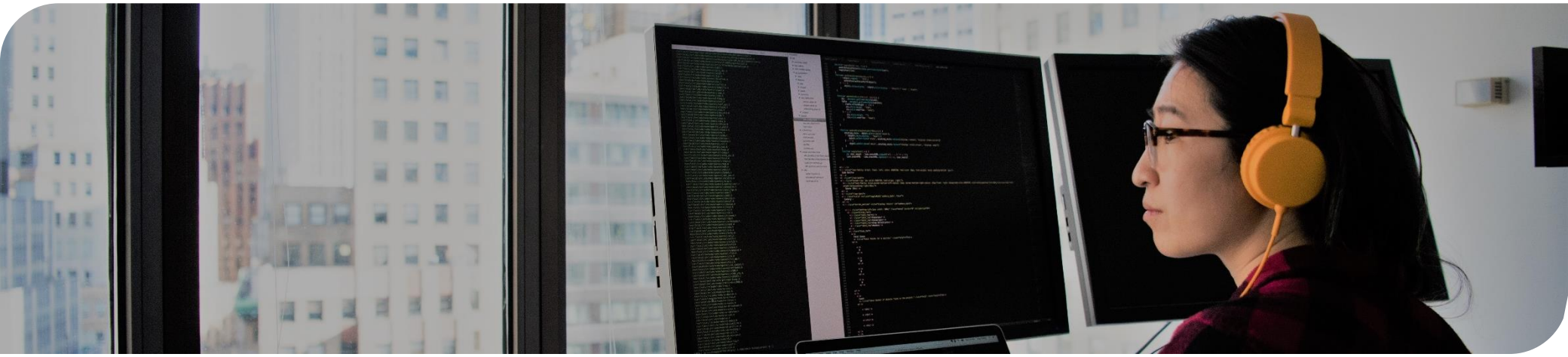
JUNIORISEP



Formation Symfony 5.0

Cours 4

Lundi 13 Janvier | Pierre MARQUET & Laurianne PROGENT & Florent FAVOLE

Symfony

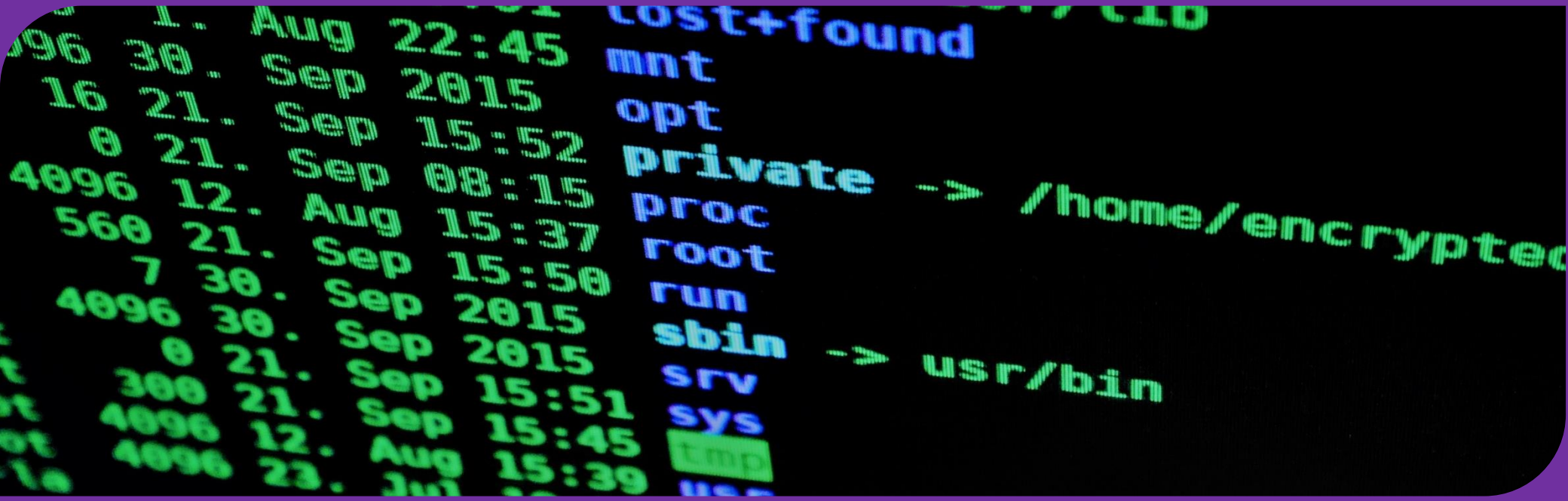
Récapitulatif cours 3

Vu la semaine dernière :

- **Les Formulaires :** builder, types de champs, Request, etc.
- **Les contraintes de validation :** en longueur, NotNull, etc.
- **Les images**

Des questions ?

Dernière ligne droite ! ☺



1 Introduction à la sécurité

La Sécurité

Si ce n'est pas déjà fait, require le bundle security :

$ composer require security

