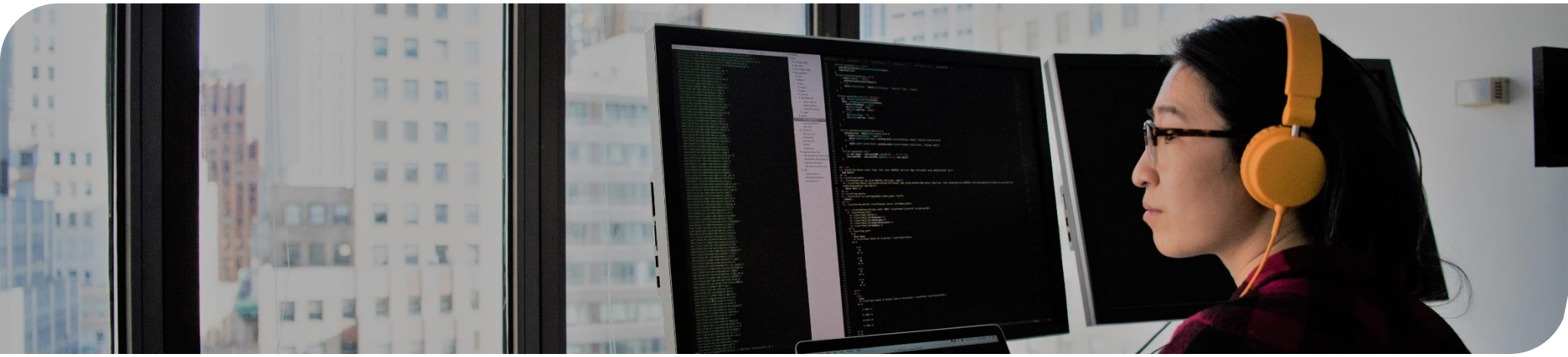
JUNIORISEP



Formation Symfony 5.0

Cours 3

Lundi 6 janvier | Pierre MARQUET & Laurianne PROGENT & Florent FAVOLE

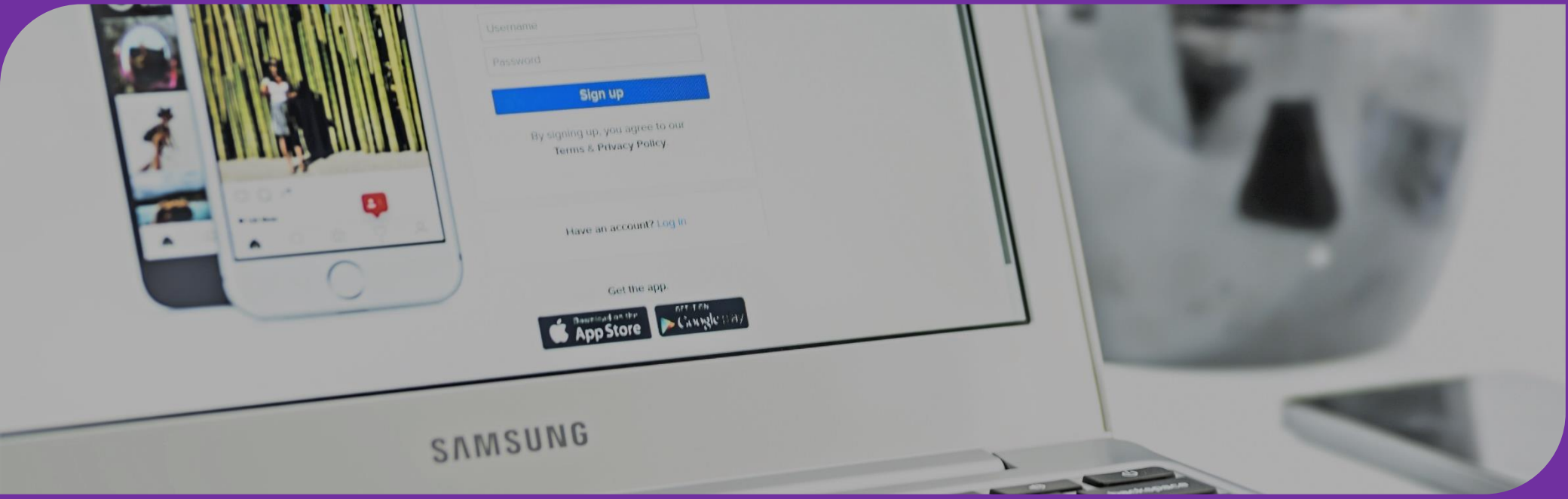
Symfony

Récapitulatif cours 2

Vu la semaine dernière :

- **Les Entités :** EntityManager et Repository
- **Doctrine :** BDD, migration, etc.
- **CRUD :** fonctionnement et mise en place
- **Relations entre entités (bonus) :** OneToMany, ManyToMany, etc.

