GET', 'POST'])]
public function login(AuthenticationUtils $authenticationUtils): Response
{
    // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();

    // last username entered by the user
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render('pages/security/login.html.twig', [
        'last_username' => $lastUsername,
        'error' => $error,
    ]);
}
```

Screen 3

```php
public function getLastUsername(): string
{
    $request = $this->getRequest();

    if ($request->attributes->has(SecurityRequestAttributes::LAST_USERNAME)) {
        return $request->attributes->get(SecurityRequestAttributes::LAST_USERNAME) ?? '';
    }

    return $request->hasSession() ? ($request->getSession()->get(SecurityRequestAttributes::LAST_USERNAME) ?? '') : '';
}

/**
 * @throws \LogicException
 */
private function getRequest(): Request
{
    $request = $this->requestStack->getCurrentRequest();

    if (null === $request) {
        throw new \LogicException('Request should exist so it can be processed for error.');
    }

    return $request;
}
```

Fichiers à modifier

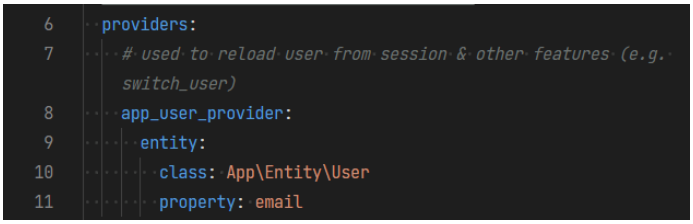
Pour configurer le processus d'authentification, définir les règles d'accès aux ressources protégées et gérer les sessions d'utilisateur. Les fichiers principaux à modifier sont les suivants :

1. Fichier de routage (**config/route.yaml**) :
    - Le fichier de routage définit les URL auxquelles les utilisateurs peuvent accéder et les contrôleurs qui gèrent les requêtes correspondantes.
    - Lors de l'implémentation de l'authentification, il est nécessaire de définir des routes pour le formulaire de connexion et de déconnexion.

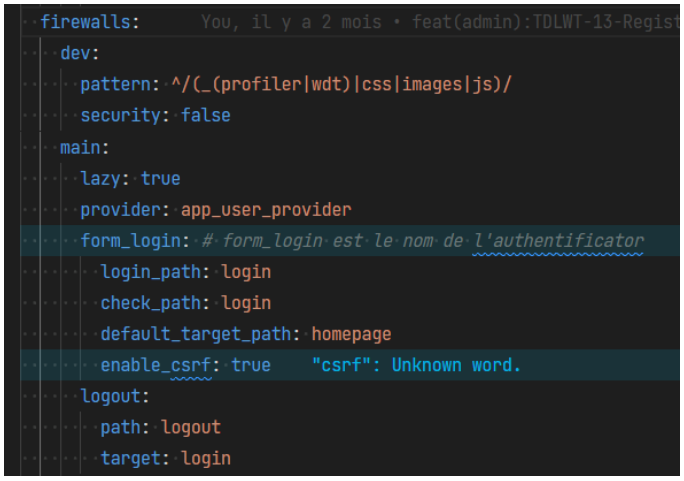
2. Le fichier de sécurité (**security.yaml**) est un fichier de configuration utilisé par Symfony pour gérer la sécurité de l'application, y compris l'authentification, l'autorisation et la gestion des utilisateurs.

