

**Niveau :** STS SIO 2<sup>ème</sup> année☐ Cours ☐ TD ☒ TP

1/35

**Module :** SLAM5 – Solutions applicatives**Intitulé :** Compléments sur Symfony 2**Durée :** 3 heures

**Objectifs :**

- ✓ Validation des données sous Symfony
- ✓ Utilisation d'un Framework CSS dans Symfony
- ✓ Programmer au sein d'un Framework

## Contenu

<b>Objectif du TP</b>	<b>2</b>
<b>Améliorons la navigation de notre site</b>	<b>2</b>
Depuis toutes les pages	2
Dans la page d'accueil (liste des abonnés)	4
Dans la page de consultation d'un abonné	5
Dans la page d'ajout d'un abonné	6
Dans la page de modification d'un abonné	6
Dans la page de suppression d'un abonné	6
<b>Validation des données</b>	<b>7</b>
Méthode 1 : Utiliser les options des champs de formulaire	7
Méthode 2 : Procéder comme d'habitude	8
Méthode 3 : Utiliser les annotations dans l'entité	9
Méthode 4 : Utiliser des méthodes isXxxValid()	10
Méthode 5 : Utiliser les callbacks	11
A vous	12
<b>Les « paramConverters »</b>	<b>13</b>
<b>Un peu de CSS</b>	<b>16</b>
Inclure Bootstrap de Twitter	17
Modifier le layout de base	17
Modifier les templates de vues	20
Gestion des formulaires	25

## Objectif du TP

Dans ce TP, vous allez améliorer un peu la navigation de votre application AppliMVC2.

Puis nous aborderons les contrôles de validation de nos formulaires.

Enfin, nous verrons comment embellir nos pages en intégrant Twitter Bootstrap.

## Améliorons la navigation de notre site

### Depuis toutes les pages

On veut ajouter un lien vers la page permettant l'ajout d'un nouvel abonné (URL `/inscriptions/ajouter`)

Un seul fichier à modifier. Devinez lequel ?

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>{% block title %}TP MVC2{% endblock %}</title>
    {% block stylesheets %}
    <link rel="stylesheet" href="{{ asset('css/styles.css') }}"
      type="text/css" />
    {% endblock %}
  </head>
  <body>
    <h1>Gestion des inscriptions</h1>
    <a href="{{ path('sioinscriptions_ajouter') }}">
      Ajouter un abonné
    </a>
    {% block body %}{% endblock %}
    {% block javascripts %}
    <script
src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.
js"></script>
    {% endblock %}
  </body>
</html>
```

**A noter** : l'instruction Twig « **path** » permet de récupérer l'url correspondant à une route.

Pratique, n'est-ce pas !!!

Nous obtenons :

← → ↻ 🏠 localhost/votreAlias/AppliMVC2/web/app\_dev.p

## Gestion des inscriptions

[Ajouter un abonné](#)

### Liste des abonnés

Nom	Prénom	Date naissance	Email
DUPONT	Pierre	05/12/1984	dupont.pierre60@sfr.fr
MAXWELL	Stan	15/03/1978	stanmax@laposte.net
BONNOT	Jean	08/08/1972	jeanbonnot@gmail.com
DESHAIES	Stan	15/03/1978	stanmax@laposte.net

← → ↻ 🏠 localhost/votreAlias/AppliMVC2/web/app\_dev.p

## Gestion des inscriptions

[Ajouter un abonné](#)

### Ajouter un abonné

Nom

Prénom

Date de Naissance  
1 ▼ janv. ▼ 1902 ▼

Adresse mail

← → ↻ 🏠 localhost/votreAlias/AppliMVC2/web/app\_

## Gestion des inscriptions

[Ajouter un abonné](#)

### Détail abonné n° 1

Nom : DUPONT  
Prénom : Pierre  
Date de naissance : 05/12/1984  
Email : pdupont@yahoo.fr

← → ↻ 🏠 localhost/votreAlias/AppliMVC2/web/app\_de

## Gestion des inscriptions

[Ajouter un abonné](#)

### Modification de l'abonné n° 3

Nom

Prénom

Date de Naissance  
8 ▼ août ▼ 1972 ▼

Adresse mail

← → ↻ 🏠 localhost/votreAlias/AppliMVC2/web/app\_dev.php/inscrip

## Gestion des inscriptions

[Ajouter un abonné](#)

### Supprimer un abonné

Etes-vous certain de vouloir supprimer l'abonné "BONNOT Jean" ?

## Dans la page d'accueil (liste des abonnés)


4/35

On veut rendre le nom de l'abonné cliquable et qui nous renvoie vers la page de consultation de cet abonné (URL /inscriptions/inscription/{id}).

Il suffit de modifier la vue correspondante ainsi :

