Advanced Symfony Security

SymfonyLive Berlin 2019

SensioLabs

Wer sind wir?

Marco Petersen
Software Developer

Simon Mönch
Software Developer

Das Repository

`https://github.com/ocrampete16/advanced-symfony-security-workshop`

➡ Übungsaufgaben

➡ diese Folien

Grundlagen

Authentifizierung und Autorisierung

Aufgabe des Security-Layers ist es, User zu verwalten.

Aufgabe des Security-Layers ist es, ~~User zu verwalten.~~

Ressourcen vor unerlaubtem Zugriff zu schützen.

Wer bist du?

Darfst du das?

Wer bist du? ➡ **Authentifizierung**

Darfst du das? ➡ **Autorisierung**

Token als „Ausweisdokument“

Der User legt ein Token (Ausweisdokument) vor.

Das Token wird authentifiziert.
(Ausweisdokument wird beglaubigt)

Mit einem authentifizierten Token kann die Autorisierung gestartet werden.

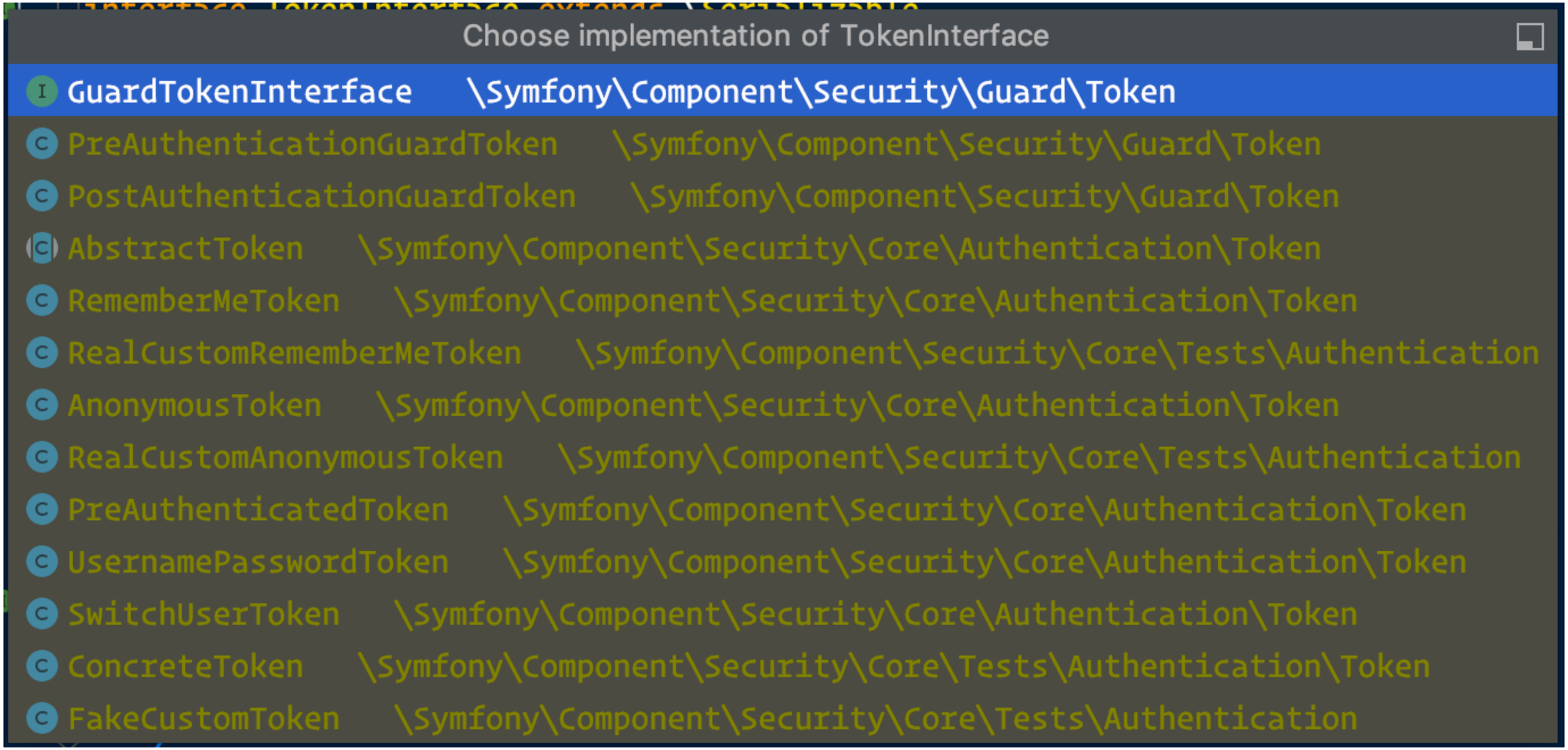
Grundlagen

Die Security-Komponente ohne das Framework

Anhand eines Tokens identifizieren sich User

```
$token = new UsernamePasswordToken(
    'marco', 'p4$$w0rd', PROVIDER_KEY
);

$token = $authenticationManager->authenticate($token);
```

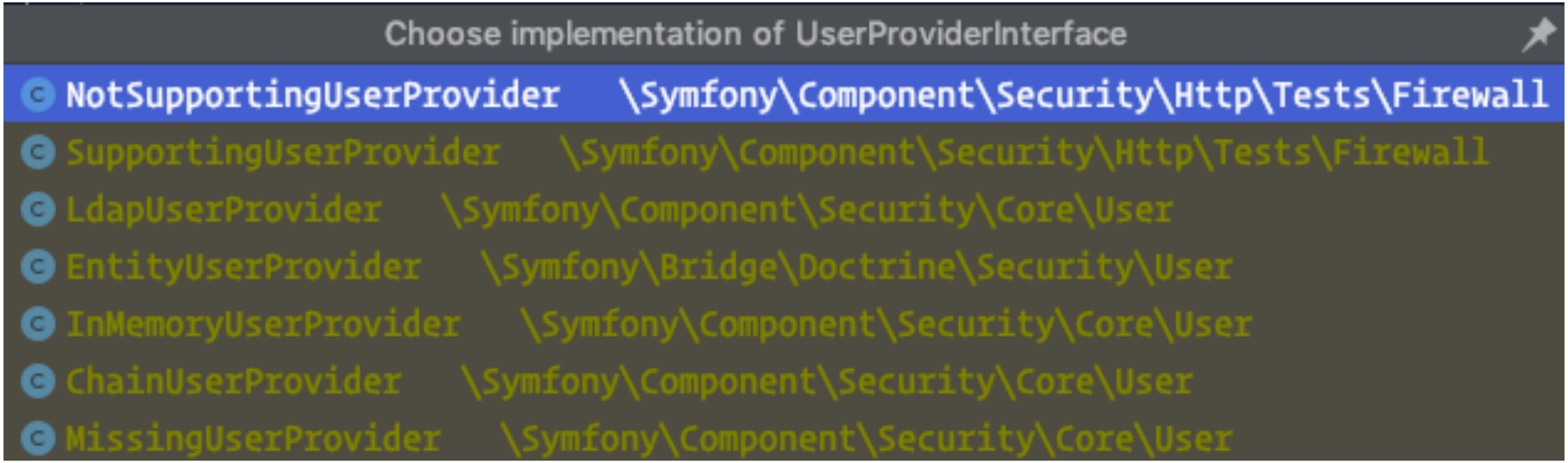
```php
interface TokenInterface extends \Serializable
{
    public function getRoles();
    public function getCredentials();
    public function getUser();
    public function setUser($user);
    public function getUsername();