Des questions ?



1 Les Formulaires

Les formulaires

Formulaire : permet à l'utilisateur d'**interagir** directement avec les données

Il se construit sur un **objet existant** et son objectif est de **remplir les attributs** de l'objet

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

$ composer require form

Il est possible d'en créer un automatiquement avec la commande suivante :

$ php bin/console make:form Nom

→ le nom final du fichier sera automatiquement **NomType**.php

Il faut lier le formulaire à l'**entité correspondante** (il faut que nom entré soit celui de l'entité)

Les formulaires se trouvent dans /src/Form/

Le builder

Dans le fichier **NomType.php** :

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add( child: 'title',  type: TextType::class)
        ->add( child: 'description',  type: TextareaType::class)
    ;
}
```

Pour **ajouter** un champ au builder :

-> add('nameChamp', champType::class)

→ Avec "champType" le type du champ et "nameChamp" le nom de l'attribut de votre Entité

Les types de champs

Text Fields	**Choice** Fields	**Date** and **Time** Fields	**Other** Fields	Field **Groups**	**Hidden** Fields	**Buttons**	**Base** Fields
TextType TextareaType EmailType IntegerType MoneyType NumberType PasswordType PercentType SearchType UrlType RangeType TelType ColorType	ChoiceType EntityType CountryType LanguageType LocaleType TimezoneType CurrencyType	DateType DateIntervalType DateTimeType TimeType BirthdayType	CheckboxType FileType RadioType	CollectionType RepeatedType	HiddenType	ButtonType ResetType SubmitType	FormType

Retourner/valider un formulaire

Dans le contrôleur :

```php
/**
 * @Route("/new", name="movie_new", methods={"GET","POST"})
 */
public function new(Request $request): Response
{
    $movie = new Movie(); // Création d'un nouvel objet
    $form = $this->createForm( type: MovieType::class, $movie); // Création d'un nouveau formulaire
    $form->handleRequest($request); // On récupère la requête

    if ($form->isSubmitted() && $form->isValid()) { // On vérifie si le formulaire est valide
        $entityManager = $this->getDoctrine()->getManager(); // On appelle l'EntityManager
        $entityManager->persist($movie); // On persiste l'objet
        $entityManager->flush(); // On exécute la persistence

        return $this->redirectToRoute( route: '/movie/'); // On est redirigé
    }

    return $this->render( view: 'movie/new', [ // Si le formulaire n'est pas valide ou pas submit
        'movie' => $movie,
        'form' => $form->createView(),
    ]);
}
```

Retourner/valider un formulaire

Dans le contrôleur :

```php
/**
 * @Route("/{id}/edit", name="movie_edit", methods={"GET","POST"})
 */
public function edit(Request $request, Movie $movie): Response
{
    $form = $this->createForm( type: MovieType::class, $movie); // Création d'un nouveau formulaire
    $form->handleRequest($request); // On récupère la requête

    if ($form->isSubmitted() && $form->isValid()) { // On vérifie si le formulaire est valide
        $this->getDoctrine()->getManager()->flush(); // On appelle l'EntityManager et on actualise la BDD

        return $this->redirectToRoute( route: '/movie/', [ // On est redirigé
            'id' => $movie->getId(),
        ]);
    }

    return $this->render( view: 'movie/edit', [ // Si le formulaire n'est pas valide ou pas submit
        'movie' => $movie,
        'form' => $form->createView(),
    ]);
}
```

L'objet Request

→ **Request** $request contient les informations de la requête de l'utilisateur.

Propriétés publiques utiles :

- **request :** équivalent de $_POST
- **query :** équivalent de $_GET
- **cookies :** équivalent de $_COOKIE
- **files :** équivalent de $_FILES
- **server :** équivalent de $_SERVER

Attention : Par défaut, le formulaire une fois soumis va générer une requête POST et renvoyer vers le contrôleur qui l'a affiché

Afficher un formulaire avec TWIG

De manière rapide :

```
{{ form_start(form) }}

    {{ form_widget(form) }}
    <button>Submit</button>

{{ form_end(form) }}
```

Toute la syntaxe :

Affiche tout le formulaire :
{{ form(form) }}

Affiche la balise <form> :
{{ form_start(form) }}

Affiche la balise de fermeture </form> :
{{ form_end(form) }}

Affiche le **label** HTML du champ donné :
{{ form_label(form.attribut) }}

Affiche les **erreurs** attachées au champ donné en argument :
{{ form_errors(form.attribut) }}

Affiche le **champ** HTML lui-même (balise input) :
{{ form_widget(form.attribut) }}

Affiche le label, les erreurs et le champ en même temps :
{{ form_row(form.attribut) }}

Affiche tous les champs **manquants** du formulaire :
{{ form_rest(form) }}

Exemple de formulaire

Exemple de formulaire plus complexe avec du bootstrap :