```
<tbody>
{% for abonne in lesAbonnes %}
  <tr>
    <td>
      <a href="{{ path('sioinscriptions_voir', {'id': abonne.id}) }}">
        {{ abonne.nom }}
      </a>
    </td>
    <td>{{ abonne.prenom }}</td>
    <td>{{ abonne.dateNaissance|date('d/m/Y') }}</td>
  </tr>
{% else %}
  <tr><td colspan="3">Aucun abonné !</td></tr>
{% endfor %}
</tbody>
```

On obtient alors :



localhost/votreAlias/AppliMVC2/web/app\_dev.php/inscriptions/

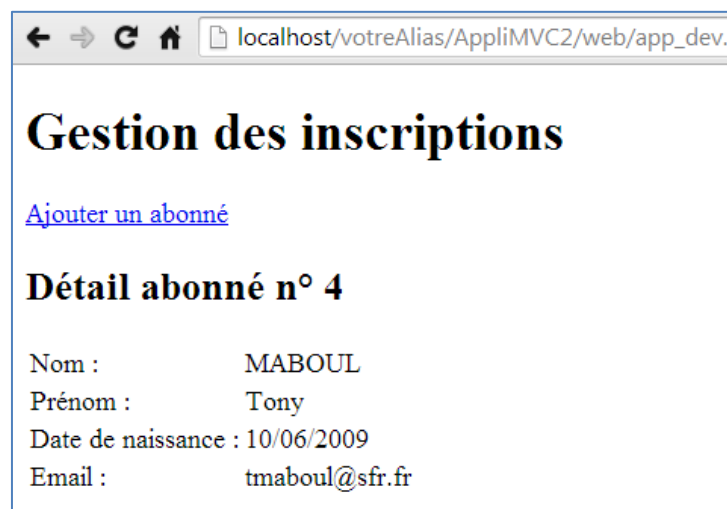
## Gestion des inscriptions

[Ajouter un abonné](#)

### Liste des abonnés

Nom	Prénom	Date naissance
<a href="#">DUPONT</a>	Pierre	05/12/1984
<a href="#">MAXWELL</a>	Stan	15/03/1978
<a href="#">BONNOT</a>	Jean	08/08/1972
<a href="#">MABOUL</a>	Tony	10/06/2009

puis en cliquant sur MABOUL,



localhost/votreAlias/AppliMVC2/web/app\_dev.

## Gestion des inscriptions

[Ajouter un abonné](#)

### Détail abonné n° 4

Nom : MABOUL  
 Prénom : Tony  
 Date de naissance : 10/06/2009  
 Email : tmaboul@sfr.fr

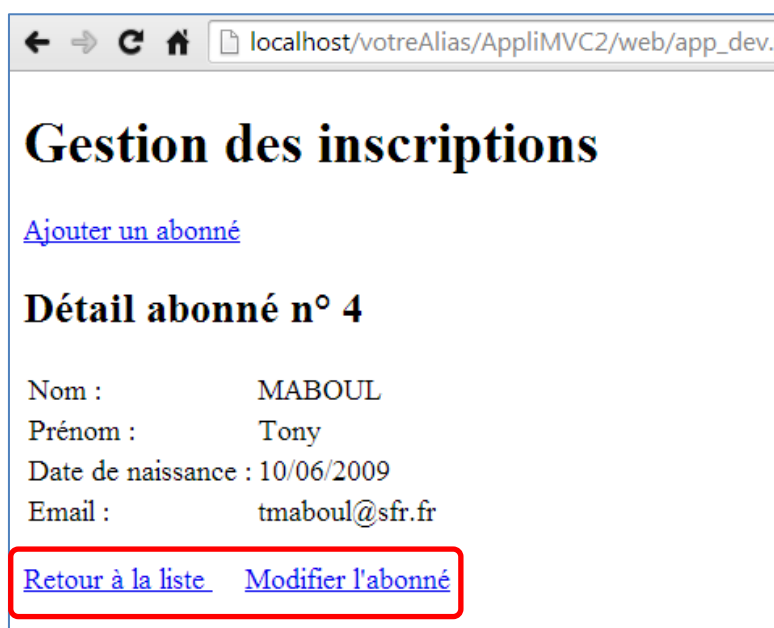
## Dans la page de consultation d'un abonné

On veut un lien permettant un retour vers la liste des abonnés ainsi qu'un lien permettant de modifier l'abonné.

Il suffit de modifier la vue correspondante ainsi :

[illegible]

Nous obtenons :



**A vous de poursuivre pour la suite ... voir les consignes page suivante.**

## Dans la page d'ajout d'un abonné

On veut un lien permettant un retour vers la liste des abonnés.

6/35

### Gestion des inscriptions

[Ajouter un abonné](#)

#### Ajouter un abonné

Nom

Prénom

Date de Naissance

Adresse mail

[Retour à la liste](#)

## Dans la page de modification d'un abonné

On veut un lien permettant un retour vers la liste des abonnés, un lien permettant de modifier l'abonné ainsi qu'un lien permettant de supprimer l'abonné.

### Gestion des inscriptions

[Ajouter un abonné](#)

#### Modification de l'abonné n° 3

Nom

Prénom

Date de Naissance

Adresse mail

[Retour à la liste](#) [Retour à l'abonné](#)

## Dans la page de suppression d'un abonné

On veut un lien permettant un retour vers l'abonné.

### Gestion des inscriptions

[Ajouter un abonné](#)

#### Supprimer un abonné

Etes-vous certain de vouloir supprimer l'abonné "BONNOT Jean" ?

[Retour à l'abonné](#)

## Validation des données

7/35

Symfony effectue un certain nombre de contrôles dans vos formulaires.  
En fait, de base, tous les champs sont obligatoires.  
Ensuite, selon le type de champ, d'autres contrôles sont effectués : nombre, date, ...

Toutefois, des ajustements sont le plus souvent nécessaires : prendre en compte des règles « métier », autoriser des champs vides, ...

Il existe différents moyens de définir les contraintes de validité des formulaires.

### Méthode 1 : Utiliser les options des champs de formulaire

Chaque type de champ (`text`, `textarea`, `email`, `integer`, `choice`, `date`, ...) possède un certain nombre d'options dont certaines peuvent être utiles pour la validation des données.

#### Exemples :

Reprenons nos formulaires d'ajout et de modification. En l'état, il est possible d'entrer n'importe quoi dans l'adresse mail.

Ceci est normal puisque nous l'avons déclaré comme champ de type « text ».

Modifiez les méthodes de votre contrôleur pour déclarer ce champ comme champ mail :

