

# Les formulaires en Symfony

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Installation et utilisation des formulaires</b>	<b>3</b>
A. Installation et utilisation des formulaires .....	3
B. Exercice : Quiz .....	6
<b>III. La gestion du formulaire dans le contrôleur et son affichage</b>	<b>6</b>
A. La gestion du formulaire dans le contrôleur et son affichage .....	6
B. Exercice : Quiz .....	10
<b>IV. Essentiel</b>	<b>11</b>
<b>V. Auto-évaluation</b>	<b>11</b>
A. Exercice .....	11
B. Test .....	12
<b>Solutions des exercices</b>	<b>12</b>

## I. Contexte

**Durée :** 1 H

**Environnement de travail :** un ordinateur connecté à Internet

**Pré requis :**

Notion de PHP

Notion de POO

### Contexte

Afin de pouvoir dialoguer avec l'utilisateur, il est courant d'utiliser des formulaires, appelés *forms* en anglais. Ces informations récupérées peuvent l'être de façon temporaire ou alors stockées en base de données. En HTML la gestion des formulaires est une tâche répétitive et complexe. En plus de gérer le rendu des champs de formulaire et la validation des données rentrées par l'utilisateur, il faut gérer le mappage des objets constitués vers la base de données.

En PHP lors de l'utilisation des formulaires, il faut récupérer les éléments entrés par l'utilisateur grâce à `$POST[]`, qui est un tableau associatif de variables que la méthode HTTP POST crée à la soumission d'un formulaire.

Avec Symfony il sera bien sûr possible de récupérer des informations par texte, liste déroulante, bouton radio, etc. Cependant, le plus souvent la récupération se fera à partir d'un `FormType`, une classe PHP permettant de mapper les données du formulaire, ce qui donne l'avantage d'être dissocié de la vue et permet une réutilisation de ce même formulaire.

Voici le workflow du composant form recommandé: le formulaire depuis un contrôleur ou depuis une classe dédiée (form type) doit être rendu du côté view via Twig avec les fonctions ou filtres du composant form. Ensuite le formulaire après envoi doit être traité, validé et utiliser cette donnée PHP soit pour traitement direct, soit pour persistance en BDD via le mapping auto des `data_class`.

Mais concrètement comment s'y prend-on ? C'est ce que nous verrons dans ce cours.

## II. Installation et utilisation des formulaires

### A. Installation et utilisation des formulaires

#### Définition

#### Formulaire

Un formulaire Web est un espace dédié sur une page web sur lequel un internaute peut saisir les informations demandées. Il peut être composé d'un champ de texte, de menus déroulants, de cases à cocher ou tout autre moyen qui permet de donner une information. Il y a différents types de formulaires, mais le plus souvent on rencontre des formulaires d'inscription, de contact et de commande. Bien sûr, il peut revêtir toute forme ou format. Les informations récoltées sont souvent stockées en base de données.

Les formulaires offrent une véritable interaction avec l'utilisateur. Non seulement elles apportent des informations précieuses mais permettent aussi d'évaluer et de mesurer son intérêt pour le contenu du site web. Pour l'internaute, remplir un formulaire nécessite de la confiance. Cela engage le site à mettre en place une sécurisation des données et parfois de les crypter ou les hacher comme les mots de passe par exemple.

Comme abordé en introduction, voici les 3 étapes d'un formulaire :

- Construire un formulaire dans un contrôleur ou en utilisant une classe de formulaire dans le dossier Form
- Rendre le formulaire dans un modèle, c'est-à-dire faire en sorte que le contrôleur appelle une page Twig avec les champs du formulaire
- Traiter le formulaire, ce qui implique une validation est une transformation des données, le plus souvent en objet pour pouvoir être transféré dans une base de données

## L'installation dans un projet Symfony

Dans les applications utilisant Symfony Flex, qui est facultatif même s'il est installé par défaut depuis Symfony 4, vous pouvez installer Symfony Form avec la ligne de commande suivante ;

```
1 $ composer require Symfony/form
```

## Type de formulaires et Type de champs

Un « Type » de formulaire est une classe PHP. Il permet de construire un formulaire et de définir les différents types de champs. Cette classe, si vous utilisez les outils Symfony comme MakerBundle, sera créée dans le dossier Form et portera par convention le nom de l'entité suivi du mot « Type ». Ainsi un formulaire pour une entité User se nommera « UserType » et aura le chemin d'accès : App\Form\UserType.