La sécurité est un ensemble de **règles** qui attribuent plus ou moins de **liberté** aux utilisateurs de votre application

→ La sécurité passe par un système d'**authentification** des utilisateurs de votre application et vous permet de leur donner des autorisations (droits)

Les configurations sur la sécurité du projet sont définies de manière générale dans le fichier **security.yaml** situé dans le dossier /config/packages

La Sécurité

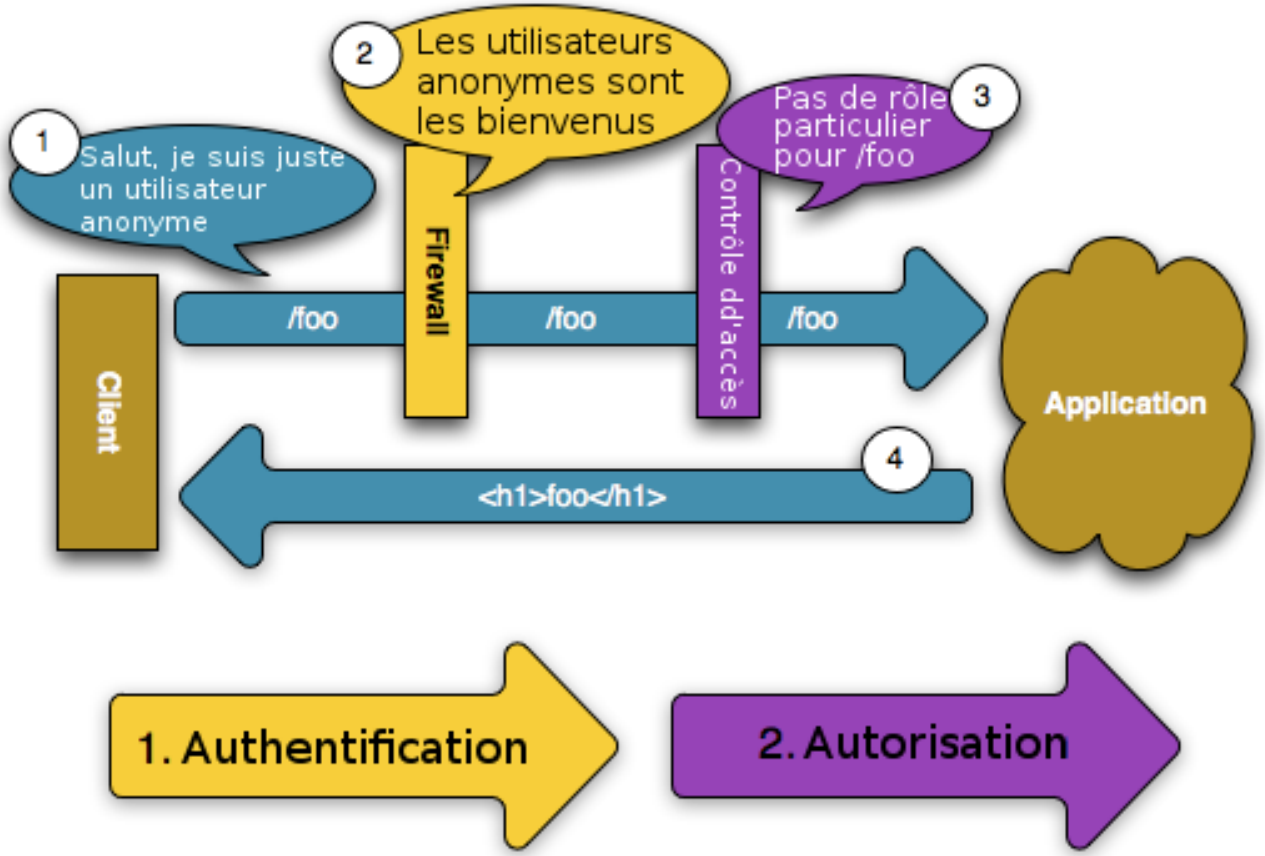
L'**authentification** est le processus permettant de savoir **qui** on est en tant que visiteur.

