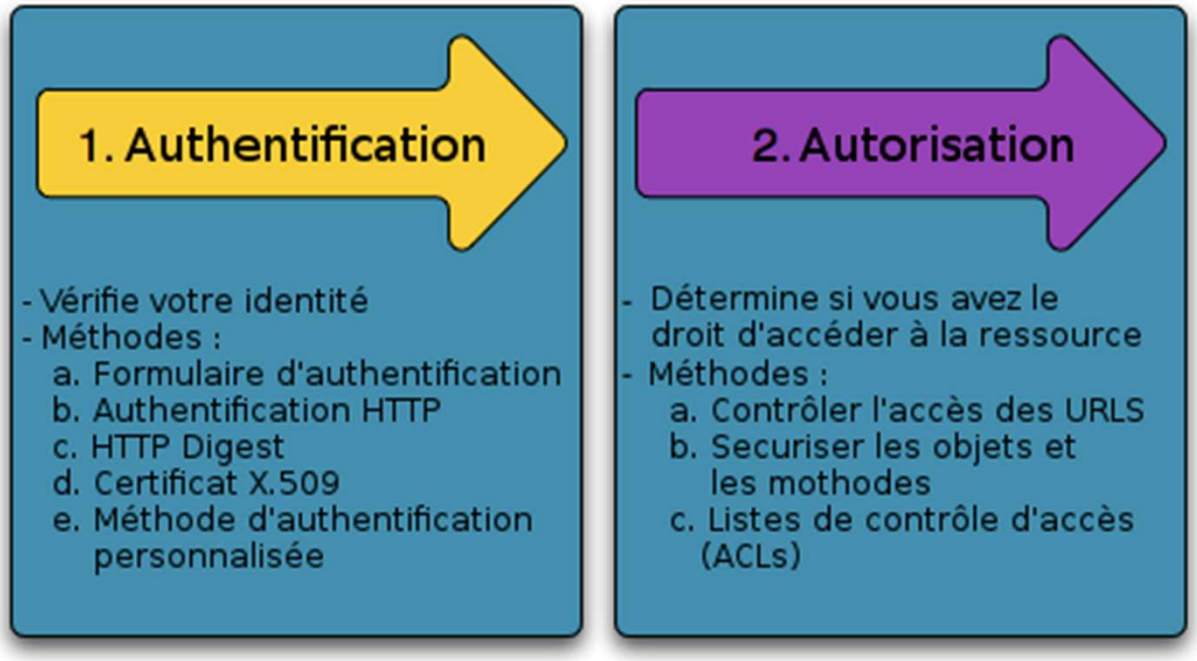
R4.01
Architecture Logicielle

Sécurité

https://symfony.com/doc/current/security.html

La sécurité : Authentification et Autorisation

- La sécurité est un processus en 2 étapes, dont le but est d'empêcher un utilisateur de consulter une ressource à laquelle il ne doit pas avoir accès.
- Dans la première étape, le système de sécurité cherche à savoir qui est l'utilisateur en lui demandant de fournir une identification. C'est **l'authentification**.
- Une fois que le système a identifié l'utilisateur, l'étape suivante est de déterminer s'il a le droit d'accéder à la ressource demandée. C'est **l'autorisation** : le système vérifie que vous avez les **privilèges,** ou le **rôle**, pour « exécuter » un contrôleur.