Cette classe devra étendre la classe AbstractType, ce qui lui donnera 2 méthodes buildForm() et configureOptions().

### BuildForm()

BuildForm() dispose d'un constructeur de formulaires \$builder (nom de variable donné par défaut) qui est un objet FormBuilderInterface. \$builder possède donc quelques méthodes dont une indispensable qui est add(). La méthode add() va permettre d'ajouter tous les champs que l'on souhaite avoir dans le formulaire. Pour cela, il faudra mettre en premier paramètre le nom du champ et son type.

#### Exemple

```
1 class CommentType extends AbstractType
2 {
3     public function buildForm(FormBuilderInterface $builder, array $options)
4     {
5         $builder
6             ->add('title', TextType::class)
7             ->add('content', TextareaType::class)
8     ;
9 }
```

La classe CommentType possède la méthode buildform() dans l'exemple ci-dessus. On constate également la présence de \$builder qui a pour méthode add() avec « title » qui est typé en TextType et « content » qui est typé en TextareaType.

Notez que TextType est le type par défaut, ainsi nous aurions pu écrire :

```
->add('title')
```

De même, nous venons de créer un formType dans l'exemple ci-dessus « CommentType », que vous pourrez inclure dans d'autres formulaires par la suite. Chaque méthode dans un formType possède un Type. FormBuilderInterface en fournit de nombreux qui ont été créés pour les utilisateurs. Sur la documentation officielle de Symfony (Symfony<sup>1</sup>), vous y trouverez la liste à jour. Pour Symfony 5.4, il y a 13 champs de texte, 8 champs de choix,

1 <https://symfony.com/doc/5.4/reference/forms/types.html>

6 champs de Date et *Time* ainsi que le *CheckboxType*, *FileType* et le *RadioType* en autres champs. On y trouve aussi des boutons comme le *ButtonType*, *ResetType*, *SubmitType*, des groupes de champs avec *CollectionType* et *RepeatedType*, 2 champs UX, 2 champs UID ainsi que le *HiddenType* et le *FormType*.

Le troisième argument de `add()`, qui est facultatif, est un tableau associatif. Il existe plusieurs options disponibles. Mais parlons des plus utilisés, à savoir « *required* » et « *label* » et « *attr* » :

- « *required* » peut être obligatoire si le champ de votre entité a l'option non nul, mais le préciser apporte plus de sécurité et permet une bonne lisibilité.
- « *label* » utilise par défaut le nom de la propriété, mais en l'ajoutant en paramètre vous pouvez personnaliser votre label.
- « *attr* » permet d'attribuer des classes CSS.

#### Exemple

Voici des exemples avec les options mentionnées ci-dessus :

```
1 class CommentType extends AbstractType
2 {
3     public function buildForm(FormBuilderInterface $builder, array $options)
4     {
5         $builder
6             ->add('title', TextType::class, [
7                 'attr' => [
8                     'class' => 'form-control mt-3 mb-3',
9                 ],
10                'label' => 'titre du commentaire',
11                'required' => true
12            ]
13        )
14        ->add('content', TextareaType::class, [
15            'attr' => [
16                'class' => 'form-control mt-3 mb-3',
17            ],
18            'label' => 'Commentaire',
19            'required' => true
20        ]
21    )
22    ;
23 }
```

#### configureOptions()

`configureOptions()` permet de créer un système d'options avec les options requises, les valeurs par défaut, la validation, la normalisation et plus encore.

Une bonne pratique est de le préciser dans `configureOptions()` l'entité sur laquelle vous travaillez dans le `formType`. Il faudra, comme le montre l'exemple ci-dessous, faire cette déclaration. Même si cela est optionnel, cela vous évitera des désagréments en cas de code qui se complexifie.

#### Exemple

```
1 public function configureOptions(OptionsResolver $resolver): void
2     {
3         $resolver->setDefaults([
4             'data_class' => Comment::class,
5         ]);
6     }
```

## B. Exercice : Quiz

[solution n°1 p.13]

### Question 1

buildform() est :

- ☐ Une méthode de \$builder
- ☐ Une méthode de Symfony
- ☐ Une méthode de FormBuilderInterface

### Question 2

La méthode add() de buildForm() permet d'ajouter un formulaire.