    public function isAuthenticated();
    public function setAuthenticated($isAuthenticated);
    public function eraseCredentials();
    public function getAttributes();
    public function setAttributes(array $attributes);
    public function hasAttribute($name);
    public function getAttribute($name);
    public function setAttribute($name, $value);
}
```


Choose implementation of TokenInterface
GuardTokenInterface \Symfony\Component\Security\Guard\Token
PreAuthenticationGuardToken \Symfony\Component\Security\Guard\Token
PostAuthenticationGuardToken \Symfony\Component\Security\Guard\Token
AbstractToken \Symfony\Component\Security\Core\Authentication\Token
RememberMeToken \Symfony\Component\Security\Core\Authentication\Token
RealCustomRememberMeToken \Symfony\Component\Security\Core\Tests\Authentication
AnonymousToken \Symfony\Component\Security\Core\Authentication\Token
RealCustomAnonymousToken \Symfony\Component\Security\Core\Tests\Authentication
PreAuthenticatedToken \Symfony\Component\Security\Core\Authentication\Token
UsernamePasswordToken \Symfony\Component\Security\Core\Authentication\Token
SwitchUserToken \Symfony\Component\Security\Core\Authentication\Token
ConcreteToken \Symfony\Component\Security\Core\Tests\Authentication\Token
FakeCustomToken \Symfony\Component\Security\Core\Tests\Authentication