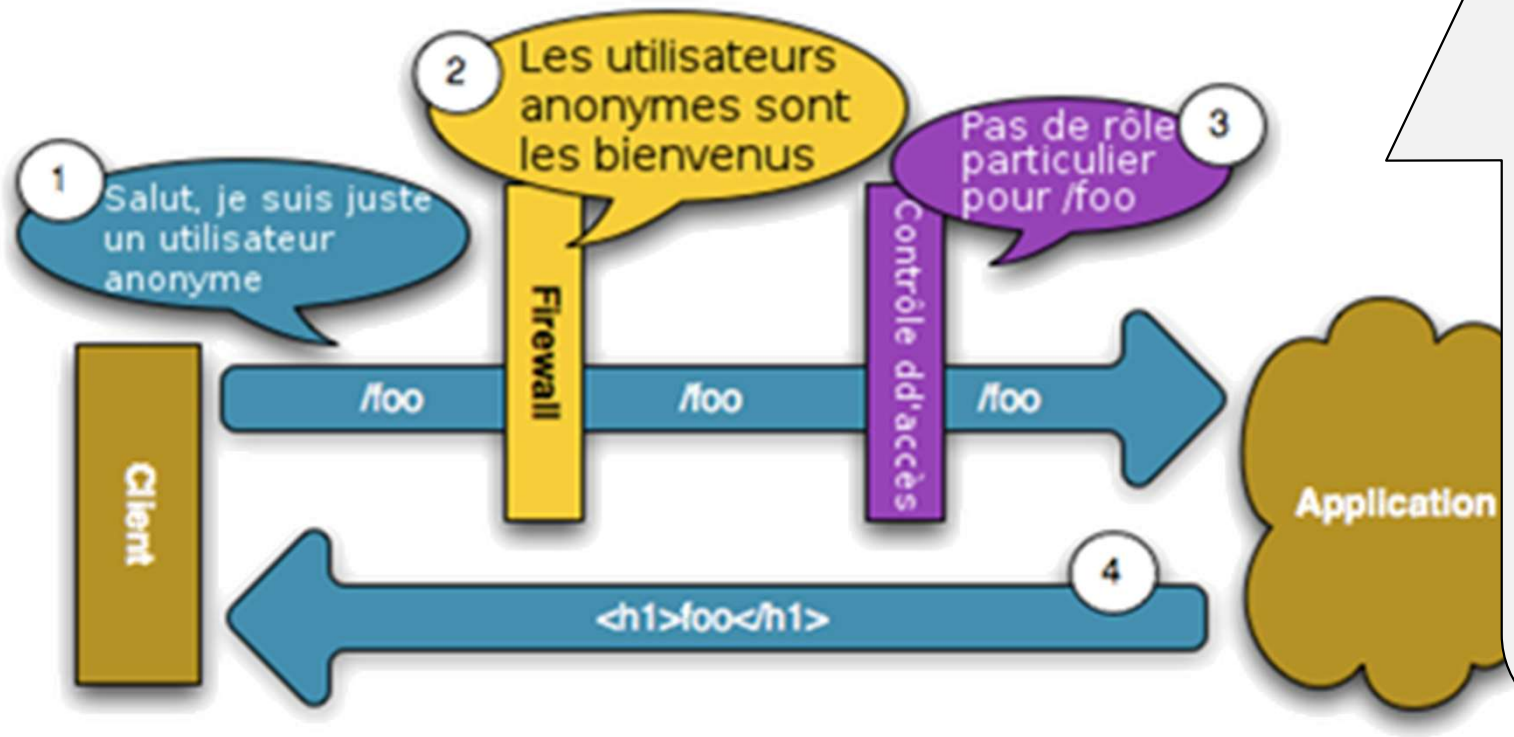
Pare-Feu : authentification (1)

- Un pare-feu, dans symfony, protège l'ensemble ou une partie des **URL** de l'application

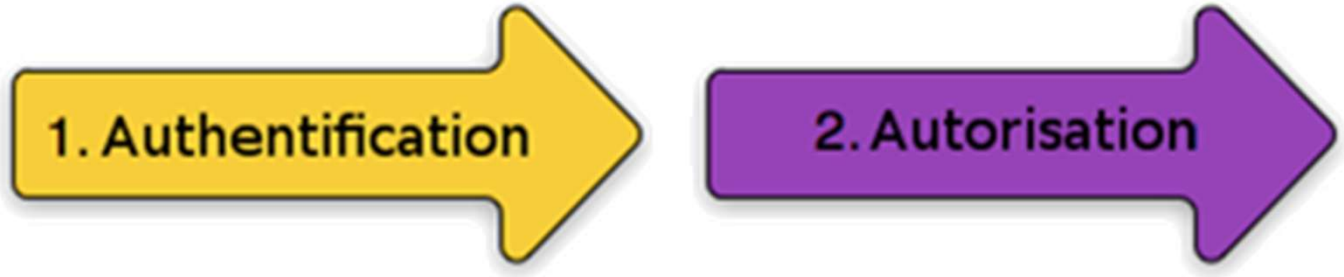
- Un pare-feu est activé dès lors que l'URL d'une requête correspond à un **pattern d'URL** (défini par une expression régulière) contenu dans la configuration du pare-feu