- ☐ Vrai
- ☐ Faux

### Question 3

Comment définir une option pour un champ de formulaire dans Symfony ?

- ☐ En utilisant la méthode addOption()
- ☐ En utilisant un tableau d'options comme deuxième argument de la méthode add()
- ☐ En utilisant la méthode setOption()

### Question 4

Dans quel ordre les étapes de formulaire doit-on procéder ?

- ☐ Construire, rendre, traiter
- ☐ Rendre, construire, traiter
- ☐ Traiter, rendre, construire

### Question 5

Qu'est-ce qu'un Type de formulaire ?

- ☐ Un Type spécial
- ☐ Un Text Type
- ☐ Une classe PHP

## III. La gestion du formulaire dans le contrôleur et son affichage

### A. La gestion du formulaire dans le contrôleur et son affichage

Le contrôleur joue un rôle clé dans l'architecture MVC (Modèle-Vue-Contrôleur) de Symfony. Il va envoyer à la vue le formulaire et va ensuite traiter les informations recueillies lors de la soumission, du formulaire de l'utilisateur. Découvrons le fonctionnement du contrôleur, avec quelques exemples de sa gestion d'un formulaire, ainsi que les différentes façons d'utiliser la vue du formulaire dans un fichier Twig.

## createFormBuilder

Nous avons vu qu'une bonne pratique était de créer un formulaire à part sous forme de Type. D'ailleurs, si vous utilisez MakerBundle pour la création d'une entité, il vous le créera automatiquement à partir des propriétés de l'entité que vous venez de créer. Mais, il est possible de créer un formulaire directement dans le contrôleur grâce à la méthode que fournit AbstractController : createFormBuilder().

### Exemple

```
1
2 #[Route('/comment')]
3 class CommentController extends AbstractController
4 {
5     #[Route('/new', name: 'app_comment_new', methods: ['GET', 'POST'])]
6     public function new(Request $request) : Response
7     {
8         $comment = new Comment();
9         $form = $this->createFormBuilder($comment)
10             ->add('title', TextType::class, [
11                 'attr' => [
12                     'class' => 'form-control mt-3 mb-3',
13                 ],
14                 'label' => 'titre du commentaire',
15                 'required' => true
16             ])
17         )
18             ->add('content', TextareaType::class, [
19                 'attr' => [
20                     'class' => 'form-control mt-3 mb-3',
21                 ],
22                 'label' => 'Commentaire',
23                 'required' => true
24             ])
25         ->getForm();
26
27         // ...
28     }
29 }
```

Dans l'exemple ci-dessus, nous constatons que créer un formulaire directement dans le contrôleur ressemble beaucoup à la façon de créer un formulaire Type. Nous avons remplacé la méthode \$Build par \$form. Cette dernière appelle la méthode createFormBuilder(), qui a en paramètre l'entité voulue. Par la suite, la méthode add() se comporte comme avec BuildForm(). Enfin avant de clôturer, il faut appeler la méthode getForm().

### Exemple

N'oubliez pas d'importer toutes les classes qui sont utilisées, telles que TextType, TextareaType et Comment.

## Formulaire de rendu

Maintenant, nous voulons que notre application puisse afficher le formulaire. Dans l'exemple ci-dessous, nous mettons dans la variable \$form le formulaire en appelant la méthode createForm() dans laquelle on passe en argument CommentType. Attention à ne pas oublier de l'importer avec autoloader.

### Exemple

Voici un exemple de rendu avec le formulaire en paramètre :

```
1 #[Route('/comment')]
2 class CommentController extends AbstractController
3 {
4     #[Route('/new', name: 'app_comment_new', methods: ['GET', 'POST'])]
5     public function new(Request $request) : Response
6     {
7
8         $form = $this->createForm(CommentType::class);
9
10        return $this->render('comment.html.twig', [
11            'form' => $form,
12        ]);
13    }
14 }
```

C'est la méthode `render()` qui va appeler le fichier Twig, qui représente la Vue dans le MVC et qui va lui transmettre les données voulues, ici le formulaire `$form`.

### La vue

Le fichier `comment.html.twig`, à présent qu'il a reçu du contrôleur le formulaire, va devoir l'afficher.