User-Provider laden User aus einer Quelle

```
$userProvider = new InMemoryUserProvider([
    'marco' => [
        'password' => 'p4$$w0rd',
        'roles' => ['ROLE_USER'],
    ],
]);

$user = $userProvider->loadUserByUsername('marco');
```

```php
interface UserProviderInterface
{
    public function loadUserByUsername($username);

    public function refreshUser(UserInterface $user);

    public function supportsClass($class);
}
```


Choose implementation of UserProviderInterface
NotSupportingUserProvider \Symfony\Component\Security\Http\Tests\Firewall
SupportingUserProvider \Symfony\Component\Security\Http\Tests\Firewall
LdapUserProvider \Symfony\Component\Security\Core\User
EntityUserProvider \Symfony\Bridge\Doctrine\Security\User
InMemoryUserProvider \Symfony\Component\Security\Core\User
ChainUserProvider \Symfony\Component\Security\Core\User
MissingUserProvider \Symfony\Component\Security\Core\User

Der User-Checker überprüft, dass User bestimmte Eigenschaften (nicht) vorweisen

```
$userChecker = new UserChecker();

$userChecker->checkPreAuth($user);

// authenticate user here

$userChecker->checkPostAuth($user);
```

```php
interface UserCheckerInterface
{
    /**
     * Checks the user account before authentication.
     *
     * @throws AccountStatusException
     */
    public function checkPreAuth(UserInterface $user);

    /**
     * Checks the user account after authentication.
     *
     * @throws AccountStatusException
     */
    public function checkPostAuth(UserInterface $user);
}
```

Worauf prüft der User-Checker?

Standardmäßig führt der User-Checker selbst keine Prüfung durch.

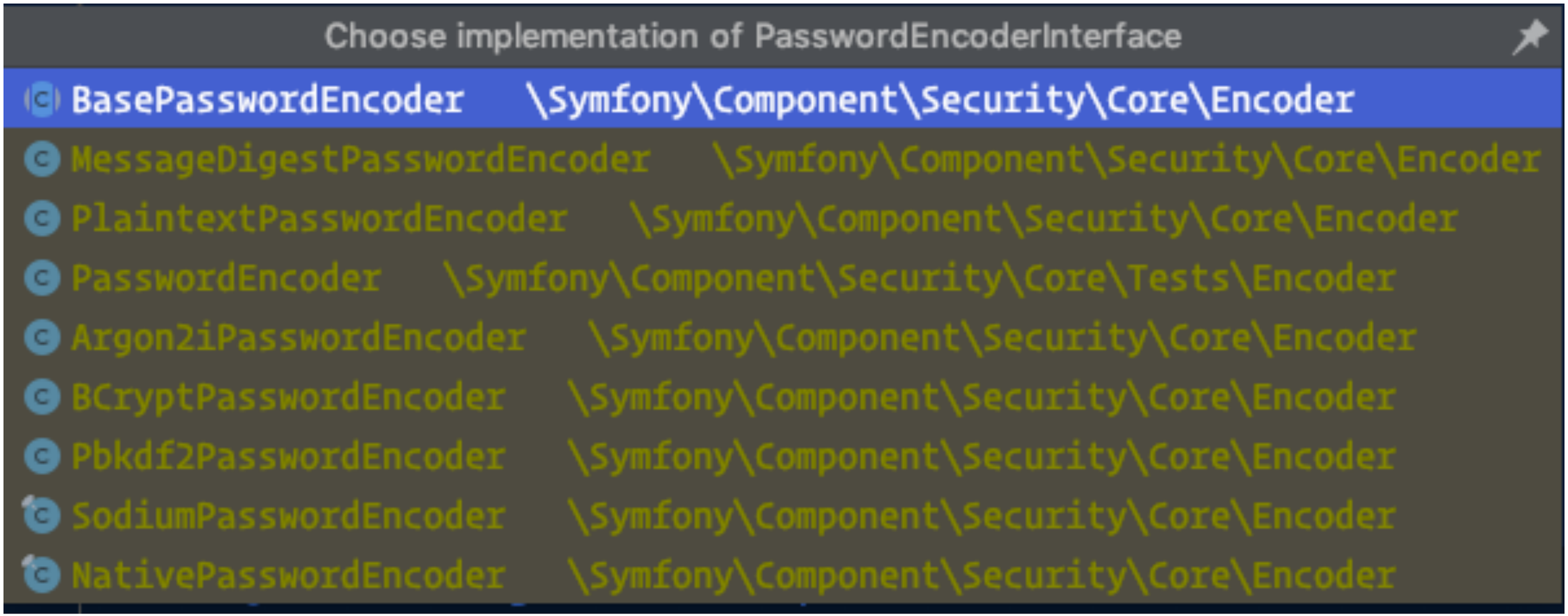
Er bietet euch einen Hook für eure eigene Logik:

