



FORMULAIRES & VALIDATION

1. FORMULAIRES

Afficher un formulaire et traiter la saisie utilisateur



Symfony

FORMS

Composant symfony/form

Classe définissant les champs et leurs configurations (type, widget, requis ou non, validation, ...)

Après la soumission du formulaire, les données saisies sont mappées sur une entité Doctrine, un objet ou un tableau

Convention: les classes formulaires sont situées dans App\Form et suffixées par Type

Maker peut générer un formulaire à partir d'une entité



FORMS

Types de champs disponibles:

- TextType
- NumberType
- UrlType
- ColorType
- DateTimeType
- BirthdayType
- EntityType
- ...

Liste complète sur

<https://symfony.com/doc/current/reference/forms/types.html>

FORMS

```
● ● ●

class CreateArticleType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('title')
            ->add('content')
            ->add('tags', EntityType::class, [
                'class' => Tag::class,
                'choice_label' => 'id',
                'multiple' => true,
            ])
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Article::class,
        ]);
    }
}
```

GÉRER UN FORMULAIRE



Symfony

Dans un controller:

- initialiser une instance formulaire à partir du type
- associer à la requête, qui contient la saisie utilisateur
- effectuer un traitement si le formulaire a été soumis et est valide

FORMS

```
#[Route('/articles/create', name: 'articles_create', methods: ['GET', 'POST'])]
public function index(Request $request): Response
{
    $form = $this->createForm(CreateArticleType::class);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $article = $form->getData();

        $this->entityManager->persist($article);
        $this->entityManager->flush();

        return $this->redirect('articles_list');
    }

    return $this->render('articles/create/index.html.twig', [
        'form' => $form->createView(),
    ]);
}
```



