Architecture(s) et application(s) Web

CSC4101 - Sessions, Contrôle d'accès

24/08/2023

Plan de la séquence

1. Sessions applicatives

Paradoxe : applications sur protocole sans état

- L'expérience utilisateur suppose une **instance unique** d'exécution d'application, comme dans un programme « sur le bureau »
- Faciliter la contextualisation du HATEOS : quelles transitions sont valides à partir de l'état courant ?
- Pourtant, HTTP est *stateless* (sans état) : chaque requête recrée un nouveau contexte d'exécution

L’application n’arrête pas de s’arrêter

À chaque requête :

1. démarrage application
2. routage vers méthode Contrôleur
    1. chargement depuis base **et/ou session**
    2. traitements
    3. sauvegarde en base **et/ou session**
3. envoi Réponse
4. **mort**

Sans la session, continuité entre deux requêtes ?

Application peut garder la mémoire

- Le programme Web peut stocker un état (par ex. en base de données) à la fin de chaque réponse à une requête
- Il peut le retrouver au début de la requête suivante
    - **Le client doit pour cela se faire reconnaître**
- Simule une session d'exécution unique comprenant une séquence d'actions de l'utilisateur

Le client HTTP peut s'identifier

- Argument d'invocation dans URL
- en-tête particulier ?
- ***Cookie***
- …

Identification du client par *cookie*

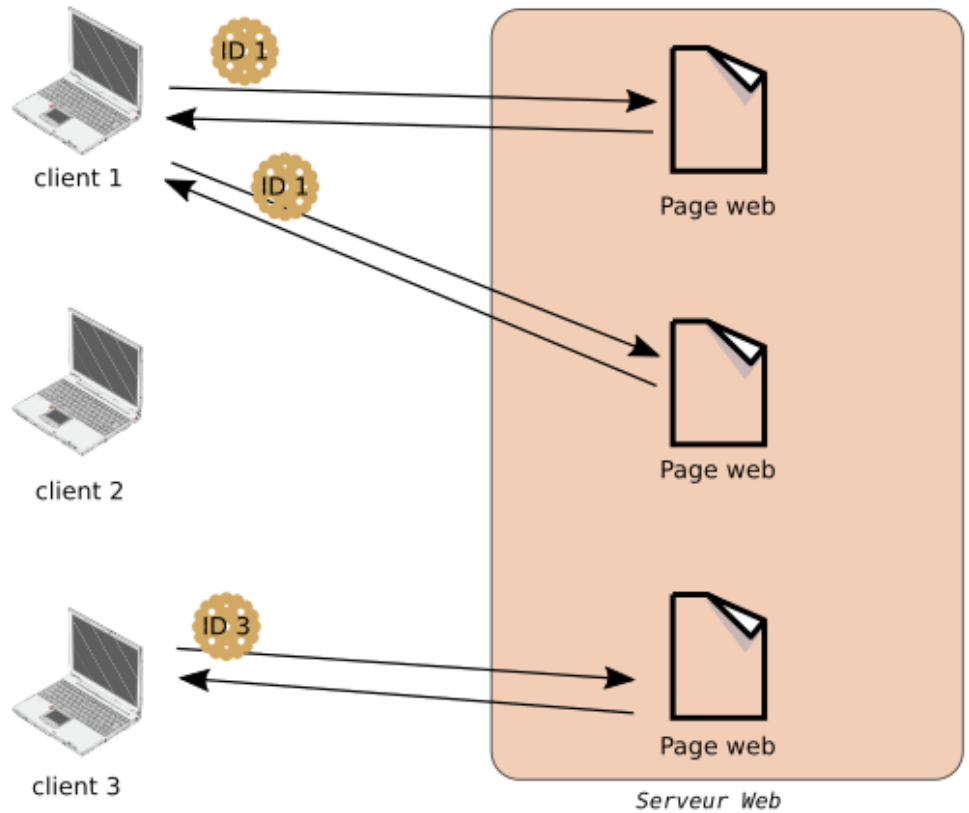

Figure 1 : Multiples clients et pages

- Identifie le client
- Commun à un ensemble d’URLs

Cookies

- Juste un « jeton » unique sur un certain périmètre (serveur, chemin d'URL, application, …)
- Format choisi par le serveur
- Peut contenir complètement un état de l'application : des données, mais **taille limitée**
- Le serveur peut trouver des données plus complètes stockées, sur présentation du jeton (une **session**).