```
{{ form_start(form) }}

    {{ form_label(form.title, "Titre", {'label_attr': {'class': 'col-lg-2 control-label'}}) }}
    {{ form_errors(form.title) }}
    {{ form_widget(form.title, {'attr': {'class': 'col-lg-10'}}) }}

    {{ form_label(form.description, "Description", {'label_attr': {'class': 'col-lg-2 control-label'}}) }}
    {{ form_errors(form.description) }}
    {{ form_widget(form.description, {'attr': {'class': 'col-lg-10'}}) }}

    <div>{{ form_widget(form.save, {'attr': {'class': 'btn btn-success col-lg-2'}}) }}</div>

{{ form_end(form) }}
```

2 Les Contraintes

Les contraintes de validation

Contraintes : vérifications des données du formulaire côté serveur.

→ Elles se font au niveau des **entités** elle-même ou du **contrôleur**. Elles sont indispensables pour vous assurer que l'utilisateur ne rentre pas n'**importe quoi** dans le formulaire (en plus des vérifications côté client) ou que des objets soient **correctes** vis-à-vis des contraintes que vous avez spécifiées.

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

```
$ composer require validator
```

Importer le service dans l'entité souhaitée :

```
use Symfony\Component\Validator\Constraints as Assert;
```

Principaux types de contraintes

Contrainte	Description
NotBlank	Vérifie que la valeur soumise n'est ni une chaîne de caractères vide, ni null.
Blank	Vérifie que la valeur soumise est une chaîne de caractères vide ou null.
True	Vérifie que la valeur vaut true, 1 ou "1".
False	Vérifie que la valeur vaut false, 0 ou "0".
NotNull	Vérifie que la valeur est strictement différente de null.
Null	Vérifie que la valeur est strictement null.
Type	Vérifie que la valeur est bien du type donné en argument.
Email	Vérifie que la valeur est une adresse e-mail valide.
Length	Vérifie que la valeur donnée fait au moins X ou au plus Y caractères de long.

Principaux types de contraintes

Contrainte	Description
Url	Vérifie que la valeur est une adresse URL valide.
Regex	Vérifie la valeur par rapport à une regex.
Ip	Vérifie que la valeur est une adresse IP valide.
Language	Vérifie que la valeur est un code de langage valide selon la norme.
Locale	Vérifie que la valeur est une locale valide. (exemple : fr ou fr_FR)
Country	Vérifie que la valeur est un code pays en 2 lettres valide. (exemple : fr)
Range	Vérifie que la valeur ne dépasse pas X, ou qu'elle dépasse Y.
Date	Vérifie que la valeur est un objet de type Datetime/une chaîne de caractères du type YYYY-MM-DD.
Time	Vérifie que la valeur est un objet de type Datetime/une chaîne de caractères du type HH:MM:SS.

Principaux types de contraintes

Contrainte	Description
DateTime	Vérifie que la valeur est un objet de type Datetime/une chaîne de caractères du type YYYY-MM-DD HH:MM:SS.
File	Vérifie que la valeur est un fichier valide, c'est-à-dire soit une chaîne de caractères qui pointe vers un fichier existant, soit une instance de la classe File (ce qui inclut UploadedFile).
Image	Vérifie que la valeur est valide selon la contrainte précédente File (dont elle hérite les options), sauf que les mimeTypes acceptés sont automatiquement définis comme ceux de fichiers images. Il est également possible de mettre des contraintes sur la hauteur max ou la largeur max de l'image.

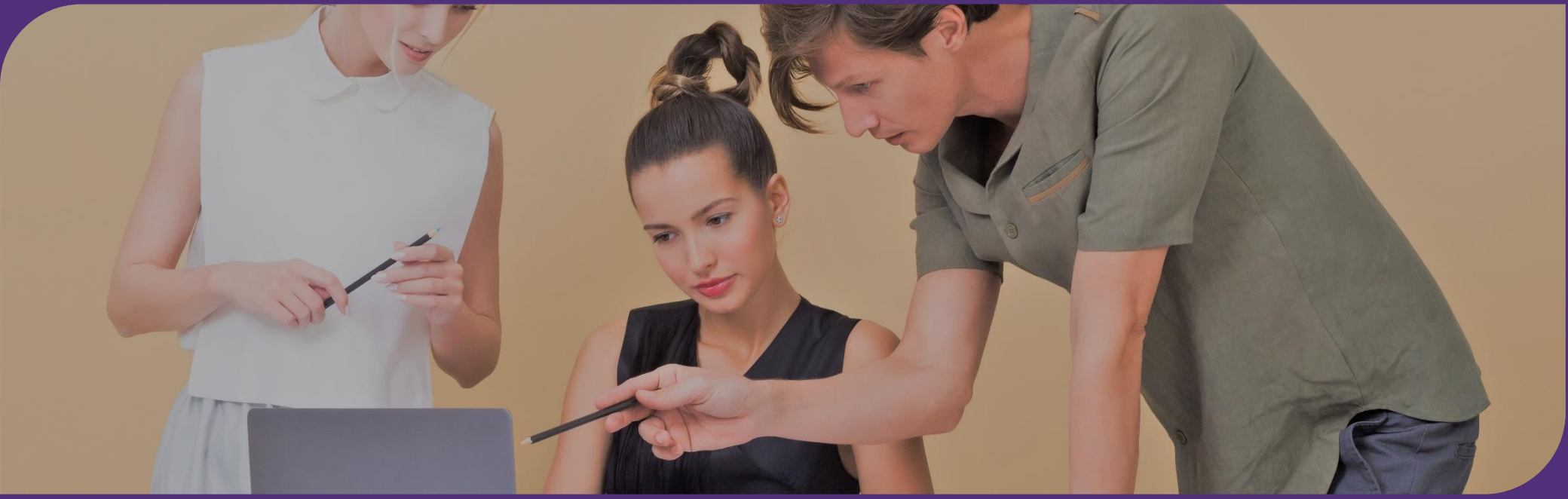
Exemples