   Voici quelques-unes des sections les plus importantes du fichier de sécurité :

   - **providers** : Cette section définit les fournisseurs utilisés pour récupérer les utilisateurs de l'application. Par exemple, ici on définit un fournisseur d'utilisateurs basé sur notre base de données mais on aurait pu y définir un service externe.

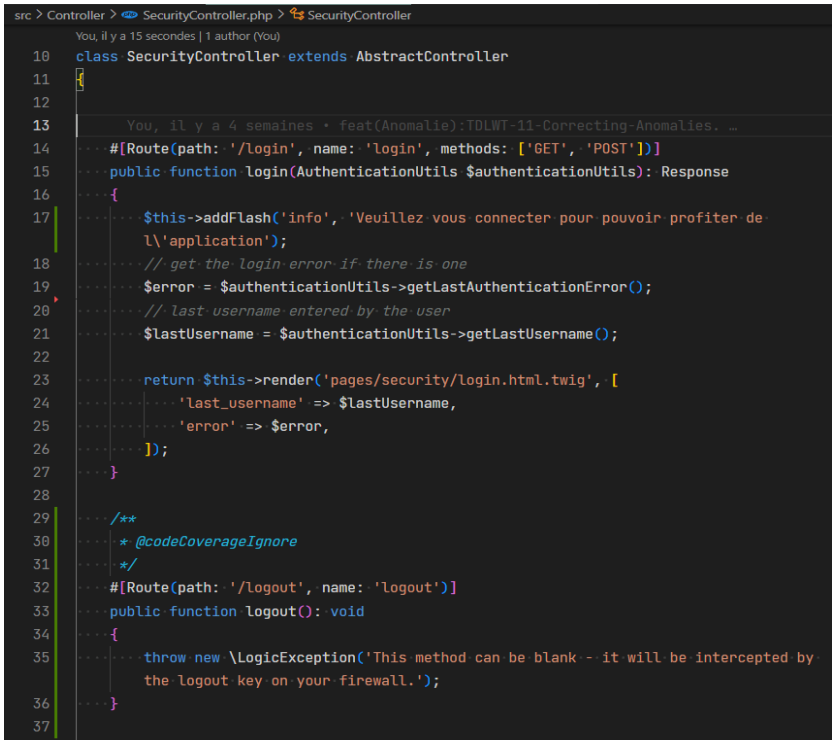


- **firewalls** : Cette section définit les règles de sécurité pour différentes parties de l'application. Chaque règle de sécurité peut spécifier des règles d'accès, des méthodes d'authentification et des options de gestion des sessions.

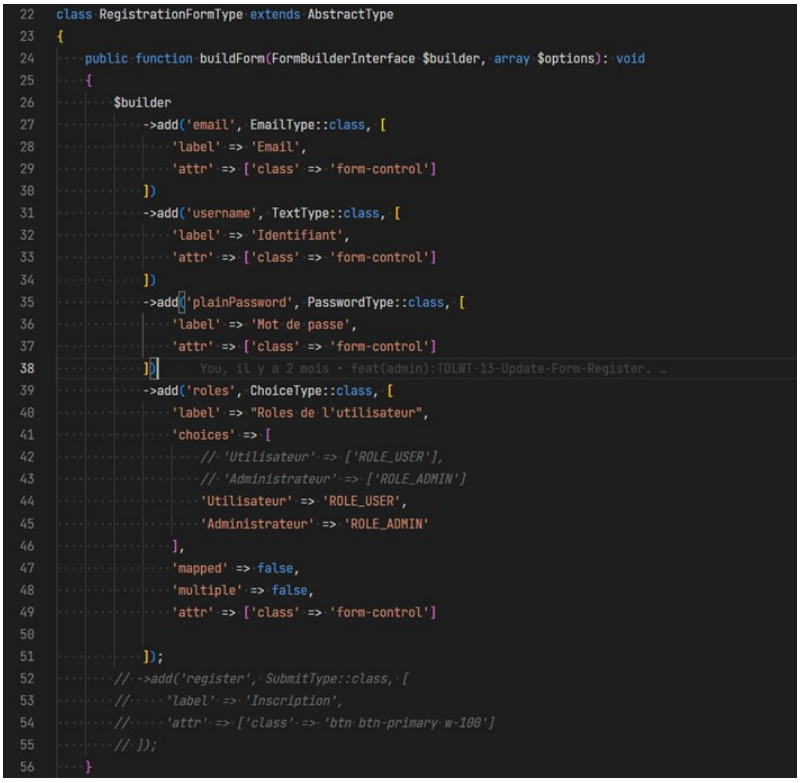


- **access_control** : Cette section définit les règles de contrôle d'accès pour différentes parties de l'application. Chaque règle de contrôle d'accès peut spécifier les rôles requis pour accéder à une URL ou à une ressource.

3. Contrôleur d'authentification (**src/Controller/SecurityController.php**) :
    - Le contrôleur d'authentification gère les requêtes liées à l'authentification, telles que la connexion et de déconnexion.
    - Il est nécessaire de créer un contrôleur d'authentification personnalisé pour gérer les requêtes et interagir avec le gestionnaire d'authentification.



4. Formulaire de connexion (**src/Form/LoginFormType.php**) :
    - Le formulaire de connexion permet aux utilisateurs de saisir leurs informations d'identification (nom d'utilisateur et mot de passe) pour se connecter à l'application.
    - Il est nécessaire de créer un formulaire de connexion pour gérer la validation des informations d'identification et l'interaction avec le gestionnaire d'authentification.

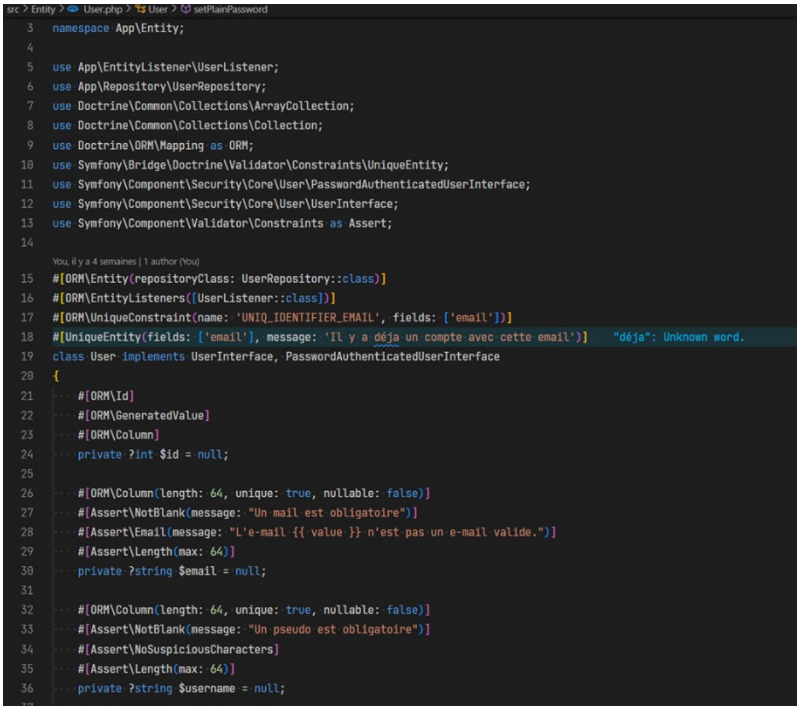