Reconnaître le client HTTP

- À chaque requête, le *cookie* permet relier :
  - nouvelle requête d'un client HTTP (en-têtes HTTP : **valeur d'un *cookie* existant**)
  - **mémoire de l'état précédent** sauvegardé côté serveur à la **fin de la réponse HTTP précédente** du même client
- Taille limitée
- Données en clair dans le *stockage de cookies du navigateur*
- Durée de vie potentiellement grande

Stocker une session

- Session : espace de stockage unique **côté serveur**
- **Objets du modèle** de l'application stockés dans la session (pas contrainte taille), *sérialisés*
- Session retrouvée via un **identifiant**
    - Identifiant fourni par un *cookie*
- Durée de vie et unicité des sessions au choix du serveur

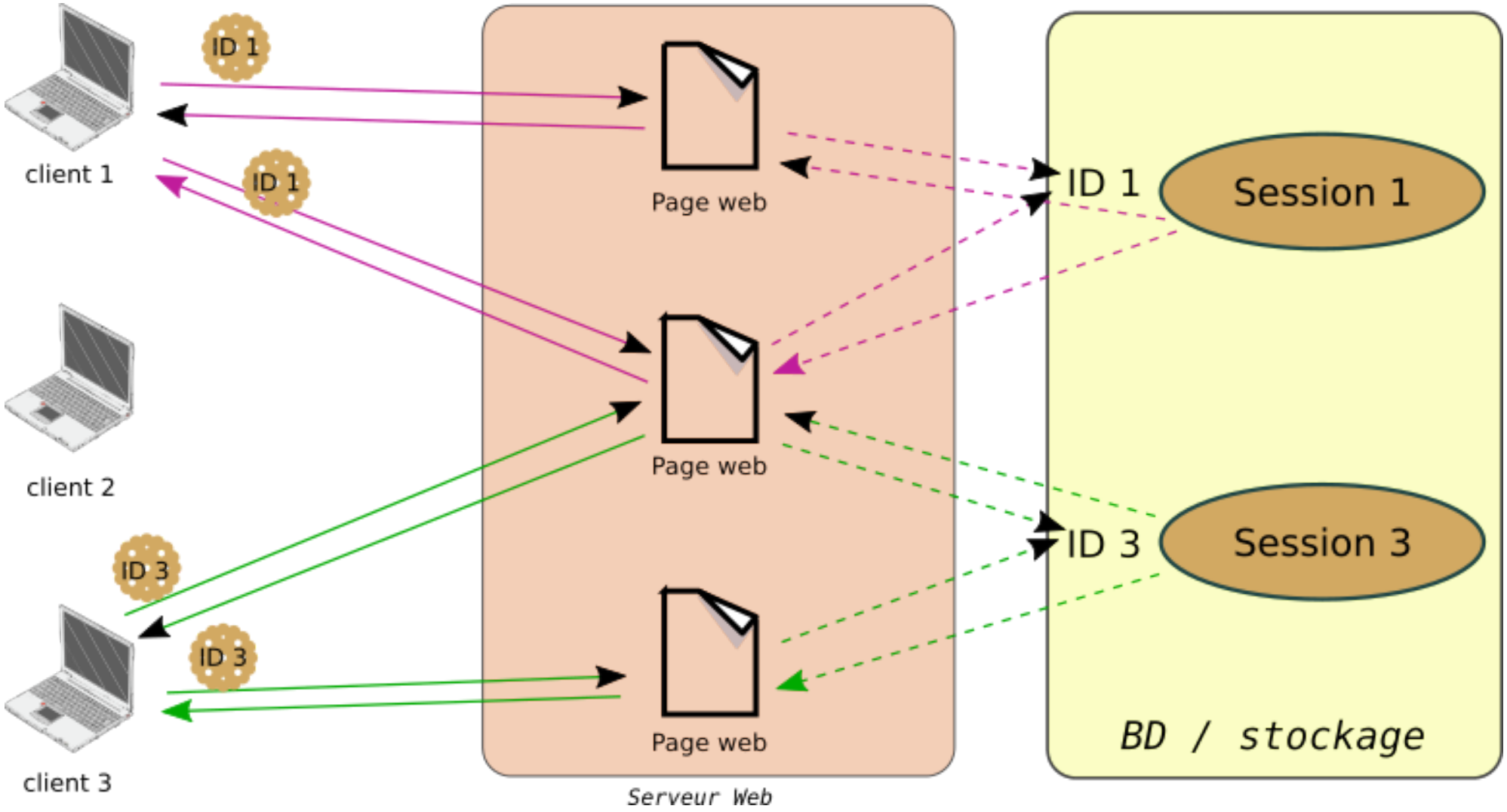

Figure 2 : Stockage session côté serveur

Détails *cookie*

- Lors de la première requête d'un client (n'ayant pas fourni de cookie)
- Le serveur crée les *cookies* et les **intégre dans en-têtes de réponse** HTTP
- Utilise l'en-tête particulier « `Set-Cookie` » (sur une ligne)

```
Set-Cookie: <nom>=<valeur>;expires=<Date>;domain=<NomDeDomaine>; path=<
```

- Exemple de réponse HTTP :

```
HTTP/1.1 200 OK
Server: Netscape-Entreprise/2.01
Content-Type: text/html
Content-Length: 100
Set-Cookie: clientID=6969;domain=unsite.com; path=/jeux
```

- Par la suite, ce cookie sera renvoyé par le client au serveur, dans chaque requête ayant comme URL de début :

http://www.unsite.com/jeux/.....

Wrap-up sessions

- Requêtes HTTP déclenchées lors demandes de **transition d'un état à l'autre** de l'application