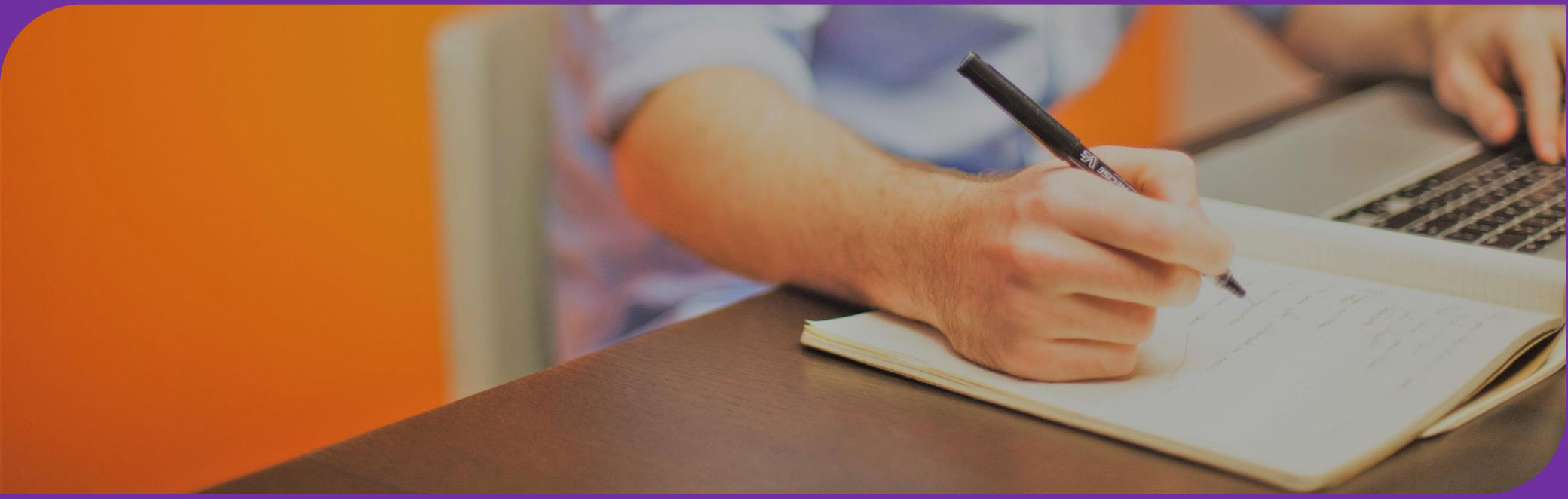
C'est le rôle du **Firewall** de gérer ça.

L'**autorisation** est le processus qui va déterminer si vous avez le **droit** d'accéder à la ressource demandée (il agit après le firewall qui peut être vu comme un filtre).