```php
class RegistrationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('email', EmailType::class, [
                'label' => 'Email',
                'attr' => ['class' => 'form-control']
            ])
            ->add('username', TextType::class, [
                'label' => 'Identifiant',
                'attr' => ['class' => 'form-control']
            ])
            ->add('plainPassword', PasswordType::class, [
                'label' => 'Mot de passe',
                'attr' => ['class' => 'form-control']
            ])
            ->add('roles', ChoiceType::class, [
                'label' => "Roles de l'utilisateur",
                'choices' => [
                    // 'Utilisateur' => ['ROLE_USER'],
                    // 'Administrateur' => ['ROLE_ADMIN']
                    'Utilisateur' => 'ROLE_USER',
                    'Administrateur' => 'ROLE_ADMIN'
                ],
                'mapped' => false,
                'multiple' => false,
                'attr' => ['class' => 'form-control']

            ]);
        // ->add('register', SubmitType::class, [
        //     'label' => 'Inscription',
        //     'attr' => ['class' => 'btn btn-primary w-100']
        // ]);
    }
```

4. Bis. Formulaire de connexion (**templates/pages/security/login.html.php**) :
    - Le formulaire de connexion permet aux utilisateurs de saisir leurs informations d'identification (nom d'utilisateur et mot de passe) pour se connecter à l'application.
    - Le formulaire de connexion est implémenté à l'aide de l'élément <form> HTML.
    - L'attribut action renvoie au contrôleur SecurityController vers la méthode login.
    - Cette méthode donne une maitrise plus granulaire.

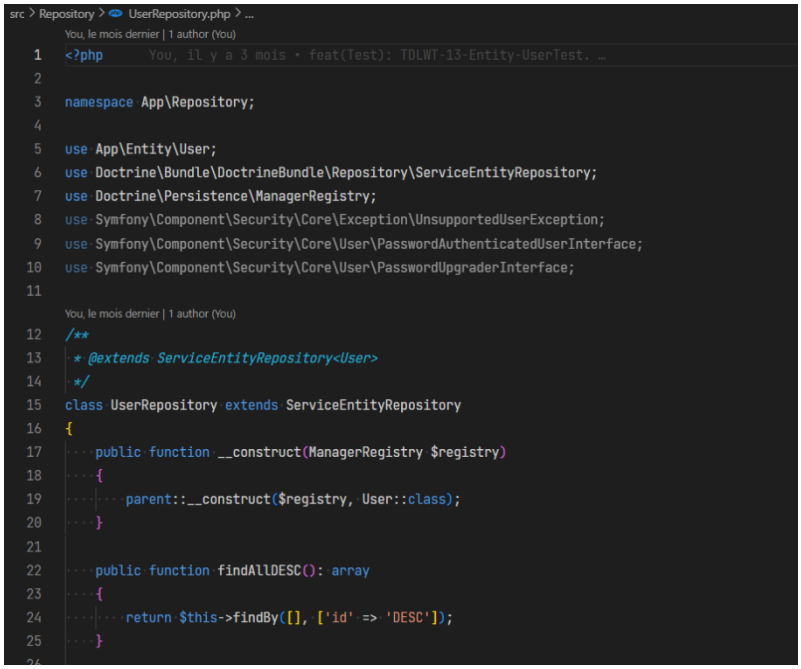
```twig
<form name="login" action="{{ path('login') }}" method="post" class="w-50 p-4 border rounded
shadow-sm">      You, il y a 2 mois • fix(Controller) Register and Security. …
    <h2 class="text-center mb-4">Connexion</h2>
    {% if error %}
        <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</
        div>
    {% endif %}