- gesperrte User
- abgelaufene Credentials
- User nach der kostenlosen Probeperiode

Password-Encoder generieren Hashes

```
$encoderFactory = new EncoderFactory([
    User::class => new PlaintextPasswordEncoder(),
]);

$encoder = $encoderFactory->getEncoder($user);

$hash = $encoder->encodePassword($plaintext, $salt);
```

```php
interface PasswordEncoderInterface
{
    public function encodePassword($raw, $salt);

    public function isPasswordValid($encoded, $raw, $salt);
}
```


Choose implementation of PasswordEncoderInterface
BasePasswordEncoder \Symfony\Component\Security\Core\Encoder
MessageDigestPasswordEncoder \Symfony\Component\Security\Core\Encoder
PlaintextPasswordEncoder \Symfony\Component\Security\Core\Encoder
PasswordEncoder \Symfony\Component\Security\Core\Tests\Encoder
Argon2iPasswordEncoder \Symfony\Component\Security\Core\Encoder
BCryptPasswordEncoder \Symfony\Component\Security\Core\Encoder
Pbkdf2PasswordEncoder \Symfony\Component\Security\Core\Encoder
SodiumPasswordEncoder \Symfony\Component\Security\Core\Encoder
NativePasswordEncoder \Symfony\Component\Security\Core\Encoder

sodium, native und auto

```yaml
# config/packages/security.yaml
security:
    # ...
    encoders:
        App\Entity\User:
            # ehemals 'argon2i'
            algorithm: 'sodium'
            # nimmt den besten vorhandenen Algorithmus
            algorithm: 'native'
            # nimmt 'sodium' wenn möglich, sonst 'native'
            algorithm: 'auto'
```

Wie kommt das alles zusammen?

```
const PROVIDER_KEY = 'default';

$unauthenticatedToken = new UsernamePasswordToken('marco', 'p4$$w0rd', PROVIDER_KEY);

$authenticationManager = new AuthenticationProviderManager([
    new DaoAuthenticationProvider(
        $userProvider, $userChecker, PROVIDER_KEY, $encoderFactory
    ),
]);

$authenticatedToken = $authenticationManager->authenticate($unauthenticatedToken);
```

(Nicht) Authentifizierte Tokens

```php
// UserPasswordToken.php
public function __construct(
    $user, $credentials, string $providerKey, array $roles = []
) {
    // ...

    parent::setAuthenticated(\count($roles) > 0);
}
```

```
$userProvider = new InMemoryUserProvider([
    'marco' => [
        'password' => 'p4$$w0rd',
        'roles' => ['ROLE_USER'],
    ],
]);
```

```php
$userProvider = new InMemoryUserProvider([
    'marco' => [
        'password' => 'p4$$w0rd',
        'roles' => [],
    ],
]);
```

```php
$userProvider = new InMemoryUserProvider([
    'marco' => [
        'password' => 'p4$$w0rd',
        'roles' => [],
    ],
]);

// $token gilt immer noch als nicht authentifiziert!
$token = $authenticationManager->authenticate($unauthenticatedToken);
```

Rollenbasierte Autorisierung (1)

Bist du `<Rolle>`, darfst du diese Seite aufrufen.

Beispiele:

- Bist du Admin, darfst du diese Seite aufrufen.
- Bist du ein eingeloggter User, darfst du diese Seite aufrufen.