La Sécurité

Exemple : un visiteur **anonyme** souhaite accéder à la page /foo. Cette page ne requiert pas de droits particuliers, donc tous ceux qui ont réussi à passer le firewall peuvent y avoir accès. La figure suivante montre le processus.





2 L’authentification

La classe User

Si ce n'est pas déjà fait, require le bundle maker-bundle :

$ composer require maker-bundle

Pour créer automatiquement la classe User, utiliser la ligne de commande suivante :

$ php bin/console make:user

Il faudra alors spécifier certains éléments :

→ **Le nom de la classe :** la norme est de nommer la classe « User »

→ **Si les utilisateurs seront stockés dans la BDD :** la norme veux que oui

→ **L'élément « d'affichage » de l'utilisateur :** au choix, il est conseillé que ce soit l'email ou le username

→ **L'encodage du mot de passe :** il faut accepter sinon les mots de passe seront stockés en clair dans la BDD

La classe User

Ensuite il faut effectuer une migration pour actualiser la BDD !

Pour rappel :

$ php bin/console make:migration

$ php bin/console doctrine:migrations:migrate

→ Vous pouvez observer l'entité User qui a été créée automatiquement avec son repository.

Mise en place de l'authentification

Maintenant que la classe User a été créée, il est possible de générer automatiquement un **formulaire de connexion**.

Pour générer le formulaire de connexion, il faut utiliser la ligne de commande :

$ php bin/console make:auth

Il faudra alors spécifier certains éléments :

→ **Le type d'authentification :** nous choisirons le « Login form authentificator » dans le cadre de notre formation

→ **Le nom de l'entité d'authentification:** il faut le nommer de manière à comprendre qu'il s'agit de l'authentificateur, par exemple « LoginFormAuthentificator »

→ **Le nom du contrôleur d'authentification:** il est fortement conseillé de le nommer « SecurityController »

Mise en place de l'authentification

Petit point de configuration :

Il faut définir manuellement vers quelle route l'utilisateur sera redirigé une fois qu'il sera identifié

Cela se fait dans l'**entité** (peut-être appelée LoginFormAuthentificator si vous avez suivis la slide précédente), dans la méthode « **onAuthenticationSuccess** ».

Vous pouvez juste décommenter la ligne et ajouter le **nom** de votre route entre guillemets

Mise en place de l'authentification

Quelques éléments des fichiers générés :

- Un contrôleur :

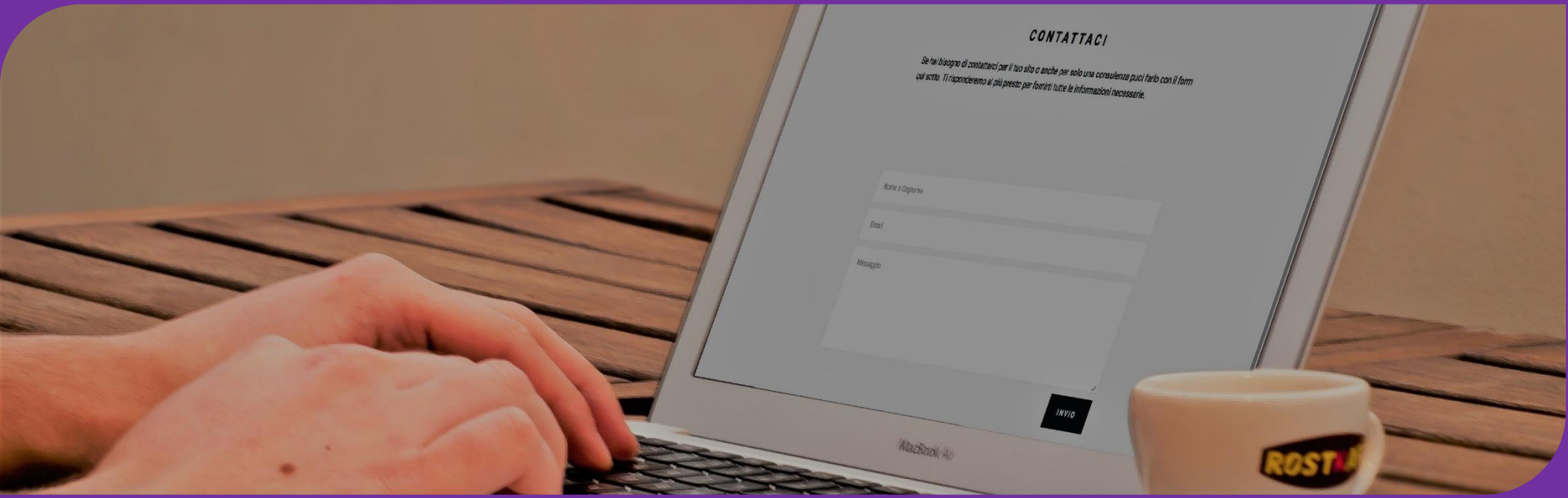
```php
/**
 * @Route("/login", name="app_login")
 */
public function login(AuthenticationUtils $authenticationUtils): Response
{
    // Reçoit l'erreur de login s'il y en a une
    $error = $authenticationUtils->getLastAuthenticationError();
    // Dernier username entré par le user pour pouvoir le rendre à la vue
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render( view: 'security/login', ['last_username' => $lastUsername, 'error' => $error]);
}
```