- L'exécution (PHP) résultante s'effectue sur serveur HTTP **sans mémoire** des interactions précédentes entre client et serveur (*stateless*)
- L'utilisateur a une **impression de continuité** : une seule session d'utilisation de l'application, où requêtes successives ont des **relations de causalité**
- Différentes solutions techniques, dont les *cookies*

Plan de la séquence

2. Contrôle des accès

Protéger les données / fonctions

- Confidentialité : application accessible sur Internet, même si processus / données privés
- Privilèges : qui fait quoi
    - Spécifications fonctionnelles (profils utilisateurs)
    - Contrôle par l’application (HATEOS)
- Contrôle d’accès : reconnaître les utilisateurs, et mettre ne place les restrictions (sans nuire à l’utilisabilité, mobilité, etc.)

Autres aspects sécurité vus dans une séance ultérieure

Contrôle des accès

- Protéger l’accès aux fonctionnalités de l’application
- Qui est autorisé à faire quoi

Dans un monde ouvert (Internet, Web, standards)

Sécurité par obscurcissement ?

- Ne pas protéger spécifiquement,
- et ne pas documenter / expliquer / rendre visible ?

Ce n’est pas parce que le code de l’application est caché sur le serveur que les méchants ne trouveront pas des failles !

#Fail

Contrôle effectif

- Au niveau de la configuration du serveur (ne pas permettre aux clients de découvrir les failles en regardant le source)
- **Dans les fonctionnalités du logiciel** : « firewall » Symfony
- Mesures complémentaires (audit, etc.)