```
$formBuilder = $this->createFormBuilder($abonne)
    ->add('nom', 'text', array('label' => 'Nom'))
    ->add('prenom', 'text', array('label' => 'Prénom'))
    ->add('dateNaissance', 'birthday', array('label' => 'Date de Naissance'))
    ->add('email', 'email', array('label' => 'Adresse mail'))
    ->add('Valider', submit);
```

Essayez maintenant de saisir n'importe quoi dans l'adresse mail de vos formulaires, vous obtenez un message d'erreur bloquant :

Supposons maintenant que l'on souhaite rendre le prénom non obligatoire :

Le type de champ « text » possède une option « required ».

Par défaut, cette option vaut true. Pour rendre le champ facultatif, il suffit donc d'affecter la valeur false à cette option, comme ci-dessous :

```
$formBuilder = $this->createFormBuilder($abonne)
->add('nom', 'text', array('label' => 'Nom'))
->add('prenom', 'text', array('label' => 'Prénom', 'required' => false))
->add('dateNaissance', 'birthday', array('label' => 'Date de Naissance'))
->add('email', 'email', array('label' => 'Adresse mail'))
->add('Valider', submit);
```

Il existe d'autres options qui peuvent servir à la validation des champs (rounding\_mode sur les champs de type integer, empty\_value sur les listes déroulantes, ...).

Je ne vais pas vous présenter tous les types de champs et leurs options.

La documentation de Symfony2 est très bien faite. N'hésitez donc pas à la consulter chaque fois que vous créez un formulaire. La page de référence, c'est là :

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

## Méthode 2 : Procéder comme d'habitude

Vous pouvez bien entendu continuer à faire comme vous le faisiez jusque-là, avant de connaître Symfony : ajoutez les contrôles dans le contrôleur

- **Dans l'action du contrôleur**