Mise en place de l'authentification

Quelques éléments des fichiers générés :

- Un formulaire :

```php
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add( child: 'username')
        ->add( child: 'plainPassword', type: PasswordType::class, [
            // instead of being set onto the object directly,
            // le mot de passe est lu et encodé dans le controller
            'mapped' => false,
            'constraints' => [
                new NotBlank([
                    'message' => 'Entrez un mot de passe',
                ]),
                new Length([
                    'min' => 6,
                    'minMessage' => 'Votre mot de passe doit faire minimum {{ limit }} caractères',
                    // Le max accordé par Symfony pour des raisons de sécurité
                    'max' => 4096,
                ]),
            ],
        ])
    ;
}
```



3 L’Inscription

Mise en place de l'inscription

Pour créer automatiquement le formulaire d'enregistrement, utiliser la ligne de commande suivante :

$ php bin/console make:registration-form

Il faudra alors spécifier certains éléments :

→ **Si l'utilisateur est unique :** ainsi son signe distinctif (choisi au préalable) ne peut pas être réutilisé lors d'un inscription

→ **Si l'utilisateur qui vient de l'inscrire est automatiquement connecté**

Mise en place de l'inscription

Quelques éléments des fichiers générés :

- Un contrôleur :

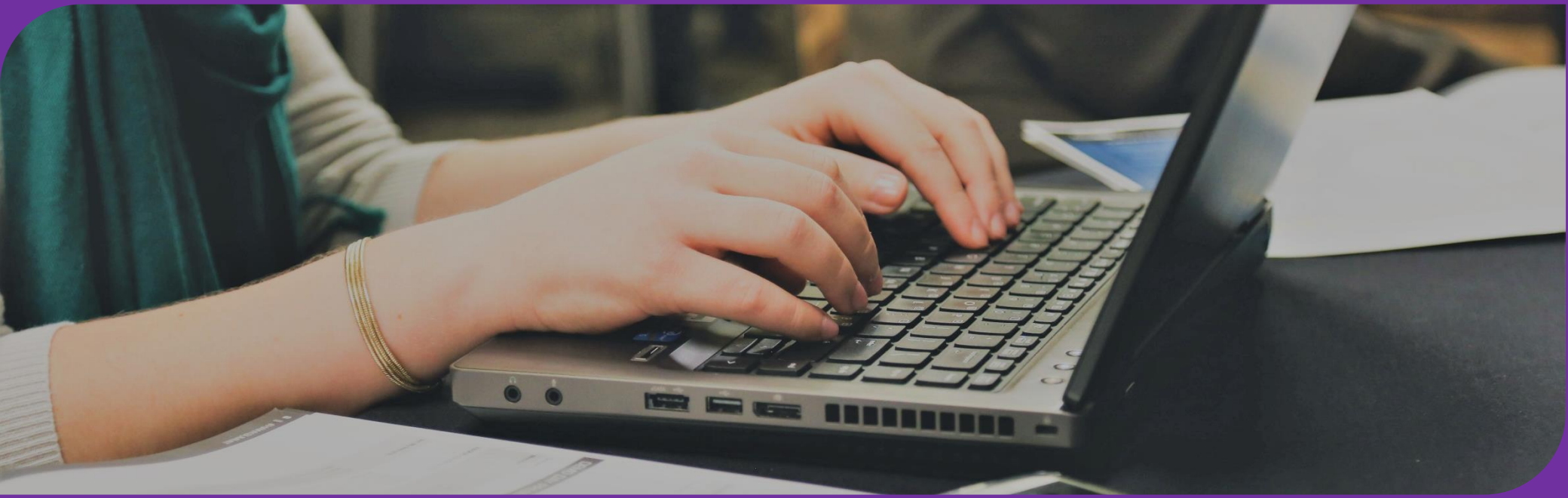
```php
/**
 * @Route("/register", name="app_register")
 */
public function register(Request $request, UserPasswordEncoderInterface $passwordEncoder,
                         GuardAuthenticatorHandler $guardHandler,
                         LoginAuthenticator $authenticator): Response
{
    $user = new User(); // Création d'un nouveau user
    $form = $this->createForm( type: RegistrationFormType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) { // Test du form
        // Encodage du mot de passe
        $user->setPassword(
            $passwordEncoder->encodePassword(
                $user,
                $form->get('plainPassword')->getData()
            )
        );
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($user);
        $entityManager->flush(); // Envoie dans la base de donnée

        return $guardHandler->authenticateUserAndHandleSuccess( // On connecte directement le new user
            $user,
            $request,