Modèle contrôle des accès

Identification

l'utilisateur fournit à un service un moyen de le reconnaître : **identité**

Authentification

le service **vérifie** cette identité

Autorisation

le service donne à l'utilisateur certaines **permissions**

Identification

- Identifiants :
  - email
  - identifiant d'utilisateur (*login*)
  - certificat client X509 (TLS)
  - Jeton dans un en-tête HTTP
  - …
- L'identification doit être confirmée par l'authentification

Authentification

- Vérifie client agit bien pour le **propriétaire légitime de l'identifiant** présenté
- Protocole de vérification :
  - mot-de-passe
  - signature valide d'une donnée quelconque (*challenge*) avec la clé privée associée au certificat client
  - délégation à service externe (Shibboleth/CAS, OAuth 2, ...)
  - nouveaux standards : U2F, Fido
  - ...

Autorisations

- Permissions applicatives associées à l'identifiant
- Pour une certaine période
- Génération d'un jeton temporaire (session authentifiée)
    - *Cookie*

Dans protocole HTTP

- Identification / authentification de « bas niveau » **dans le protocole** HTTP (cf. ~~RFC2617~~ RFC 7617: The 'Basic' HTTP Authentication Scheme)
- Rappel : HTTP est sans état
  - Le client HTTP doit se réauthentifier à chaque requête
- Permet de transporter l'authentification dans les en-têtes
- Alternative : **authentification applicative** + session applicative

</controle_acces>

Plan de la séquence

3. Mécanismes d'authentification

Mécanismes

- Authentification HTTP
  - Authentification « Basic »
  - autres
- **Authentification applicative**