- Le rôle du pare-feu est de déterminer si un utilisateur doit ou ne doit pas être authentifié, et s'il doit l'être, de retourner une réponse à l'utilisateur afin d'entamer le processus d'authentification

Pare-Feu : authentification (2)



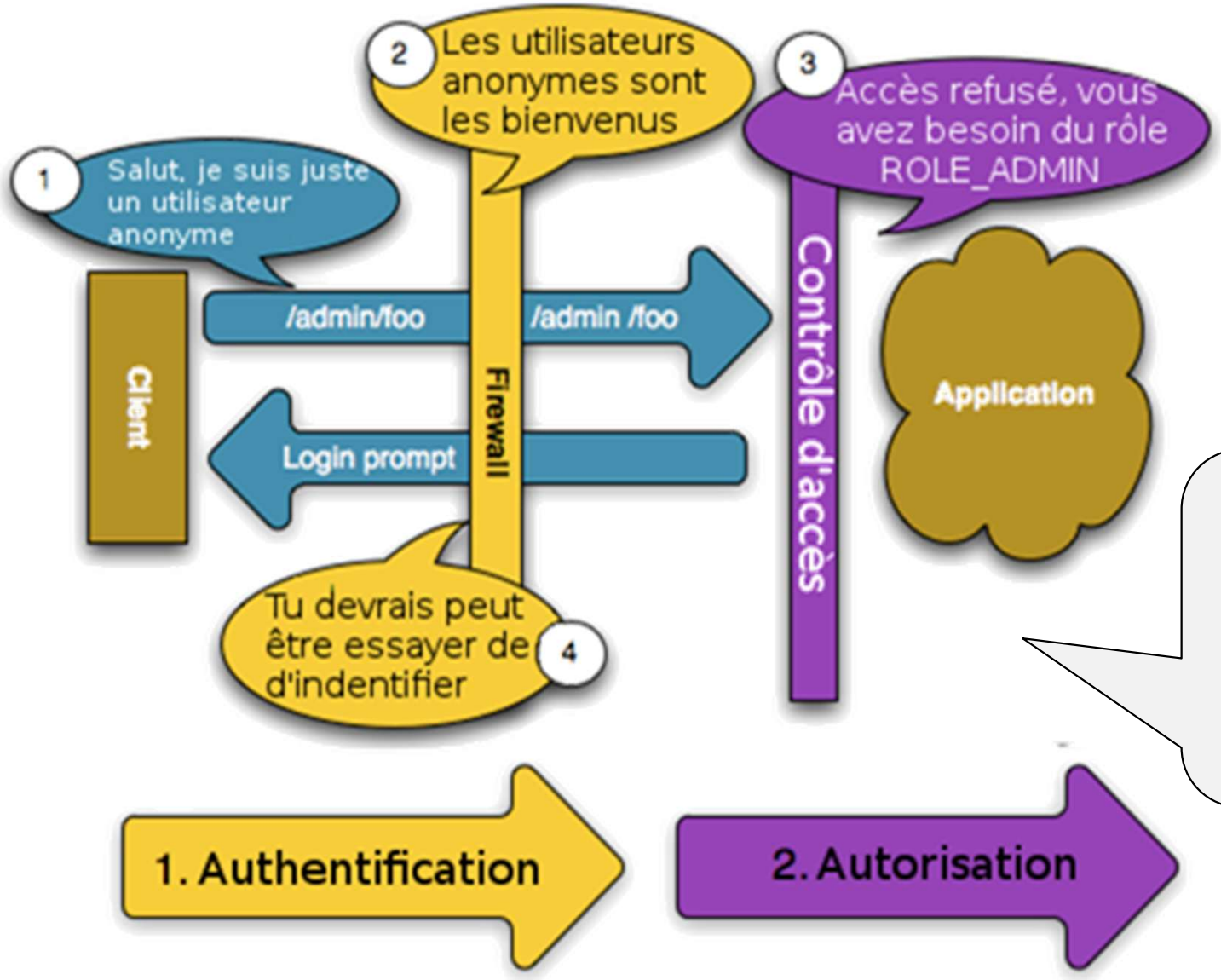
Ici le pare-feu est activé pour l'URL **/foo** mais aucune identification ne sera demandée pour cette URL car elle ne nécessite aucun rôle particulier et le pare-feu est configuré pour autoriser les utilisateurs anonymes