Rollenbasierte Autorisierung (2)

```
$accessDecisionManager = new AccessDecisionManager([
    new RoleVoter('ROLE_'),
    new AuthenticatedVoter($authenticationTrustResolver),
]);

// Besitzt der User die Rolle `ROLE_ADMIN`?
$accessDecisionManager->decide(
    $authenticatedToken, ['ROLE_ADMIN']
);

// Ist der User eingeloggt?
$accessDecisionManager->decide(
    $authenticatedToken, ['IS_AUTHENTICATED_FULLY']
);
```

Aufgabe: die Security-Komponente standalone einsetzen

Bringt das Skript wieder zum Laufen!

Die Firewall

Was ist eine Firewall?

Demo-App ohne Firewall (1)

Seiten lassen sich aufrufen.

Sobald man eine der Security-Services benutzt, schlägt Symfony Alarm.

Aber warum?

Demo-App ohne Firewall (2)

Weil der Security Bundle gar nicht erst in den Request-Response-Zyklus eingreifen konnte!

➡ Die Firewall ist der Entrypoint für Symfonys Security-Logik.

Demo-App mit Firewall und form_login (1)

Nur authentifizierte User dürfen Seiten hinter der Firewall sehen.

login_path ➡ GET-Endpoint, um die Login-Seite aufzurufen

check_path ➡ POST-Endpoint für den Daten-Submit

Henne-Ei-Problem: beide Endpoints sind hinter der Firewall

Demo-App mit Firewall und form_login (2)

Henne-Ei-Problem: beide Endpoints sind hinter der Firewall

Lösung:

- anonymous: true
- Zugriffsberechtigung über access_control regeln

Die Firewall und access_control

Die Security-Firewall ist keine herkömmliche Firewall.

Sie legt den Zuständigkeitsbereich des Security-Layers fest.

Mittels access_control kann man geschützte Teilbereiche definieren.

Mehrere Firewalls in einer Anwendung

In den meisten Fällen reicht eine einzige Firewall.

Die Konfiguration bietet euch genug Flexibilität.

Mehrere Firewalls ➡ mehrere autarke Security-Systeme

z.B. Webseite (Form Login mit Session) + API (JWT, stateless)

Debugging mit der Konfiguration

```
bin/console config:dump-reference security
```

gibt alle möglichen Konfigurationsschüssel mit Standardwerten aus

```
bin/console debug:config security
```

gibt die aktuelle Konfiguration aus

Aufgabe: Admin-Bereich absichern

Richtet eine Firewall ein und setzt folgende Anforderungen um:

- **Normale User** (ROLE_USER) dürfen sich einloggen.
- Nur **Admin-User** (ROLE_ADMIN) dürfen neue Posts verfassen.

Authentifizierung über den

Guard-Authenticator

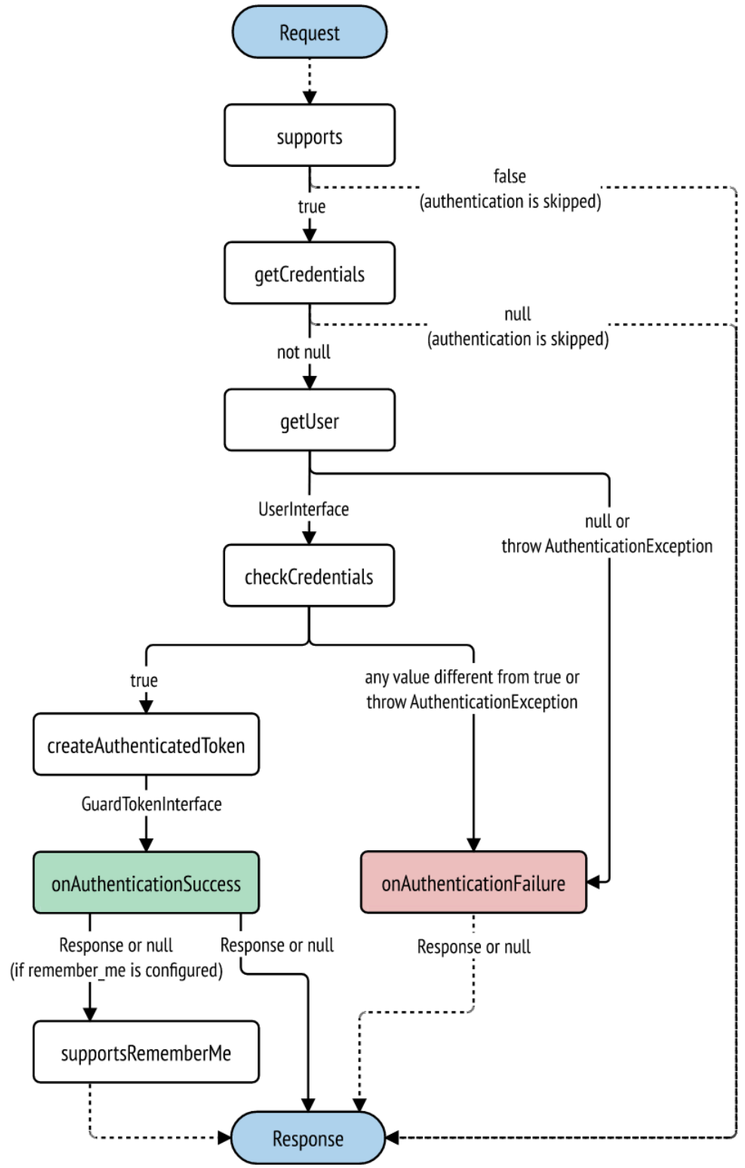
Was ist ein Guard-Authenticator?