Basic Auth

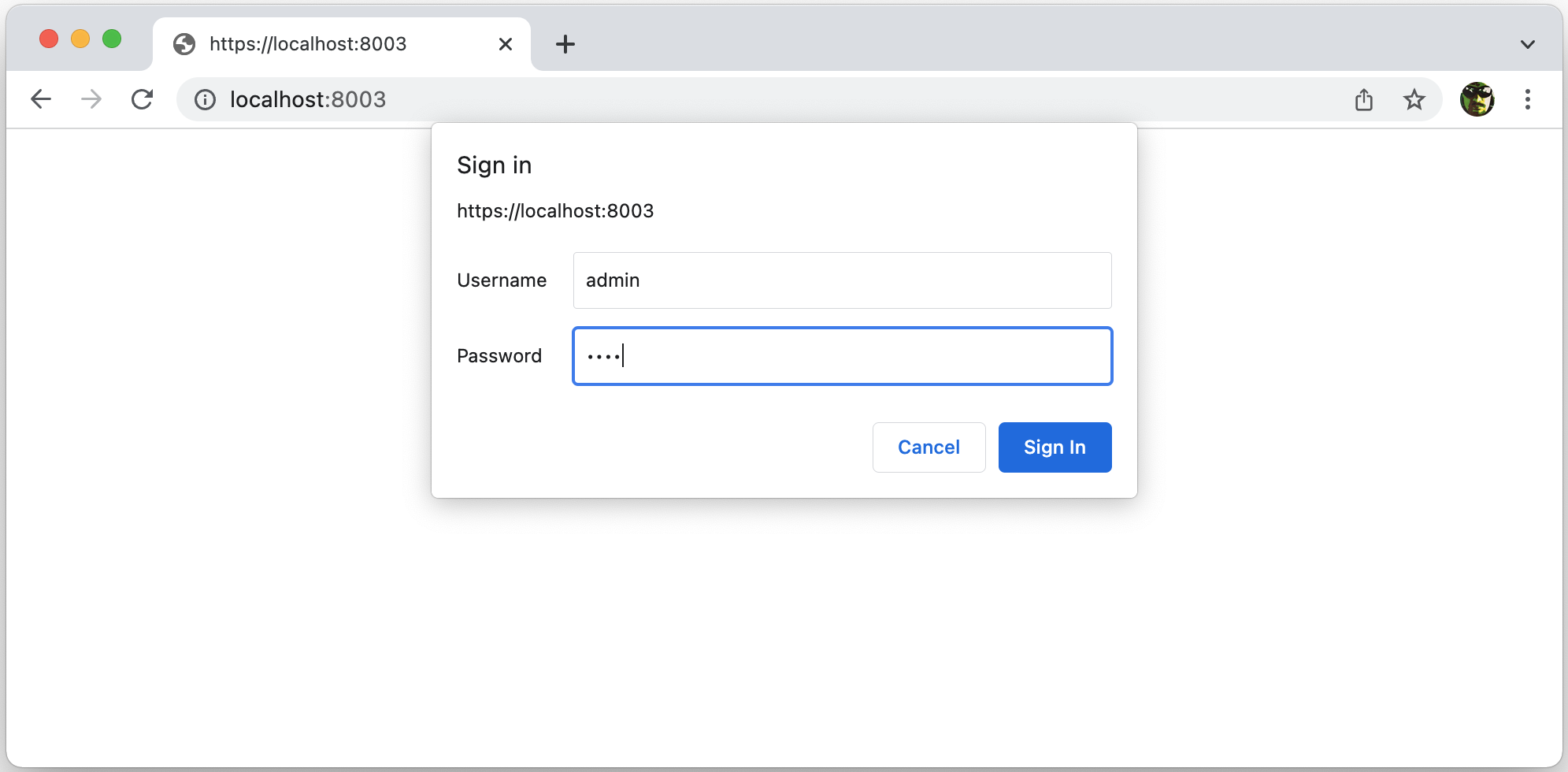

Figure 3 : Source : Why I'm Using HTTP Basic Auth in 2022 par Joel Dare

Inconvénient : pas de *logout*

Authentification applicative

Gestion de l’identification et de l’authentification par l’application

- L’authentification est une des fonctionnalités de l’application, via la session
- **Formulaire** d’authentification
  - *Login*
  - Mot-de-passe
- « base de données » de comptes

Formulaire d’authentification

- Formulaire « standard »
- Champs :
  - Login ou email
  - Mot-de-passe (saisie cachée)

Vérification de l’authentification

- Comparer avec profil d’utilisateur connu (en base de données)
- Générer une **session** pour reconnaître l’utilisateur par la suite
- Attention : attaques « force brute »
  - Invalider un compte/profil, ou faire une gestion de surcharge qui désactive les tentatives (*throttling*, *blacklist* réseau, etc.)

Dans Symfony

- Composant `Security`
- Flexible : gestion souple et extensible de l'authentification
- Gère par exemple les utilisateurs dans la base de données via classe `User` + Doctrine
- Assistants générateurs de code pour les dialogues

Procédures ?

- Gestion des mots-de-passe (qualité aléa, longueur, stockage approprié, etc.)
- Récupération de compte si oubli mot-de-passe
  - Canal sécurisé ou envoi jeton de réinitialisation sur email (implique gestion emails)
- Confirmations d’authentification pour sections critiques de l’application
- Garder des traces (audit, obligations légales)
- Conformité RGPD (données personnelles dans les profils)