La façon la plus simple de le faire est celle-ci :

### Exemple

```
1 {% block body %}
2     {{ form(form) }}
3     <button type="submit">Enregistrer</button>
4 {% endblock %}
```

Notez que le formulaire passé à Twig par le contrôleur se nomme « *form* », mais il aurait pu avoir une autre nomination. Ici la fonction `form()` du composant `symfony/form`, utile pour Twig, permet de rendre visuelles les balises `<form>` HTML ainsi que chacun des champs avec labels, inputs et autres éléments (help, erreurs, etc.). De plus, le bouton est de type « *submit* ». Une autre solution aurait été d'ajouter `->add('save', SubmitType::class)` dans le `Typeform` qui définit le formulaire.

Voici une autre façon de procéder :

### Exemple

```
1 {% block body %}
2     {{ form_start(form) }}
3         {{ form_label(form.title) }}
4         {{ form_widget(form.title) }}
5         {{ form_label(form.content) }}
6         {{ form_widget(form.content) }}
7         <button type="submit">Enregistrer</button>
8     {{ form_end(form) }}
9 {% endblock %}
```



Ici, le formulaire est plus détaillé, alors il faut commencer le formulaire avec `form_start` et finir avec `form_end`.

Puis, pour chaque champ on met la balise `form_label(form.nomChamp)` afin de mettre le label et `form_widget(form.nomChamp)` pour l'input. Il s'agit donc d'un exemple court et très simple. Sachez qu'en cas de formulaire avec de nombreux champs, il est possible de commencer l'énumération des champs puis d'appeler avant le bouton submit la fonction `form_rest(form)`.

Cependant, cette méthode n'est pas recommandée, car cela vous oblige à maintenir chacun des champs individuellement (CSS, gestion d'erreurs, etc.). Il est préférable d'empaqueter le plus possible et de rendre visuellement les champs en une seule fois avec la fonction Twig `{{ form(form) }}` ou les `{{ form_row(form.champ) }}` que l'on va voir juste après.

Voyons à présent une autre possibilité :

#### Exemple

```
1 {% block body %}
2     {{ form_start(form) }}
3         {{ form_row(form.title) }}
4         {{ form_row(form.content) }}
5         <button type="submit">Enregistrer</button>
6     {{ form_end(form) }}
7 {% endblock %}
```

Dans l'exemple ci-dessus, nous avons utilisé `form_row` qui est une combinaison de `form_label()`, `form_widget()` et `form_errors()`, que nous n'avons pas encore traité mais qui permet d'afficher les erreurs du champ. Vous pouvez notamment utiliser ces fonctions Twig `form_row()` lorsque vous voulez ajouter des options visuelles (css, placeholder, label) de façon individuelle. Néanmoins, il existe d'autres façons de le faire, comme les `form_themes` qui sont recommandées par Symfony.

Nous avons vu qu'il est possible d'ajouter des attributs lors de la création d'un formulaire, mais c'est aussi possible de le faire lors de l'affichage.

#### Exemple

```
1 {% block body %}
2     {{ form_start(form) }}
3         {{ form_row(form.title) }}
4         {{ form_row(form.content, {'attr': {'class': 'form-control', 'placeholder':
5 'Commentaire'}}) }}
6         <button type="submit">Enregistrer</button>
7     {{ form_end(form) }}
8 {% endblock %}
```

Dans l'exemple ci-dessus, on ajoute les attributs `class` et `placeholder` à `form.content`.

C'est un choix à faire : on déclare les attributs soit dans le Type (ou contrôleur), soit dans la vue comme dans l'exemple ci-dessus. On préconisera l'ajout de toutes les options dans le `FormType`, pour faciliter la maintenance dans le temps et respecter les principes KISS, SOLID des langages objets.

L'utilité de le déclarer dans la vue peut être intéressante : par exemple, dans le cas d'un développeur front-end qui s'occupe uniquement des fichiers Twig du projet. Il pourra ajouter des classes CSS sans avoir à coder en PHP.