Contrôle d'accès : Autorisation (1)

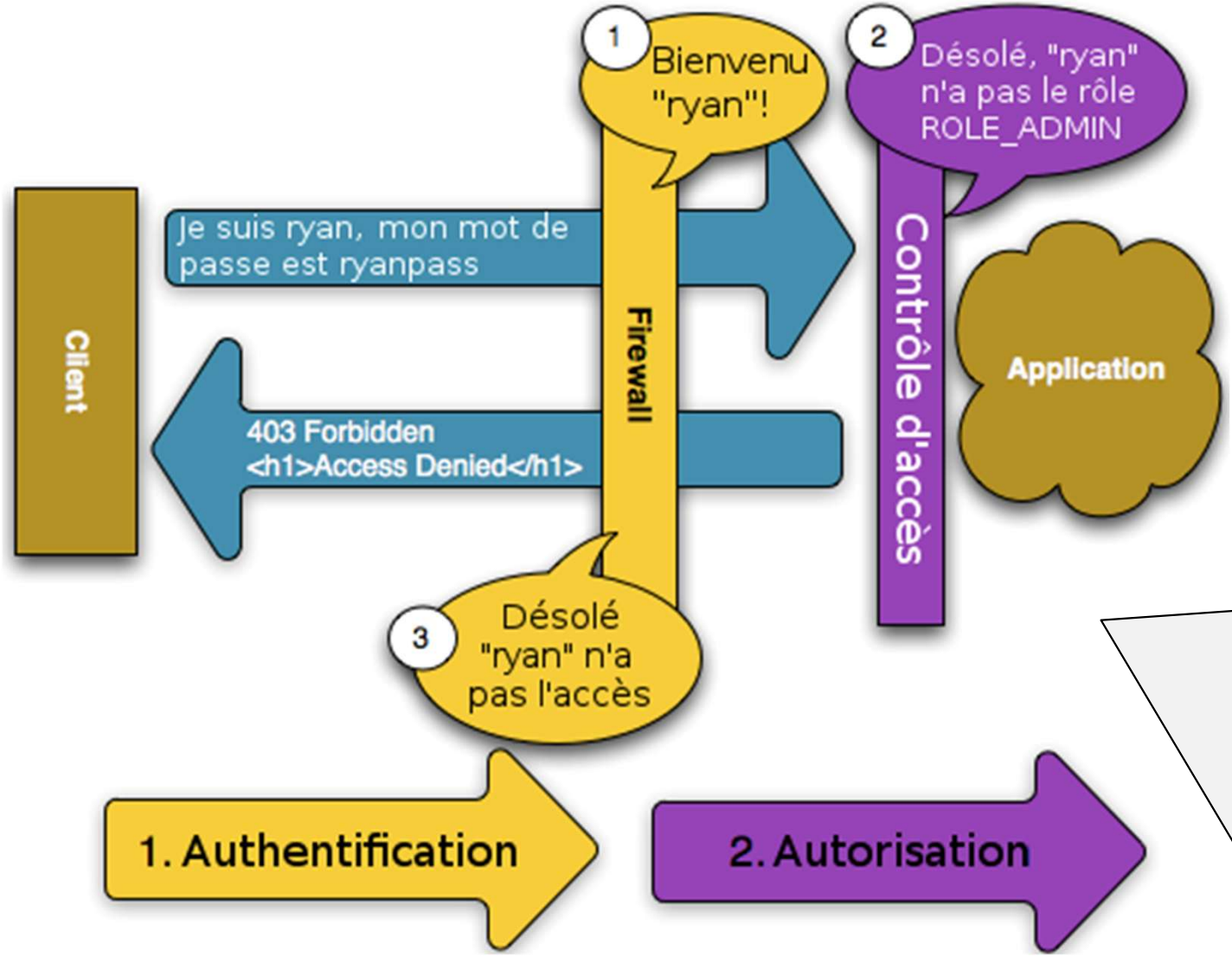
- Dès que la couche de contrôle d'accès refuse l'accès à l'utilisateur, le pare-feu initialise le processus d'authentification.
- Le processus d'authentification dépend du mécanisme choisi :
  - Si authentification *via* un formulaire de connexion, l'utilisateur est redirigé vers la page de formulaire de connexion.
  - Si authentification HTTP, l'utilisateur reçoit une réponse HTTP 401 et verra donc la page login/mot de passe de son navigateur
  - …