Captcha

Completely Automated Public Turing test to tell Computers and Humans Apart

Vérifier qu'un humain est aux commandes

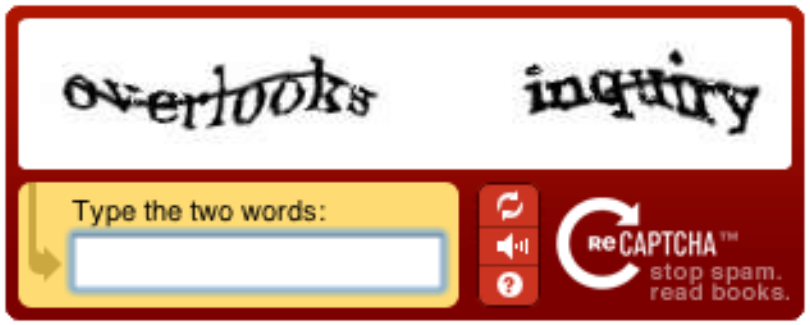

Figure 4 : Exemple reCAPTCHA

- Pas infaillible
- Problèmes accessibilité

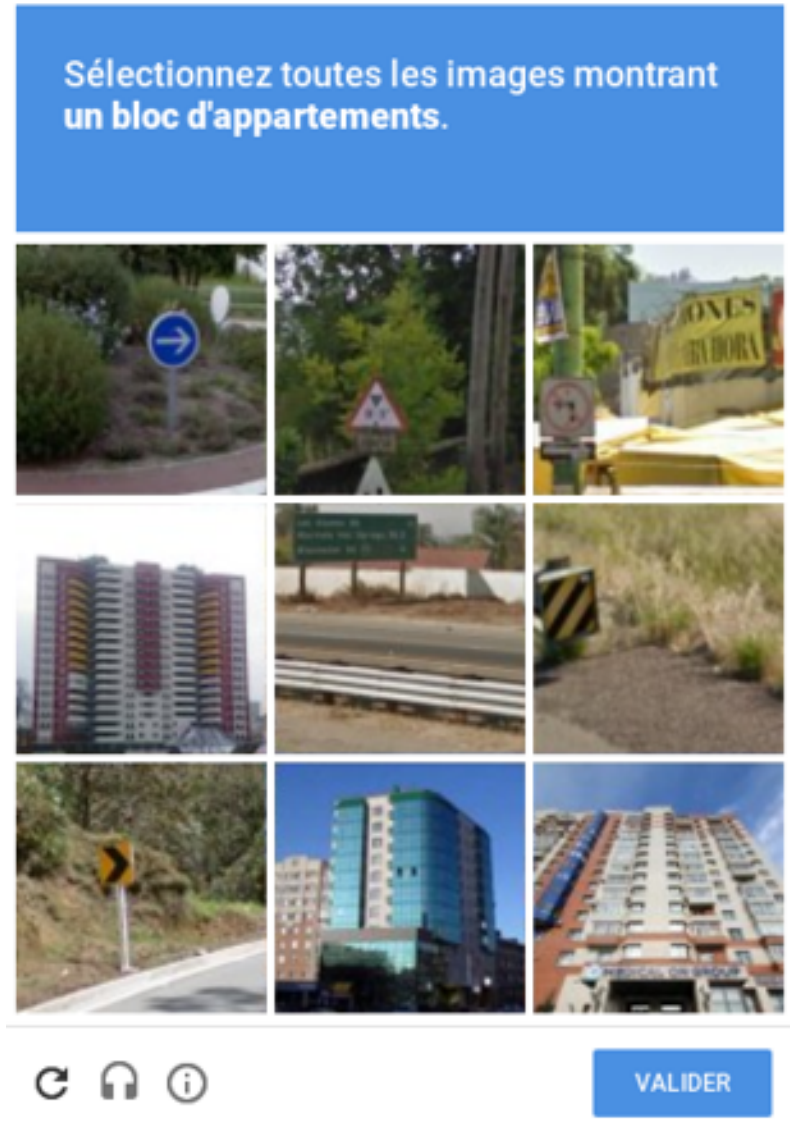

Figure 5 : CAPTCHA et digital labor

- Travail dissimulé
- Surveillance

Authentification à double facteur

2FA (*Two factor authentication*)

- + robuste :
    1. élémént connu
    2. élément possédé
- Exemples :
    - carte bancaire (possession) + code PIN (connu)
    - login + mdp (connu) + SMS reçu (possession mobile)
    - login + mdp (connu) + badge de sécurité générant un code unique (possession)

</authentification>

Plan de la séquence

4. Rôles et permissions

Role-Based Access Control (RBAC)