```
public function ajouterAction()
{
    $lesErreurs = array();
    if ($request->getMethod() == 'POST') {
        if (...) {
            $lesErreurs[] = "mesg erreur 1";
        }
        if (...) {
            $lesErreurs[] = "mesg erreur 2";
        }
    }

    [...]
    return $this->render('sioFilmsBundle:Films:ajouter.html.twig', array(
        'form' => $form->createView(),
        'lesErreurs' => $lesErreurs
    ));
}
```

- **Dans la vue**

```
<div class ="erreur">
<ul>
    {% for erreur in lesErreurs %}

        <li>{{erreur}}</li>

    {%endfor%}
</ul></div>
```



## Méthode 3 : Utiliser les annotations dans l'entité

9/35

```
[...]
use Symfony\Component\Validator\Constraints as Assert;

/**
 * Film
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="sio\FilmsBundle\Entity\FilmRepository")
 */
class Film
{
    [...]
    /**
     * @var string
     *
     * @ORM\Column(name="titre", type="string", length=100)
     * @Assert\Length(
     *     min = "10",
     *     max = "100",
     *     minMessage = "Le titre doit faire au moins {{ limit }} caract.",
     *     maxMessage = "Le titre ne doit pas dépasser {{ limit }} caract."
     * )
     */
    private $titre;

    /**
     * @var string
     *
     * @ORM\Column(name="realisateur", type="string", length=100)
     * @Assert\Regex(
     *     pattern="/\d/",
     *     match=false,
     *     message="Realisateur ne peut pas contenir de chiffres"
     * )
     */
    private $realisateur;

    /**
     * @var integer
     *
     * @ORM\Column(name="annee", type="smallint")
     * @Assert\Range(
     *     min = 1930,
     *     max = 2040,
     *     minMessage="L'année ne peut pas être antérieure à 1930",
     *     maxMessage="L'année ne peut pas être postérieure à 2040"
     * )
     */
    private $annee;

    /**
     * @var string
     *
     * @ORM\Column(name="synopsis", type="text")
     * @Assert\NotBlank()
     */
    private $synopsis;

    [...]
```

Pour plus d'infos sur les contraintes

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

L'instruction `if ($form->isValid())` dans le contrôleur fera appel au service **Validator** pour valider l'objet hydraté par le formulaire : les erreurs sont affectées au formulaire puis affichées dans la vue. 10/35

Dans ce cas, le message généré apparaît au-dessus du formulaire si on utilise le widget `form_widget`.

## Méthode 4 : Utiliser des méthodes `isXxxValid()`

Il est également possible de définir une validation globale au niveau de l'entité ou au niveau d'un attribut en utilisant une méthode `isXxxValid()` où Xxx correspond au nom de l'entité ou au nom de l'attribut.

Dans ce cas, le message généré apparaît au-dessus du formulaire si on utilise le widget `form_widget`.

Pour plus d'informations : <http://symfony.com/fr/doc/current/book/validation.html>

- **Validation au niveau de l'entité**

Dans ce cas on crée une méthode particulière dont le nom a la forme **`isEntiteValid()`** :

```
[...]
use Symfony\Component\Validator\Constraints as Assert;

/**
 * Film
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="sio\FilmsBundle\Entity\FilmRepository")
 */
class Film
{
    [...]
    /**
     * @Assert\True(message="Il faut renseigner l'emprunteur")
     */
    * @return
    */
    public function isFilmValid()
    {
        if ($this->pret == true and $this->emprunteur == null) {
            return false;
        }
    }
}
```

- **Validation au niveau d'un attribut**

Dans ce cas on crée une méthode particulière dont le nom a la forme `isAttributValid()`

```
[...]
use Symfony\Component\Validator\Constraints as Assert;

/**
 * Film
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="sio\FilmsBundle\Entity\FilmRepository")
 */
class Film
{
    [...]
    /**
     * @Assert\True(message="Il faut renseigner l'emprunteur")
     */
    * @return
    */
    public function isPretValid() {
        if ($this->pret == true and $this->emprunteur == null) {
            return false;
        }
    }
}
```

## Méthode 5 : Utiliser les callbacks

L'intérêt de la contrainte callback est qu'elle est personnalisable à souhait et permet de réaliser tous les contrôles dont on a besoin.