Mit dem Guard-Authenticator kann man seinen eigenen Authentifizierungslayer implementieren.

Die komplette Logik bleibt in einem einzigen Objekt (dem Authenticator) gekapselt.

Dabei behält man komplett die Kontrolle über den Authentifizierungsprozess.

```php
interface AuthenticatorInterface extends AuthenticationEntryPointInterface
{
    public function supports(Request $request);

    public function getCredentials(Request $request);

    public function getUser($credentials, UserProviderInterface $userProvider);

    public function checkCredentials($credentials, UserInterface $user);

    public function createAuthenticatedToken(UserInterface $user, $providerKey);

    public function onAuthenticationFailure(
        Request $request, AuthenticationException $exception
    );

    public function onAuthenticationSuccess(
        Request $request, TokenInterface $token, $providerKey
    );

    public function supportsRememberMe();
}
```


Request
supports
false
(authentication is skipped)
true
getCredentials
null
(authentication is skipped)
not null
getUser
UserInterface
null or
throw AuthenticationException
checkCredentials
true
any value different from true or
throw AuthenticationException
createAuthenticatedToken
GuardTokenInterface
onAuthenticationSuccess
onAuthenticationFailure
Response or null
(if remember_me is configured)
Response or null
Response or null
supportsRememberMe
Response

Aufgabe: form_login durch einen Guard-Authenticator ersetzen

Ersetzt die `form_login` Konfiguration durch einen eigenen Guard-Authenticator.

Hello

Rollenbasierte Autorisierung

my old friend

Wie setzt man sie ein?

access_control
➡ ganze Bereiche eurer Webseite

@Security Annotation
➡ Controller oder einzelne Actions

Authorization-Checker
➡ für den Einsatz in Services

Die @Security Annotation (1)

```php
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;
// ...

/**
 * @Route("/new", name="admin_post_new")
 * @Security("has_role('ROLE_ADMIN')")
 */
public function new()
{
    // ...
}
```

Die @Security Annotation (2)

```
use App\Entity\Post;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;

/**
 * @Route("/{id}/edit", name="admin_post_edit")
 * @Security("user.getEmail() == post.getAuthorEmail()")
 */
public function edit(Post $post)
{
    // ...
}
```

Expressions können nicht wiederverwendet werden ...

```
use App\Entity\Post;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;

/**
 * @Route("/{id}/edit", name="admin_post_edit")
 * @Security("user.getEmail() == post.getAuthorEmail()")
 */
public function edit(Post $post) { ... }
```

```
{% if app.user and app.user.email == post.authorEmail %}
    <a href=""> ... </a>
{% endif %}
```

... Domain-Logik hingegen schon (1)

```php
// src/Entity/Post.php
// ...

class Post
{
    // ...

    /**
     * Is the given User the author of this Post?
     */
    public function isAuthor(User $user = null)
    {
        return $user && $user->getEmail() == $this->getAuthorEmail();
    }
}
```

... Domain-Logik hingegen schon (2)

```php
use App\Entity\Post;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;

/**
 * @Route("/{id}/edit", name="admin_post_edit")
 * @Security("user.isAuthor(post)")
 */
public function edit(Post $post) { ... }
```

```twig
{% if app.user and app.user.isAuthor(post) %}
    <a href=""> ... </a>
{% endif %}
```

Aktionsbasierte Autorisierung mit

Security-Votern

Was macht ein Security-Voter? (1)

Security-Voter stimmen darüber ab, ob der User eine Aktion ausführen darf.