Contrôle d’accès à base de rôles

- Utilisateur
- **Rôle**
- Permissions

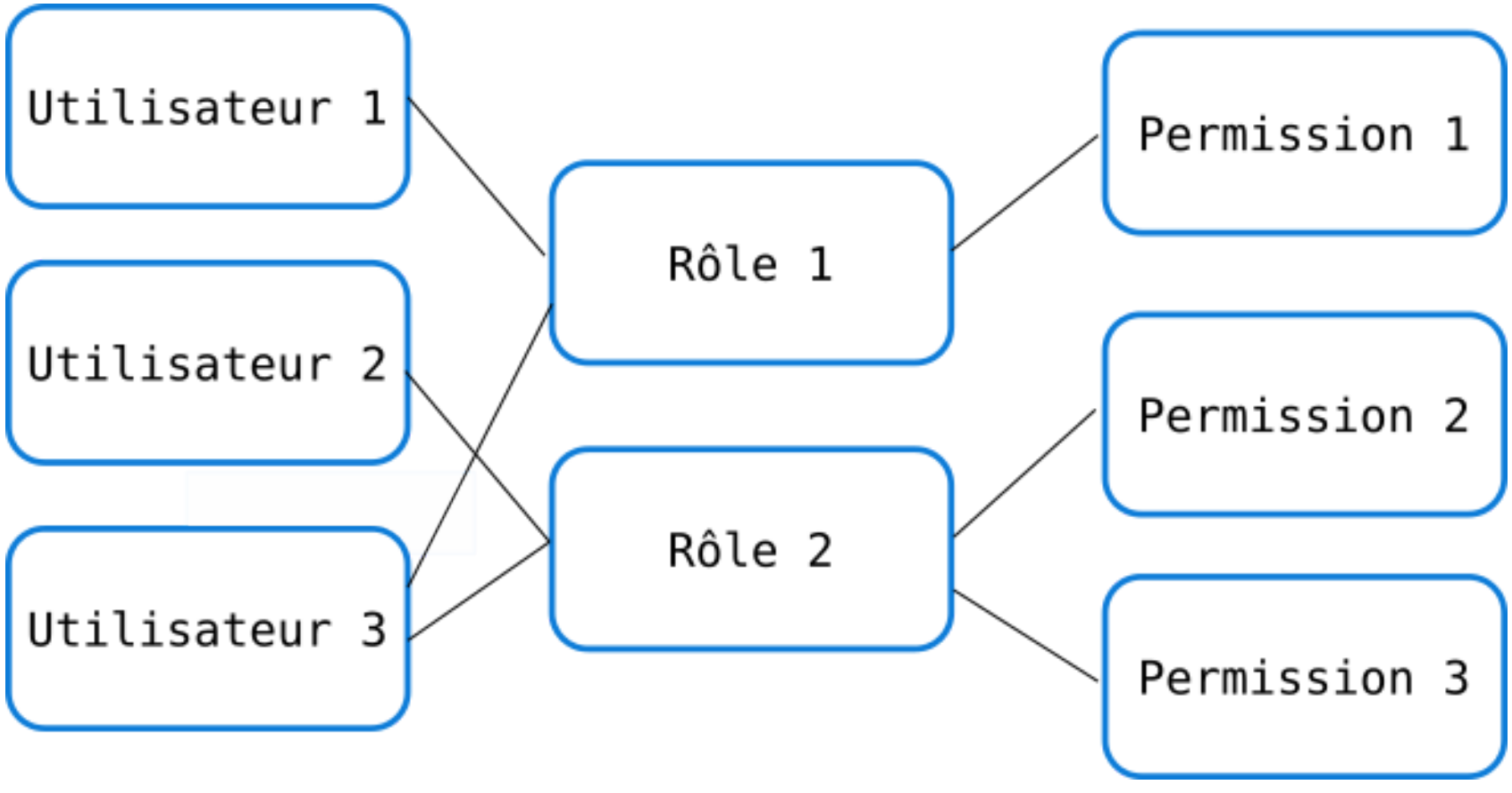

Figure 6 : Exemple d’affectation de rôles

Permissions

- Modèle applicatif de permissions
- Vérifier les permissions à chaque traitement de requête
    - Routage
    - Dans les traitements fonctionnels

Module « *Firewall* » de Symfony

Réponses Web

- Code 200 + Page mentionnant problème de permissions
- **Code 403** (et peut-être un message dans la page) ?