```
[...]
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Component\Validator\ExecutionContextInterface;

/**
 * Film
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="sio\FilmsBundle\Entity\FilmRepository")
 */
* @Assert\Callback(methods={"filmValide"})
*/
class Film
{
    [...]
    public function filmValide(ExecutionContextInterface $context) {
        if ($this->pret == true and $this->emprunteur == null) {
            $context->addViolationAt('emprunteur', 'Vous devez sélectionner
            une personne', array(), null);
        }
    }
}
```

Le champ spécifié en 1<sup>er</sup> paramètre de la méthode `addViolation()` est le champ sur lequel l'erreur apparaîtra dans le formulaire.

Pour plus d'informations :

<http://symfony.com/fr/doc/current/reference/constraints/Callback.html>

## A vous...

12/35

Allez, juste pour découvrir un peu tout cela !

- Vous utiliserez tout d'abord la **méthode des annotations** pour vérifier que le nom et le prénom doivent comporter entre 5 et 30 caractères.  
Testez et lorsque ça fonctionne, supprimez vos tests.

- Vous utiliserez ensuite des **méthodes isXXXValid()** pour vérifier que :

- Le nom comporte entre 5 et 30 caractères
- Le prénom comporte entre 5 et 30 caractères
- L'année de naissance est postérieure à 1940.

Note : vous devez utiliser la classe PHP DateTime.

Exemple de création d'une date : `$dateRef = new \DateTime('1940-12-31');`

Ensuite, il suffit de comparer avec l'opérateur habituel (< ou > ou ...)

**Remarque** : l'antislash (\) devant DateTime signifie que l'on ne se réfère pas à un espace de nom. Si vous ne le mettez pas, vous obtenez une erreur.

Testez et lorsque ça fonctionne, mettez vos méthodes en commentaires.

- Enfin, vous referez ces 3 tests en utilisant un **callback**.  
Testez.

Vous pouvez constater que le rendu n'est pas le même selon l'endroit où fait les tests.  
Après, chacun utilise la méthode qu'il préfère

## Les « paramConverters »

13/35

Les **ParamConverters** ou « **convertisseurs de paramètres** » permettent de convertir des paramètres de requêtes en objets. Ces objets sont stockés comme attributs de requête de telle sorte qu'ils puissent être injectés comme arguments de méthodes de contrôleur.

Ils permettent de faire gagner du temps et des lignes de code.

Symfony possède deux convertisseurs préconstruits : celui de **Doctrine**, et un convertisseur **DateTime**.

Le convertisseur de Doctrine fait partie du bundle **Sensio\FrameworkBundle**. C'est un bundle **activé par défaut** avec la distribution standard de Symfony2.

Ce convertisseur tente de convertir des attributs de requête en entités Doctrine récupérées depuis la base de données. Deux approches sont possibles :

- Récupérer l'objet par sa clé primaire ;
- Récupérer l'objet par un ou plusieurs champ(s) qui contien(nen)t une valeur unique en base de données.

Je ne vais pas vous faire un cours sur les ParamConverters. Pour ceux qui veulent en découvrir plus, rendez-vous sur la page de Symfony dédiée aux convertisseurs :

<http://symfony.com/fr/doc/current/bundles/SensioFrameworkExtraBundle/annotations/converters.html#convertisseurs-preconstruits>

Nous allons utiliser le fonctionnement de base du convertisseur de Doctrine : celui permettant de faire correspondre un paramètre {id} d'une route avec une instance d'une entité Doctrine (une classe, quoi) !

Dans ce mode de base, il n'y a rien (enfin, presque rien) à faire. Par contre, il va vous faire gagner du temps et des lignes de code !!!

Imaginez une **route** définie ainsi :

```
siofilms_voir:
  path:      /films/film/{id}
  defaults: { _controller: sioFilmsBundle:Films:voir }
```

Nous disposons de l'**entité** suivante (sans les annotations pour simplifier) :