## Traitement des formulaires

Symfony recommande de traiter le formulaire dans la même méthode que le rendu. Voici, dans l'exemple ci-dessous, la méthode `new()` de `CommentController` complétée pour le traitement des données.

```
1 #[Route('/comment')]
2 class CommentController extends AbstractController
3 {
4     #[Route('/new', name: 'app_comment_new', methods: ['GET', 'POST'])]
5     public function new(Request $request, EntityManagerInterface $em) : Response
6     {
7         $comment = new Comment();
8         $form = $this->createForm(CommentType::class);
9         $form->handleRequest($request);
10        if ($form->isSubmitted() && $form->isValid) {
11            $em->persist($comment);
12            $em->flush();
13            return $this->redirectToRoute('comment_list');
14        }
15        return $this->render('comment.html.twig', [
16            'form' => $form,
17        ]);
18    }
19 }
```

Qu'est-ce que nous avons fait concrètement ? La méthode `handleRequest()` va gérer le traitement de la saisie du formulaire après soumission de celui-ci par l'utilisateur. Ensuite, nous rentrons la condition conseillée par Symfony qui vérifie si la soumission est bonne et que le formulaire est valide, c'est-à-dire qu'il correspond à ce qui est attendu, notamment vis-à-vis des contraintes de validation qui sont ajoutées pour chaque champ de votre formulaire.

Dans la condition grâce `entityManager`, on met à jour la base de données, puis on redirige. La redirection est une bonne pratique, car elle évite que l'utilisateur actualise la page. Notez que nous ne parlerons pas de la validation dans ce cours, car cela mérite un cours à part entière.

## B. Exercice : Quiz

[solution n°2 p.14]

### Question 1

Quelle façon d'afficher un formulaire dans un fichier Twig ne fonctionnera pas ?

- ☐ `{{ form(form) }}`
- ☐ `{{ form_row(form.title) }}`
- ☐ `{{ form_start(form) }} {{ form_row(form.title) }} {{ form_end(form) }}`

### Question 2

Les méthodes « `form_row()` » et « `form_widget` » donnent le même résultat.

- ☐ Vrai
- ☐ Faux

### Question 3

« `createFormBuilder()` » est une méthode fournie par :

- ☐ Symfony
- ☐ Doctrine
- ☐ AbstractController

#### Question 4

La balise `form_rest()` permet d'afficher tous les champs qui n'ont pas été appelés.

- ☐ Vrai
- ☐ Faux

#### Question 5

La méthode « `handleRequest()` » joue un rôle à la soumission d'un formulaire.

- ☐ Vrai
- ☐ Faux

## IV. Essentiel

Les formulaires sont et resteront des composants majeurs afin de dialoguer avec les utilisateurs ou visiteurs d'un site internet. Que cela soit un simple formulaire de contact, d'inscription ou un véritable questionnaire, la gestion de ceux-ci reste essentielle.

Symfony fournit de nombreux outils qui facilitent le travail. `AbstractType` nous a permis de définir un formulaire dans un fichier PHP grâce à la méthode `buildForm()` et `AbstractController` quant à lui fournit `CreateFormBuilder()` dans le contrôleur. C'est grâce à eux qu'il est possible de personnaliser les champs de formulaire.

Le rendu, c'est-à-dire l'affichage, est aussi aisé avec le moteur de templates Twig. Il est possible d'appeler un formulaire à partir d'une seule ligne de code. Mais, il est aussi possible d'ajouter des attributs CSS au formulaire pour permettre à un développeur front-end de ne pas avoir à utiliser le fichier PHP.

Enfin, c'est la méthode `handlesubmit()` du contrôleur qui permet le traitement du formulaire.

Néanmoins, nous avons uniquement vu les principes de base d'un formulaire avec Symfony 5.4. Alors pour laisser libre cours à vos besoins et imagination, il va falloir pratiquer, pratiquer et encore pratiquer !

## V. Auto-évaluation

### A. Exercice

Vous êtes un développeur Full-stack Symfony dans une agence web avec un personnel limité et vous récupérez une application web qui nécessite une connexion et où les formulaires n'ont pas été créés avec les entités.

#### Question 1

[solution n°3 p.15]

Définissez une classe `UserType`, `User` ayant un email, un nom, un mot de passe et un mot de passe de confirmation. Attribuez à chaque champ la classe `form-control` parce que vous utilisez bootstrap ainsi qu'un label.

#### Question 2

[solution n°4 p.16]

Votre collègue du front-end ne peut pas vous donner un coup de main, vous allez maintenant devoir créer un formulaire dans un fichier Twig. Le titre sera « *Inscrivez-vous* », il aura pour classes CSS « *d-flex justify-content-center* », ainsi que toutes les div que vous créerez. Les champs auront pour classe « *my-3* » et un placeholder. Le bouton quant à lui possédera les classes « *btn btn-success mt-3* ».