            $authenticator,
            providerKey: 'main'
        );
    }
    return $this->render( view: 'registration/register', [ // Rendu de la view du form
        'registrationForm' => $form->createView(),
    ]);
}
```

Mise en place de l'inscription

Quelques éléments des fichiers générés :

- Un formulaire :

```php
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add( child: 'username')
        ->add( child: 'plainPassword',  type: PasswordType::class, [
            // A la place d'être modifié dans l'objet directement,
            // le mot de passe est lu et encodé dans le controller
            'mapped' => false,
            'constraints' => [
                new NotBlank([
                    'message' => 'Entrez un mot de passe',
                ]),
                new Length([
                    'min' => 6,
                    'minMessage' => 'Votre mot de passe doit faire minimum {{ limit }} caractères',
                    // Le max accordé par Symfony pour des raisons de sécurité
                    'max' => 4096,
                ]),
            ],
        ])
    ;
}
```



4 Configuration

security.yaml

Normalement votre fichier security.yaml doit ressembler à ça :

```yaml
security:
    encoders:
        App\Entity\User:
            algorithm: argon21

    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: username

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: true
            guard:
                authenticators:
                    - App\Security\LoginAuthenticator
```

Ici est configuré d'**encodage** des mots de passes

Le **fournisseur d'utilisateurs**. Les firewalls s'adressent aux providers pour récupérer les utilisateurs et les identifier.

Permet d'avoir un faux firewall pendant la période de **développement**, et un « main » firewall où on définit l'**authentificateur** (ou d'autres configurations si souhaité).

Access_control

Nous allons ajouter une quelques lignes au fichier **security.yaml**

« **access_control** » permet de définir quel routes seront **autorisées** à quelles **types** d'utilisateurs. La synthaxe est la suivante dans le cas où on souhaite différencier les anonymes des utilisateurs et des admins :

```
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/article, roles: ROLE_USER }
```

En définissant cela, toutes les routes **/admin/...** seront réservées aux administrateurs, les routes **/articles/...** aux utilisateurs, etc.