Contrôle d'accès : Autorisation (2)



Ici l'URL **/admin/foo** nécessite le rôle **ROLE_ADMIN** et l'utilisateur se voit donc demandé de s'authentifier

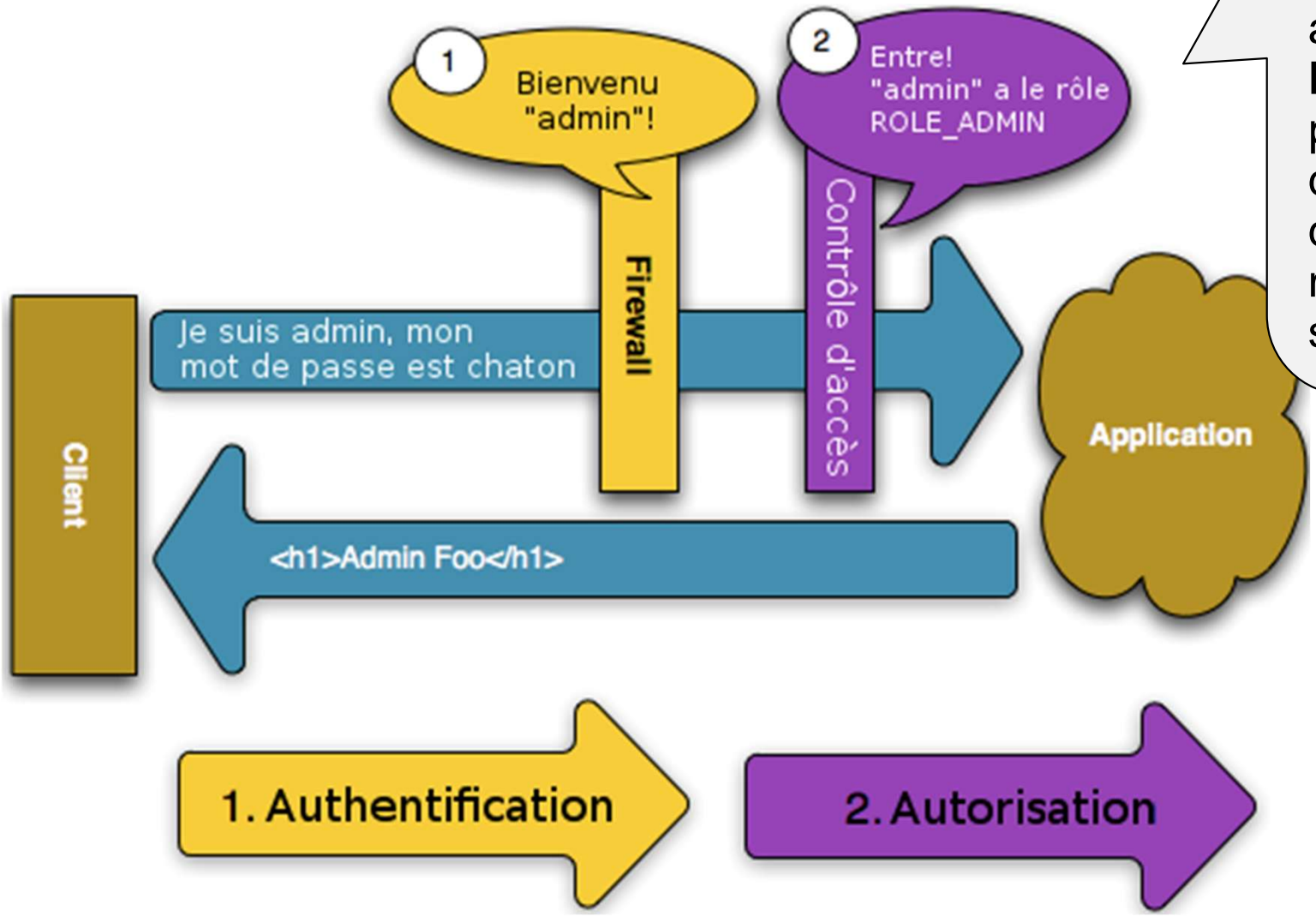
Contrôle d'accès : Autorisation (3)