Darfst du `<Aktion>` (an `<Objekt>`) ausführen?

Beispiele:
- Darfst du diesen Post bearbeiten?
- Darfst du diese Nachricht verschicken?

Was macht ein Security-Voter? (2)

Bei jedem Autorisierungsvorgang stimmen alle Voter mit ab.

Nicht relevante Voter enthalten sich.

In den meisten Fällen beschließt nur ein Voter, ob die Aktion erlaubt oder abgelehnt wird.

Access Decision Strategies

affirmative

➡ mindestens ein Voter muss die Aktion erlauben

consensus

➡ mehr Voter müssen die Aktion erlauben als zurückweisen

unanimous

➡ kein Voter darf die Aktion zurückweisen und ...

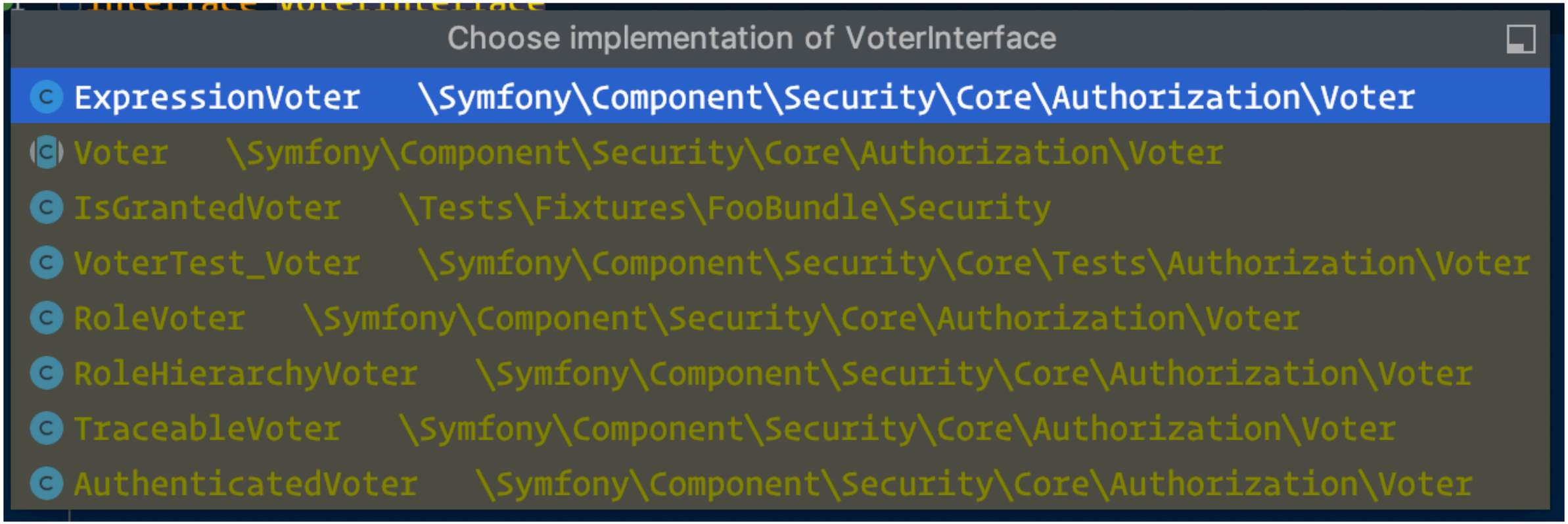
Unanimous

Kein Voter darf die Aktion zurückweisen und

`allow_if_all_abstain: false` (default)
➡ mindestens ein Voter muss die Aktion erlauben