```php
/**
 * @ORM\Column(type="string", length=255)
 * @Assert\Length(
 *      min=5,
 *      max=30,
 *      minMessage="Le titre est trop court.",
 *      maxMessage="Le titre est trop long."
 * )
 */
private $title;

/**
 * @ORM\Column(type="text")
 * @Assert\NotBlank()
 */
private $description;
```

Pour étudier les différentes options de chaque contraintes vous pouvez aller regarder la documentation, très complète, de Symfony :

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



7

TP

TP

But du TP : Améliorer notre CRUD

Pour cela on ajoutera :

- un système de contrainte / validation des formulaires
- une meilleure disposition des formulaires
- la prise en charge d'une image

Bonus : mettre en place un système de traduction pour les entités

TP


Blog Formation Symfony
Accueil
Création d'Article
Titre
ERREUR Ce titre est trop court
a
Auteur
ERREUR Cette chaîne est trop courte. Elle doit avoir au minimum 5 caractères.
a
Contenu
ERREUR Cette chaîne est trop courte. Elle doit avoir au minimum 15 caractères.
a
Fichier Image
Browse
Date de Création
27
mai
2019
15
04
Sauvegarder
Revenir à la liste

TP

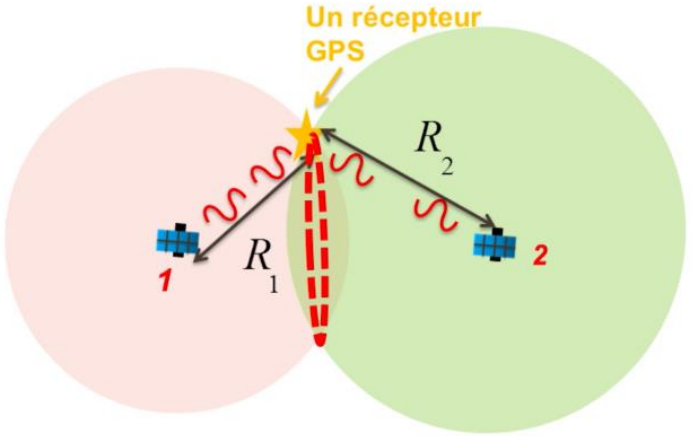

Blog Formation Symfony
Accueil
Administration des articles
Déconnexion
Création d'Article
Titre
Auteur
Contenu
Fichier Image
Browse
Date de Création
27
mai
2019
15
04
Sauvegarder
Revenir à la liste

TP

Blog Formation Symfony Accueil

quasi dolor ut

$$R_1 = c.\Delta t_1$$
$$R_2 = c.\Delta t_2$$



Auteur : André Joly Date de publication : 2019-05-13

Lorem ipsum dolor sit amet consectetur adipisicing elit, sed do eiusmod tempor lectus eget felis incididunt ut labore et **dolore magna aliqua**: Duis aute irure dolor in sunt in culpa reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Pellentesque tincidunt excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia occaecat cupidatat deserunt mollit anim id est laborum.

* Ut enim ad minim veniam
* Quis nostrud exercitation *ullamco laboris*
* Nisi ut aliquip ex ea commodo consequat

Merci d’avoir participé à cette formation Junior ISEP !

N’hésitez pas à nous contacter :

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