## B. Test

### Exercice 1 : Quiz

[solution n°5 p.16]

#### Question 1

C'est grâce à l'héritage de la classe `AbstractType` qu'il est possible d'utiliser `BuildForm()`.

- ☐ Vrai
- ☐ Faux

#### Question 2

Faire une redirection après le traitement d'un formulaire est une bonne pratique.

- ☐ Vrai
- ☐ Faux

#### Question 3

Il est préférable d'ajouter les attributs CSS d'un formulaire dans :

- ☐ Le contrôleur
- ☐ Un fichier PHP Type
- ☐ Le fichier Twig

#### Question 4

« *FormBuilderInterface* » supporte de nombreux types de bouton différents.

- ☐ Vrai
- ☐ Faux

#### Question 5


« *createFormBuilder()* » est une méthode que l'on peut utiliser dans une classe qui hérite de `AbstractType`.

- ☐ Vrai
- ☐ Faux

## Solutions des exercices


**Exercice p. 6 Solution n°1****Question 1**

buildform() est :

- ☐ Une méthode de \$builder
- ☐ Une méthode de Symfony
- ☒ Une méthode de FormBuilderInterface
-  buildform() est une méthode de FormBuilderInterface qui va permettre de créer un objet qui est souvent nommé par défaut \$builder.


**Question 2**

La méthode add() de buildForm() permet d'ajouter un formulaire.

- ☐ Vrai
- ☒ Faux
-  La méthode add() de buildForm() permet d'ajouter des champs à un formulaire.


**Question 3**

Comment définir une option pour un champ de formulaire dans Symfony ?

- ☐ En utilisant la méthode addOption()
- ☒ En utilisant un tableau d'options comme deuxième argument de la méthode add()
- ☐ En utilisant la méthode setOption()
-  On peut ajouter de nombreuses options dans un tableau facultatif de la méthode add().

**Question 4**


Dans quel ordre les étapes de formulaire doit-on procéder ?

- ☒ Construire, rendre, traiter
- ☐ Rendre, construire, traiter
- ☐ Traiter, rendre, construire
-  Il faut suivre l'ordre des étapes suivantes : construire un formulaire dans un contrôleur ou un fichier à séparer, rendre le formulaire dans un modèle et le traiter dans le contrôleur.

**Question 5**

Qu'est-ce qu'un Type de formulaire ?


- ☐ Un Type spécial
- ☐ Un Text Type
- ☒ Une classe PHP

-  Un type de formulaire `FormType` est une classe PHP qui pourra être utilisée n'importe où, notamment dans un contrôleur, grâce à l'autoloader de Composer et l'utilisation des namespaces

### Exercice p. 10 Solution n°2


#### Question 1

Quelle façon d'afficher un formulaire dans un fichier Twig ne fonctionnera pas ?

- ☐ `{{ form(form) }}`
- ☒ `{{ form_row(form.title) }}`
- ☐ `{{ form_start(form) }} {{ form_row(form.title) }} {{ form_end(form) }}`
-  Pour utiliser `form_row`, il faut être entre `form_start()` et `form_end()`.


#### Question 2

Les méthodes « `form_row()` » et « `form_widget` » donnent le même résultat.

- ☐ Vrai
- ☒ Faux
-  Faux. Ce sont des alternatives. `Form_widget` ne permettra d'afficher que l'input.


#### Question 3

« `createFormBuilder()` » est une méthode fournie par :

- ☐ Symfony
- ☐ Doctrine
- ☒ `AbstractController`
-  Il est important d'étendre la classe `controller` avec `AbstractController`, car cette classe fournit de nombreuses méthodes, dont `CreateFormBuilder()` qui permet de construire un formulaire.

#### Question 4


La balise `form_rest()` permet d'afficher tous les champs qui n'ont pas été appelés.

- ☒ Vrai
- ☐ Faux
-  Vrai. Effectivement, `form_rest()` peut être utile lorsque le formulaire contient de nombreux champs et que l'on souhaite avoir des précisions que sur certains champs.

#### Question 5

La méthode « `handleRequest()` » joue un rôle à la soumission d'un formulaire.