</RBAC>

Plan de la séquence

5. Mise en œuvre avec Symfony

Flexibilité

- Symfony permet de gérer plein de modalités de l’authentification
- Choix : s’appuyer sur la base de données, et des contrôleurs d’authentification générés par les assistants

Gestion des utilisateurs avec Doctrine

- Classe `User` du Modèle (et *mapping* Doctrine en base)
- Définition de règles dans le *firewall* Symfony
- Rôles
- Ajouter des formulaires (+ *templates*) :
    - Login + password
    - Logout
    - (Inscription, rappel du mot-de-passe, ...)

Classe User

```
symfony console make:user
```

```php
namespace App\Entity;

use App\Repository\UserRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterfa
use Symfony\Component\Security\Core\User\UserInterface;

#[ORM\Entity(repositoryClass: UserRepository::class)]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    // ...

    #[ORM\Column(length: 180, unique: true)]
    private ?string $email = null;
```

Hiérarchie de rôles

- Arbitraire, selon les besoins de l'application
- Exemple :
    1. ROLE_SUPER_ADMIN
    2. ROLE_ADMIN
    3. ROLE_CLIENT
    4. ROLE_USER

```
# security.yml

role_hierarchy:
    ROLE_CLIENT:       ROLE_USER
    ROLE_ADMIN:        ROLE_USER
    ROLE_SUPER_ADMIN: [ROLE_USER, ROLE_ADMIN]
```

Firewall

Contrôle l'accès aux URLs en fonction des rôles :

```
# app/config/security.yml
security:
    # ...

    firewalls:
        # ...
        default:
            # ...

    access_control:
        # require ROLE_ADMIN for /admin*
        - { path: ^/admin, roles: ROLE_ADMIN }
```

Utilisation dans les contrôleurs

- Contrôle d'accès sur les routes :

```
#[Route('/comment/{postId}/new', name: 'comment_new', methods: ['GET
#[IsGranted('IS_AUTHENTICATED_FULLY')]
```

Encore une fois, importance des attributs

- Contrôle d'autorisation dans le code des méthodes

```
$this->denyAccessUnlessGranted('ROLE_ADMIN', null, 'Access denied!')
```

« Entrée interdite, à moins que… »

Profil de l’utilisateur

- Accès aux propriétés de l’utilisateur :

```
$this->getUser()

// ...

$email = $this->getUser()->getEmail();
$post->setAuthorEmail($email);
```

Personnalisation apparence

Gabarits Twig

```
{% if is_granted('ROLE_ADMIN') %}
  <a href="...">Delete</a>
{% endif %}
```

Gestion fine

- Dans code d'une méthode de contrôleur :

```php
$this->denyAccessUnlessGranted('ROLE_ADMIN', null, 'Access denied!');
```

- équivalent à :

```php
if (! $this->get('security.authorization_checker')->isGranted('ROLE_
    throw $this->createAccessDeniedException('Access denied!');
}
```

Déclenche une **exception** :

- erreur 403
- ou redirection vers login

Exceptions et codes retour

```php
try {
    // faire quelque chose qui appelle : throw
} catch (Exception $e) {
    echo 'Exception reçue : ',  $e->getMessage(), "\n";
}
```

Permet d'intercepter de façon standard les exceptions :

- `AccessDeniedException` (403)
- `NotFoundHttpException` (404)

</symfony>

Take away

- Sessions
    - Cookies
    - Session Symfony
- Contrôle d’accès
    - Principes
        - Identification
        - Authentification
        - Autorisations
    - Rôles (RBAC)
    - Contrôle dans Symfony

Récap

- [X] HTTP (GET)
- [X] PHP
- [X] Doctrine
- [X] Routeur Symfony
- [X] HTML
- [X] Twig
- [X] CSS
- [X] Formulaires
- [X] **Sessions**
- [X] **Contrôle d’accès**

Postface

Crédits illustrations et vidéos

- Illustration copie écran Basic Auth HTTP via Joel Dare

Copyright

- Document propriété de ses auteurs et de Télécom SudParis (sauf exceptions explicitement mentionnées).
- Réservé à l'utilisation pour la formation initiale à Télécom SudParis.