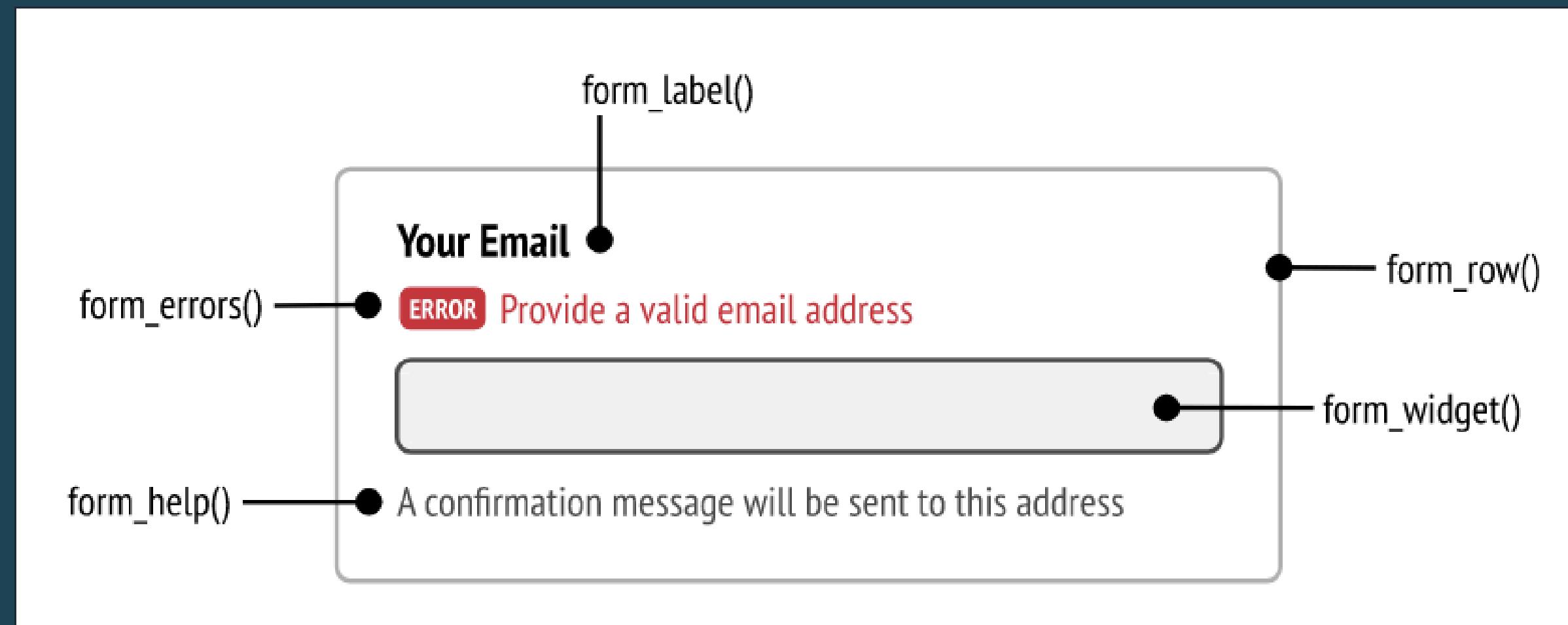
TWIG

Méthodes pour faciliter l'affichage des champs et des messages d'erreur en HTML



```
 {{ form_start(form) }}  
 {{ form_row(form.title) }}  
 {{ form_row(form.content) }}  
 {{ form_row(form.tags) }}  
  
 <input type="submit" value="Save" />  
 {{ form_end(form) }}
```

FORMS



2. ATELIER

Créer et éditer un article



ATELIER - FORMULAIRE

Créer une classe de formulaire pour l'entité Article

Créer un nouveau Controller et un template twig, qui permettent d'afficher ce formulaire

Créer une autre page pour mettre à jour un article

Enregistrer l'article en base de données à la soumission du formulaire

En option: mettre en place un message flash de succès

Saisir des données invalides dans le formulaire



Symfony

PLUS LOIN

Possibilité de configurer un thème pour avoir un rendu HTML conforme à Bootstrap, Tailwind ou Foundation

Upload de fichiers, utiliser la librairie vich/uploader-bundle

Les évènements de formulaires permettent de dynamiser le comportement des champs



Symfony

CSRF

Cross-Site Request Forgery

Mécanisme de protection activé par défaut sur tous les formulaires

Ajoute un champ masqué `_token`, contenant un hash spécifique à chaque utilisateur

Bonne pratique: ne pas le désactiver

3. VALIDATION

Contrôler les données et afficher les erreurs



ATELIER - PREAMBULE

Retirer Stimulus, puis nettoyer importmap.php



```
composer remove symfony/stimulus-bundle symfony/ux-turbo
```



Symfony

VALIDATION

Composant symfony/validator

Fourni:

- un ensemble de contraintes
- un service permettant de vérifier la validité d'une donnée (objet, tableau, scalar) et de retourner la liste des violations

La configuration des contraintes se fait via les attributs PHP

VALIDATION - ATTRIBUTS

```
class Article
{
    #[ORM\Column(length: 255)]
    #[Assert\NotBlank]
    #[Assert\Length(
        min: 5,
        max: 255,
        minMessage: 'Title is too short',
        maxMessage: 'Title is too long',
    )]
    private ?string $title = null;

    #[ORM\Column(type: Types::TEXT)]
    #[Assert\NotBlank]
    private ?string $content = null;
}
```



CONTRAINTES

Contraintes disponibles:

- Type
- PasswordStrength
- Range
- Iban
- Country
- UniqueEntity
- Expression
- ...

Liste complète sur

<https://symfony.com/doc/current/reference/constraints.html>

!

3. ATELIER

Ajouter des contraintes sur l'entité Article



ATELIER - VALIDATION

Ajouter des contraintes sur l'entité Article

- longueur du titre entre 5 et 255 caractères
- le titre et le contenu ne doivent pas être vides

Vérifier que les messages d'erreurs sont bien affichés sur l'interface

En option: traduire les messages

En option: analyser l'objet ConstraintViolationList