- ☒ Vrai
- ☐ Faux

 Vrai. La méthode « *handleRequest()* » va gérer le traitement de la saisie du formulaire après soumission de celui-ci par l'utilisateur.

### p. 11 Solution n°3

```

1 <?php
2
3 namespace App\Form;
4
5 use App\Entity\User;
6 use Symfony\Component\Form\AbstractType;
7 use Symfony\Component\Form\Extension\Core\Type\EmailType;
8 use Symfony\Component\Form\Extension\Core\Type\TextType;
9 use Symfony\Component\Form\Extension\Core\Type>PasswordType;
10 use Symfony\Component\Form\FormBuilderInterface;
11 use Symfony\Component\OptionsResolver\OptionsResolver;
12
13 class RegistrationType extends AbstractType
14 {
15     public function buildForm(FormBuilderInterface $builder, array $options): void
16     {
17         $builder
18             ->add('email', EmailType::class, [
19                 'attr' => [
20                     'class' => 'form-control'
21                 ],
22                 'label' => 'E-mail'
23             ])
24             ->add('username', TextType::class, [
25                 'attr' => [
26                     'class' => 'form-control'
27                 ],
28                 'label' => 'Nom'
29             ])
30             ->add('password', PasswordType::class, [
31                 'attr' => [
32                     'class' => 'form-control'
33                 ],
34                 'label' => 'Mot de passe'
35             ])
36             ->add('confirm_password', PasswordType::class, [
37                 'attr' => [
38                     'class' => 'form-control'
39                 ],
40                 'label' => 'Mot de passe'
41             ])
42         ;
43     }
44
45     public function configureOptions(OptionsResolver $resolver): void
46     {
47         $resolver->setDefaults([
48             'data_class' => User::class,
49         ]);
50     }
51 }

```

p. 11 Solution n°4

```

1 {% block body %}
2   <div class="d-flex justify-content-center">
3     <h1 class=" d-flex justify-content-center ">Inscrivez-vous !</h1>
4     <div class=" d-flex justify-content-center ">
5       <div class="d-flex justify-content-center ">
6         {{ form_start(form) }}
7
8         <p>{{ form_row(form.username, {'attr':
9           {'class':'my-3', 'placeholder':'username'}}) }}
10      </p>
11      <p>{{ form_row(form.email, {'attr':
12        {'class':'my-3', 'placeholder':'email'}}) }}</p>
13      <p>{{ form_row(form.password, {'attr':
14        {'class':'my-3', 'placeholder':'mot de passe'}}) }}</p>
15      <p>{{ form_row(form.confirm_password, {'attr':
16        {'class':'my-3', 'placeholder':'mot de
17        passe':' '}}) }}</p>
18      <button type="submit" class="btn btn-success mt-3">Inscription</button>
19      {{ form_end(form) }}
20    </div>
21  </div>
22
23 {% endblock %}

```

Exercice p. 12 Solution n°5

Question 1

C'est grâce à l'héritage de la classe AbstractType qu'il est possible d'utiliser BuildForm().

- ☒ Vrai
- ☐ Faux
- ☒ Vrai. AbstractType fournit entre autres les méthodes buildform() et configureOption().

Question 2

Faire une redirection après le traitement d'un formulaire est une bonne pratique.


- ☒ Vrai
- ☐ Faux
- ☒ Vrai. Cela évite que l'utilisateur actualise la page.



**Question 3**

---


Il est préférable d'ajouter les attributs CSS d'un formulaire dans :

- ☐ Le contrôleur
- ☐ Un fichier PHP Type
- ☒ Le fichier Twig
-  Bien que l'ajout des attributs soit possible dans ces 3 endroits, Symfony préconise de mettre les attributs CSS dans le modèle, ce qui permet une meilleure collaboration entre les développeurs.

**Question 4**

---


« *FormBuilderInterface* » supporte de nombreux types de bouton différents.

- ☒ Vrai
- ☐ Faux
-  Vrai. Nous pourrions citer `CheckboxType` ou encore `RadioType`.

**Question 5**

---

« *createFormBuilder()* » est une méthode que l'on peut utiliser dans une classe qui hérite de `AbstractType`.

- ☐ Vrai
- ☒ Faux
-  Faux. Cette méthode est issue d'`Abstract Controller` et donc ne peut s'utiliser que dans le contrôleur.