/logout

La déconnexion se configure dans le fichier security.yaml également :

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        anonymous: true
(1)     logout:
            path: /logout (2)
            target: /
        guard:
            authenticators:
                - App\Security\LoginAuthenticator
```

On définit que si la route au nom « **logout** » (1) est appellée, alors on exécute **/logout** (2) qui ferme la session puis qui redirige vers la route **/** (on peut donc modifier la page sur laquelle l'utilisateur déconnecté est redirigé)

/logout

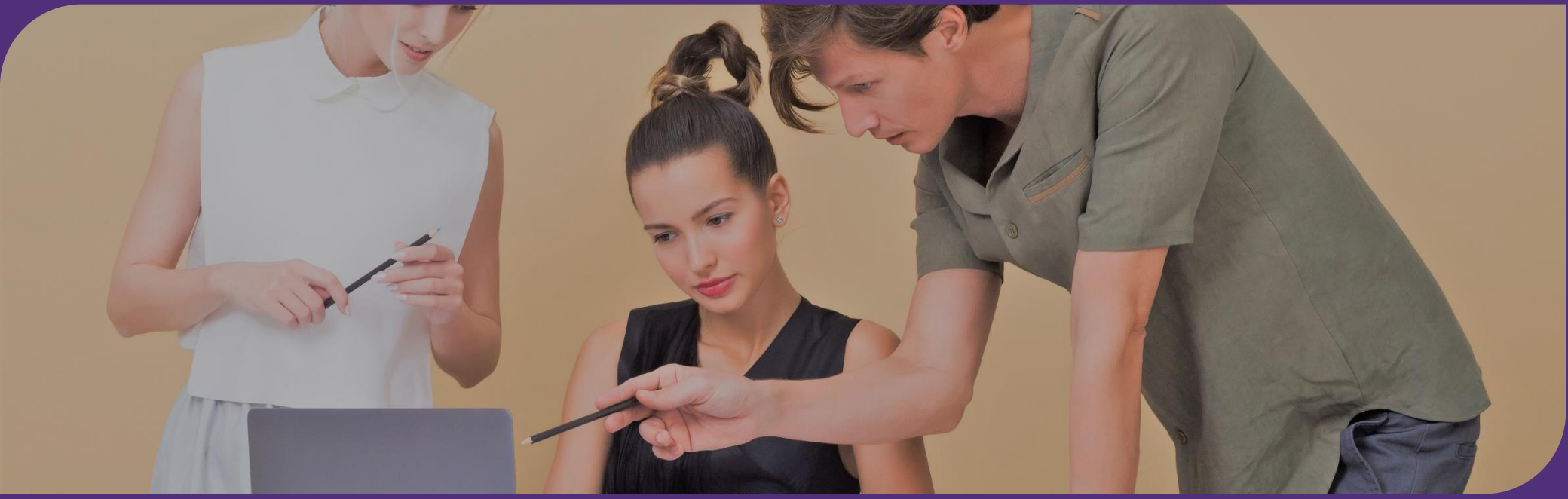
Cependant la route « logout » (1) malheureusement ne figure dans aucun contrôleur, elle n'est donc pas fonctionnelle.

Pour qu'elle le soit, il faut forcément la définir en **YAML** (alors que jusque la nous avons utilisé les **annotations**).

Pour ce faire, il faut se rendre dans le fichier **route.yaml** se trouvant dans le dossier config et ajouter les lignes suivantes :