Une fois l'authentification réalisée avec succès, la requête initiale est relancée

Si en s'authentifiant l'utilisateur n'a pas acquis des privilèges (un **rôle**) lui permettant d'accéder à l'URL demandée, il recevra une page d'erreur et un code d'erreur **HTTP 403** (Forbidden)

Contrôle d'accès : Autorisation (4)



Si par contre, en s'authentifiant, l'utilisateur a bien acquis le rôle **ROLE_ADMIN** lui permettant d'accéder à l'URL demandée, sa requête initiale sera satisfaite

Configuration (1)

- Lorsque l'on développe avec Symfony, la sécurité de l'application n'est pas laissée à la charge du développeur. Il doit simplement la **configurer**
- La configuration de la sécurité de l'application se fait dans le fichier **security.yml** du répertoire **config/packages** du projet.
- Dans ce fichier on trouve les sections suivantes :
  - **providers** : définition du ou des « fournisseurs d'utilisateurs ». C'est ici que l'on précise la source de données pour identifier les utilisateurs. Ils peuvent être stockés « in-memory » dans le fichier **security.yml**, provenir d'un repository d'entités (Usager), …
  - **role_hierarchy** : une définition hiérarchique des différents rôles reconnus par l'application (par exemple le rôle ROLE_ADMIN qui hérite du rôle ROLE_CLIENT)
  - **password_hashers :** choix du type d'encodage des mots de passe des utilisateurs
  - **firewalls** : la liste des « firewall » définis pour l'application et pour chacun d'eux :
    - Le masque des URL qu'il surveille
    - S'il autorise au pas les accès anonymes
    - La méthode d'authentification qu'il utilise
  - **access_control** : une liste d'URLs ou de « templates » d'URLs (définis par des expressions régulières) avec, pour chacun d'eux, les rôles qu'il faut détenir pour pouvoir y accéder

Configuration (2)

- Dans le cas de notre application, nous allons nous restreindre à l'étude de la configuration suivante :
  - Les utilisateurs proviennent d'un dépôt d'entités de type **`Usager`**, créé au TP05 par :

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

  - Il n'y aura que deux rôles :
    - **`ROLE_CLIENT`** : le rôle que devra avoir un utilisateur authentifié qui veut passer une commande, consulter ses commandes antérieures, …
    - **`ROLE_ADMIN`** : le rôle que devrait avoir un utilisateur authentifié pour accéder au Back-Office d'administration
    - Le **`ROLE_ADMIN`** contiendra le **`ROLE_CLIENT`**
  - L'authentification se fera *via* un formulaire traditionnel