`allow_if_all_abstain: true`
➡ es reicht, wenn sich alle enthalten

```php
abstract class Voter implements VoterInterface
{
    public function vote(TokenInterface $token, $subject, array $attributes)
    {
        $vote = self::ACCESS_ABSTAIN;
        foreach ($attributes as $attribute) {
            if (!$this->supports($attribute, $subject)) {
                continue;
            }
            $vote = self::ACCESS_DENIED;
            if ($this->voteOnAttribute($attribute, $subject, $token)) {
                return self::ACCESS_GRANTED;
            }
        }
        return $vote;
    }

    abstract protected function supports($attribute, $subject);

    abstract protected function voteOnAttribute($attribute, $subject, TokenInterface $token);
}
```


Choose implementation of VoterInterface
ExpressionVoter \Symfony\Component\Security\Core\Authorization\Voter
Voter \Symfony\Component\Security\Core\Authorization\Voter
IsGrantedVoter \Tests\Fixtures\FooBundle\Security
VoterTest_Voter \Symfony\Component\Security\Core\Tests\Authorization\Voter
RoleVoter \Symfony\Component\Security\Core\Authorization\Voter
RoleHierarchyVoter \Symfony\Component\Security\Core\Authorization\Voter
TraceableVoter \Symfony\Component\Security\Core\Authorization\Voter
AuthenticatedVoter \Symfony\Component\Security\Core\Authorization\Voter

Aufgabe: eigenen Security-Voter schreiben

Schreibt einen Security-Voter, welcher folgende Anforderungen erfüllt:

- Alle User dürfen ihre eigenen Posts bearbeiten.
- Admin-User dürfen alle von normalen Usern verfassten Posts bearbeiten.
- Ein User mit eurem Namen darf alle Posts bearbeiten 😉

Rollenbasierte und aktionsbasierte Autorisierung (1)

Die Autorisierung mit Rollen ist einfacher zu warten und verstehen und wird daher empfohlen.

Wo gilt diese Einschränkung?
➡ security.yaml / Controller / Service

Welche Voraussetzungen muss man erfüllen?
➡ Rolle besitzen

Rollenbasierte und aktionsbasierte Autorisierung (2)

Lassen sich die Anforderungen nicht mit Rollen umsetzen, sollte man auf aktionsbasierte Security-Voter zurückgreifen.

Wo gilt diese Einschränkung?
➡ Controller / Service
Welche Voraussetzungen muss man erfüllen?
➡ im Voter

Multi-Faktor-Authentifizierung

Disclaimer!

Dieses Beispiel soll exemplarisch aufzeigen, wie man MFA mithilfe von Guard implementieren könnte und ist nicht als Best Practice zu verstehen.

`github.com/scheb/two-factor-bundle`

`github.com/symfony/symfony/issues/30914`

Berechtigungsnachweise (Faktoren)

Knowledge ➡ physische Besitzobjekte (Handy, YubiKey)

Possession ➡ geheimes Wissen (PIN, Passwort)

Inherent ➡ eindeutige physische Merkmale (biometrische Daten)

Location ➡ Standort (Verbindung mit Netzwerk)

Authentifizierungsattribute (1)

`IS_AUTHENTICATED_ANONYMOUSLY` ➡
AnonymousToken

`IS_AUTHENTICATED_REMEMBERED` ➡
RememberMeToken

`IS_AUTHENTICATED_FULLY` ➡ !AnonymousToken && !
RememberMeToken

Der Plan (1)

Wir können "authentifiziert" und "nicht authentifiziert" abbilden, aber nicht "Authentifizierung laufend".

➡ `IS_AUTHENTICATED_PARTIALLY` einführen

➡ `IS_AUTHENTICATED_FULLY` soll nur gelten, wenn die Authentifizierung auch wirklich abgeschlossen ist

Der Plan (2)

➡ Pro Authentifizierungsschritt einen eigenen Guard-Authenticator implementieren
➡ Der erste Authenticator soll den User in den Zustand `IS_AUTHENTICATED_PARTIALLY` setzen
➡ Der letzte Authenticator soll den User in den Zustand `IS_AUTHENTICATED_FULLY` setzen
➡ Jeder Authenticator leitet auf den nächsten weiter

Aufgabe: Multi-Faktor-Authentifizierung implementieren

- Einen Security-Voter, der bei `IS_AUTHENTICATED_PARTIALLY` Zugriff gewährt.
- den aktuellen Guard-Authenticator anpassen
- einen neuen Guard-Authenticator für die Email-Code-Verifizierung

SensioLabs