```
logout:
  path: /logout
```

Maintenant il suffit de créer un bouton de déconnection dans vos vues avec comme path : « **logout** » (1)

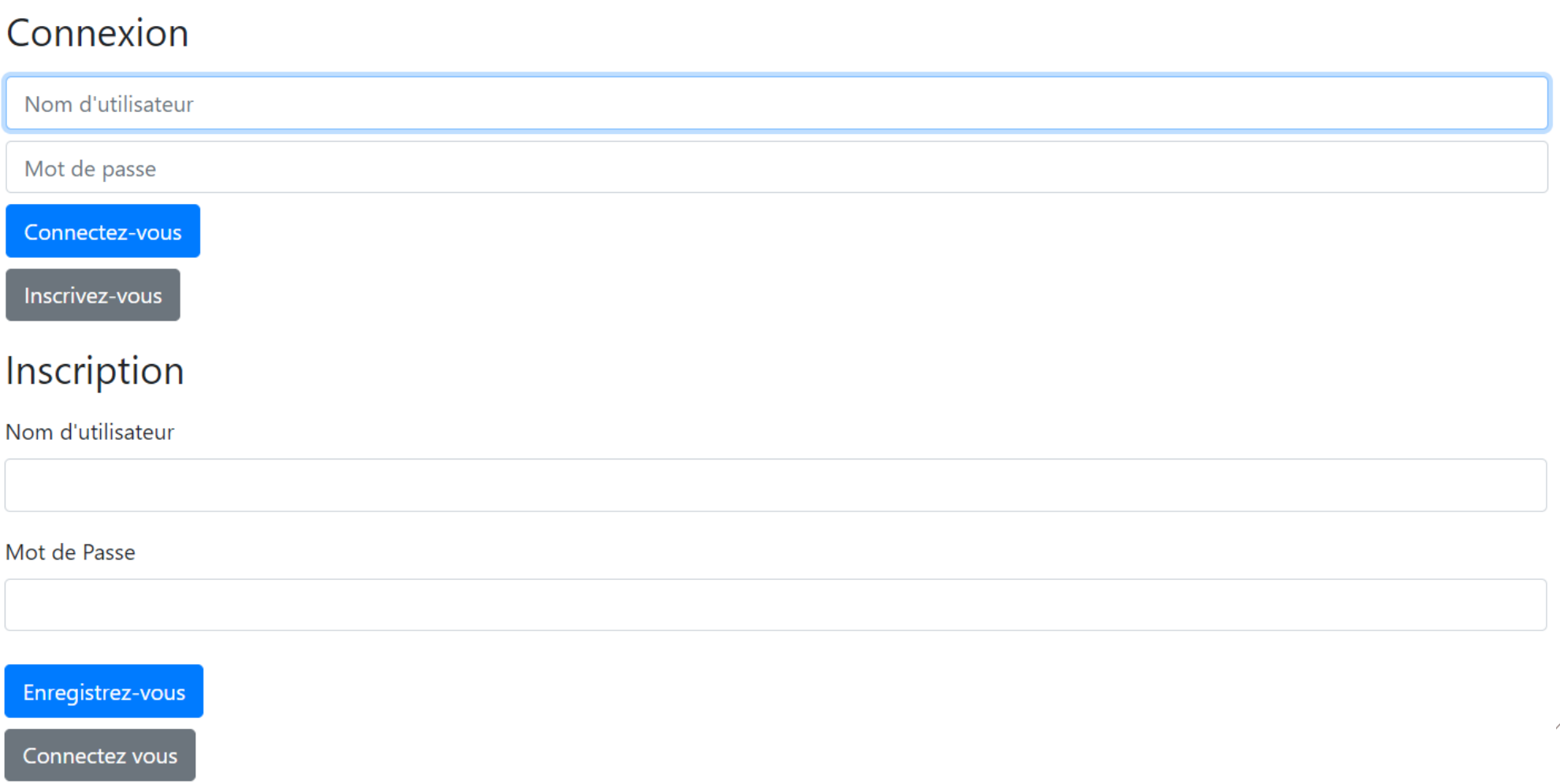


7

TP

TP

Objectif : création d'un formulaire de connexion et d'un formulaire d'inscription



Merci d'avoir participé à cette formation Junior ISEP !

N'hésitez pas à nous contacter :

- lprogent@juniorisep.com
- pmarquet@juniorisep.com
- ffavole@juniorisep.com

Quelques petits conseils pour la fin

Super site pour trouver des **bundles** :

https://flex.symfony.com/

Et surtout : **la doc de Symfony** !! Elle est très bien faite ca doit devenir votre premier réflexe si vous cherchez quelque chose ☺

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