1. Mettre en place l'Authentification (1)

- Pour gérer la sécurité, il faut demander à Symfony :
  - De créer un **service d'authentification** :
    Un service (**`FormLoginAuthenticator`**) est créé par défaut par la commande que nous allons utiliser. Il n'y a pas à le modifier. Il est possible (autre commande) de créer un service « personnalisable »
  - De créer un **contrôleur** :
    `src/Controller/SecurityController.php`
    Ce contrôleur va gérer les routes de connexion (login) et de déconnexion (logout)
  - De créer un **template** d'authentification avec deux champs login et mot de passe : **`templates/security/login.html.twig`**
  - D'adapter en conséquence le **fichier de configuration** de la sécurité : **`config/packages/security.yml`**
- Ces éléments sont créés grâce à la commande :

```
php bin/console make:security:form-login
```

1. Mettre en place l'Authentification (2)

```
php bin/console make:security:form-login

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
 >

 Do you want to generate a '/logout' URL? (yes/no) [yes]:
 >

 Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]:
 >

 created: src/Controller/SecurityController.php
 created: templates/security/login.html.twig
 updated: config/packages/security.yaml


  Success!


 Next: Review and adapt the login template: security/login.html.twig to suit your needs.
```

Attention : parfois la commande demande le nom de l'entité utilisée pour l'authentification. Indiquer : **`App\Entity\Usager`**

Il va falloir maintenant « adapter » ces fichiers !

2. Adapter SecurityController

- Si votre application gère l'internationalisation (paramètre `_locale` dans les URL), il faut penser à modifier les deux routes **`app_login`** et **`app_logout`** qui sont créées lors de la mise en place de la sécurité, dans **`SecurityController.php`** :

```php
// src/Controller.SecurityController
class SecurityController extends AbstractController {
    #[Route(
        path: '/{_locale}/login',
        name: 'app_login',
        requirements: ['_locale' => '%app.supported_locales%']
    )]
    public function login(AuthenticationUtils $authenticationUtils): Response {
        // …
    }

    #[Route(
        path: '/{_locale}/logout',
        name: 'app_logout',
        requirements: ['_locale' => '%app.supported_locales%']
    )]
    public function logout(): void {
        // …
    }
}
```

3. Adapter login.html.twig

- Il faut également modifier le *template* du formulaire d'authentification pour l'intégrer correctement à l'interface de votre site (par exemple mettre en place l'i18n).
- Les champs login et password doivent impérativement être nommés **_username** et **_password**

```
{# templates/security/login.html.twig #}
{% extends 'base.html.twig' %}
{% block title %}Connexion{% endblock %}
{% block content %}
<form method="post">
  {% if error %}
    <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
  {% endif %}
  {% if app.user %}
    <div class="mb-3">
    Vous êtes connecté en tant que {{ app.user.username }}, <a href="{{ path('app_logout') }}">Logout</a>
    </div>
  {% endif %}
  <div class="col-md-6 offset-3">
    <h1 class="h3 mb-3 font-weight-normal">Sinon : Authentification</h1>
    <label for="inputEmail">Email</label>
    <input type="email" value="{{ last_username }}" name="_username" id="username" class="form-control"
           required autofocus>
    </br>
    <label for="inputPassword">Mot de Passe</label>
    <input type="password" name="_password" id="password" class="form-control" required>
    <input type="hidden" name="_csrf_token"
           value="{{ csrf_token('authenticate') }}">
    </br>
    <button class="btn btn-lg btn-primary" type="submit">
        Connexion
    </button>
  </div>
</form>
{% endblock %}
```

Un token est inséré dans le formulaire d'authentification pour renforcer la sécurité