    <label for="username">Email</label>
    <input type="email" value="{{ last_username }}" name="_username" id="username"
    class="form-control" autocomplete="email" placeholder="Adresse email" required autofocus>
    <label for="password">Mot de passe</label>
    <input type="password" name="_password" id="password" class="form-control"
    autocomplete="current-password" required>

    <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">

    <button type="submit" class="btn btn-primary w-100">Connexion</button>
</form>
```

5. Entité utilisateur (**src/Entity/User.php**) :
    - L'entité utilisateur représente les utilisateurs de l'application et leurs informations associées, telles que le nom d'utilisateur, le mot de passe et le rôle.
    - Il est nécessaire de créer une entité utilisateur pour stocker les informations des utilisateurs dans la base de données.



```php
src > Entity > User.php > User > setPlainPassword
namespace App\Entity;

use App\EntityListener\UserListener;
use App\Repository\UserRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Validator\Constraints as Assert;

You, il y a 4 semaines | 1 author (You)
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ORM\EntityListeners([UserListener::class])]
#[ORM\UniqueConstraint(name: 'UNIQ_IDENTIFIER_EMAIL', fields: ['email'])]
#[UniqueEntity(fields: ['email'], message: 'Il y a déja un compte avec cette email')]    "déja": Unknown word.
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 64, unique: true, nullable: false)]
    #[Assert\NotBlank(message: "Un mail est obligatoire")]
    #[Assert\Email(message: "L'e-mail {{ value }} n'est pas un e-mail valide.")]
    #[Assert\Length(max: 64)]
    private ?string $email = null;

    #[ORM\Column(length: 64, unique: true, nullable: false)]
    #[Assert\NotBlank(message: "Un pseudo est obligatoire")]
    #[Assert\NoSuspiciousCharacters]
    #[Assert\Length(max: 64)]
    private ?string $username = null;
```

6. Gestionnaire d'utilisateurs (**src/Repository/UserRepository.php**) :
    - Le gestionnaire d'utilisateurs permet de récupérer et de gérer les utilisateurs de l'application dans la base de données.
    - Il est nécessaire de créer un gestionnaire d'utilisateurs pour récupérer les utilisateurs par leur nom d'utilisateur, vérifier les informations d'identification et gérer les sessions d'utilisateur.



Stockage des utilisateurs

Lors de la création d'une entité utilisateur, il est important de définir les propriétés nécessaires pour stocker les informations de l'utilisateur. L'entité utilisateur peut avoir des propriétés telles que username, password, email, roles, etc.

Il est important de stocker les mots de passe de manière sécurisée. Pour cela on utilise des outils de **hachage** de mots de passe, ce qui permet de stocker les mots de passe de manière sécurisée dans la base de données.

A cela on ajoute la validation des données entrée de l'utilisateur, grâce aux **Constraints** de Symfony ce qui permet de garantir l'intégrité des données stockées dans la base de données.

Conclusion

En résumé l'implémentation de l'authentification est un processus important pour garantir la sécurité et la confidentialité des applications web.

Pour implémenter l'authentification, il est nécessaire de comprendre :

- Le processus d'authentification
- Les composants impliqués,
- Les fichiers principaux
- Savoir comment on stocke les utilisateurs de manière sécurisée et de gérer les sessions d'utilisateur de manière sécurisée.

En suivant ces étapes, il est possible d'implémenter une authentification sécurisée.