4. Configurer security.yml (1)

```yaml
# config/packages/security.yml
security:
    providers:
        app_user_provider:
            entity:
                class: App\Entity\Usager
                property: email
    role_hierarchy:
        ROLE_ADMIN: ROLE_CLIENT
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
        App\Entity\Usager:
            algorithm: auto
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: app_user_provider
            form_login:
                login_path: app_login
                check_path: app_login
                enable_csrf: true
            logout:
                path: app_logout
                target: home
    access_control:
        - { path: ^/(%app.supported_locales%)/admin, roles: ROLE_ADMIN }
        - { path: ^/(%app.supported_locales%)/usager/, roles: ROLE_CLIENT }
```

Les utilisateurs seront recherchés parmi les entités **Usager** ,
le login est la propriété **email** de cette entité

Le rôle **ROLE_ADMIN** inclut le rôle **ROLE_CLIENT**

à définir

Par défaut c'est **bcrypt** qui est utilisé… Passer à **sodium** (argon2)

Le firewall est activé pour toutes les URL,
l'accès anonyme est autorisé,
L'authentification se fera par le service **LoginFormAuthenticator**
La route de déconnexion est **app_logout** et elle redirige ensuite vers la route **home**

Il faut le rôle **ROLE_ADMIN** pour accéder au back_office du site
Il faut le rôle **ROLE_CLIENT** pour accéder à la partie « usager connecté » du site

C'est un exemple à adapter !

4. Configurer security.yml (2)

- Dans la section **`access_control`**, l'ordre dans lesquels les règles sont déclarées est important : c'est la première règle dont l'expression régulière correspond l'URL courante qui sera appliquée.
- Par exemple : si l'on veut que ROLE_CLIENT soit nécessaire pour **toutes** les routes qui commencent par le préfixe /usager, **sauf** l'URL /usager/new qui doit être accessible à tous, alors il faut définir les règles dans cet ordre :

```
- { path: /usager/new, roles: PUBLIC_ACCESS }
- { path: ^/usager, roles: ROLE_CLIENT }
```

- Si l'on écrit les règles dans l'ordre inverse cela ne fonctionnera pas !
- Il est aussi possible de définir les règles d'accès au niveau des contrôleurs, par des attributs :

```php
use Symfony\Component\Security\Http\Attribute\IsGranted;
#[IsGranted('ROLE_ADMIN')]
class AdminController extends AbstractController
{
    // Il est possible de définir un message spécifique
    #[IsGranted('ROLE_SUPER_ADMIN', message: Accès reserve Admin.')]
    public function adminDashboard(): Response
    {
        // ...
    }
}
```

Une fois la sécurité mise en place...

- Dans un **contrôleur**, on peut récupérer l'objet utilisateur, savoir s'il est authentifié
- On peut tester le rôle de l'utilisateur

```
// Dans un contrôleur quelconque...
public function administrator(): Response {
     // on peut récupérer l'objet utilisateur
    $user = $this->getUser(); // $user sera une entité de type Usager (ou null)
    ...

    // on peut tester par exemple que l'utilisateur a bien le rôle d'administrateur
    if (! $this->isGranted('ROLE_ADMIN')) {
        ...
    }
}
```

- Dans un **service**, il faut injecter **Symfony\Component\Security\Core\Security** et on peut accéder à l'usager authentifié par : **`$security->getUser()`**
- Dans un *template* twig on peut également accéder à l'objet utilisateur (**app.user**) et tester ses rôles :

```
{% if app.user %}
  <p>Nom de l'usager : {{ app.user.nom }}</p>
{% endif %}
{% if is_granted('ROLE_ADMIN') %}
    <a href="...">Supprimer</a>
{% endif %}
```

TP 06 : travail à réaliser

- Mettez en place la sécurité comme elle vient de vous être présentée
- Pensez à bien définir les pages « protégées » dans la section **`access_control`** du fichier « security.yaml »…
  Toute la sécurité de votre application en dépend !
- Modifiez la validation du panier pour que l'usager qui passe la commande soit maintenant **l'usager authentifié**
- Modifiez éventuellement votre barre de navigation pour qu'elle s'adapte s'il y a ou non un usager authentifié
- Mettez en place une page qui permet à l'usager authentifié de consulter la liste de ses commandes