```
<?php
namespace sio\FilmsBundle\Entity;
class Film
{
    private $id;
    private $titre;
    private $realisateur;

    public function getId()
    {
        return $this->id;
    }
    public function setTitre($titre)
    {
        $this->titre = $titre;

        return $this;
    }
    public function getTitre()
    {
        return $this->titre;
    }
    public function setRealisateur($realisateur)
    {
        $this->realisateur = $realisateur;

        return $this;
    }
    public function getRealisateur()
    {
        return $this->realisateur;
    }
}
```

Dans le **contrôleur**, nous avons la méthode ci-dessous qui recherche le film dont l'id est passé en paramètre (\$id). Si le film n'est pas trouvé, une exception est générée. Sinon, une page sera affichée avec les informations du film.

```
public function voirAction($id)
{
    // On récupère le repository
    $repository = $this->getDoctrine()
        ->getManager()
        ->getRepository('sioFilmsBundle:Film');

    // On récupère l'entité correspondant à l'id $id
    $film = $repository->find($id);

    if ($film == null) // le film n'a pas été trouvé
    {
        throw $this->createNotFoundException("Film[id='$id'] inexistant.");
    }
    return $this->render('sioFilmsBundle:Films:voir.html.twig',
        array('film' => $film));
}
```

En utilisant le convertisseur de base de Doctrine, voici ce que devient la méthode :

15/35

```
public function voirAction(Film $film)
{
    return $this->render('sioFilmsBundle:Films:voir.html.twig',
        array('film' => $film));
}
```

Le convertisseur a tenté de convertir l'id reçu en paramètre de la requête en un objet Film.

S'il n'y parvient, il lève une exception. S'il y parvient, la méthode du contrôleur récupère directement l'objet Film correspondant à l'id passé sur l'url.

Fabuleux, non ?

**Allez, à vous : modifiez toutes les méthodes de votre contrôleur qui reçoivent un identifiant d'abonné en paramètre.**

N'oubliez pas de faire référence au namespace du bundle FrameworkBundle :

```
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
```

## Un peu de CSS

16/35

Bon, je vous avoue que je n'ai pas très envie de me pencher sur les CSS.

Nous allons donc utiliser Bootstrap de Twitter pour aller vite et faire quelque chose de propre.

Voici la structure de nos pages :

Un entête commun à toutes les pages

Une zone navigation commune à toutes les pages


Un panel pour afficher le titre des pages

Le contenu spécifique à chaque page

Un pied de page fixé au bas de la fenêtre

### Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015



[Accueil](#)
[Ajouter un abonné](#)
[Voir le tutoriel](#)

#### Liste des abonnés

Nom	Prénom	Date naissance
DUPONT	Pierre	05/12/1984
MAXWELL	Stan	15/03/1978
BONNOT	Jean	08/08/1972
MABOUL	Tony	10/06/2009


Les SIO SLAM promo 2015 du lycée Jean Rostand de Chantilly ©2014

Une zone de navigation spécifique

### Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015



[Accueil](#)
[Ajouter un abonné](#)
[Voir le tutoriel](#)

#### Détail abonné n° 1

Nom :
DUPONT

Prénom :
Pierre

Date de naissance :
05/12/1984

Email :
pdupont@yahoo.fr

[Retour à la liste](#)
[Modifier l'abonné](#)
[Supprimer l'abonné](#)

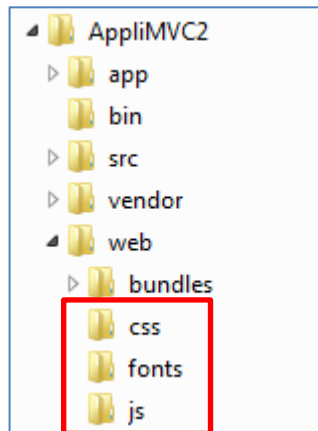
Les SIO SLAM promo 2015 du lycée Jean Rostand de Chantilly ©2014



## Inclure Bootstrap de Twitter

17/35

Vous allez commencer par décompresser l'archive bootstrap dans le dossier web de votre application pour obtenir :



Ajouter également un dossier `img` dans le répertoire web et y placer le fichier `logo.jpg`.

## Modifier le layout de base

Rappelez-vous : toutes nos vues héritent du layout de base :

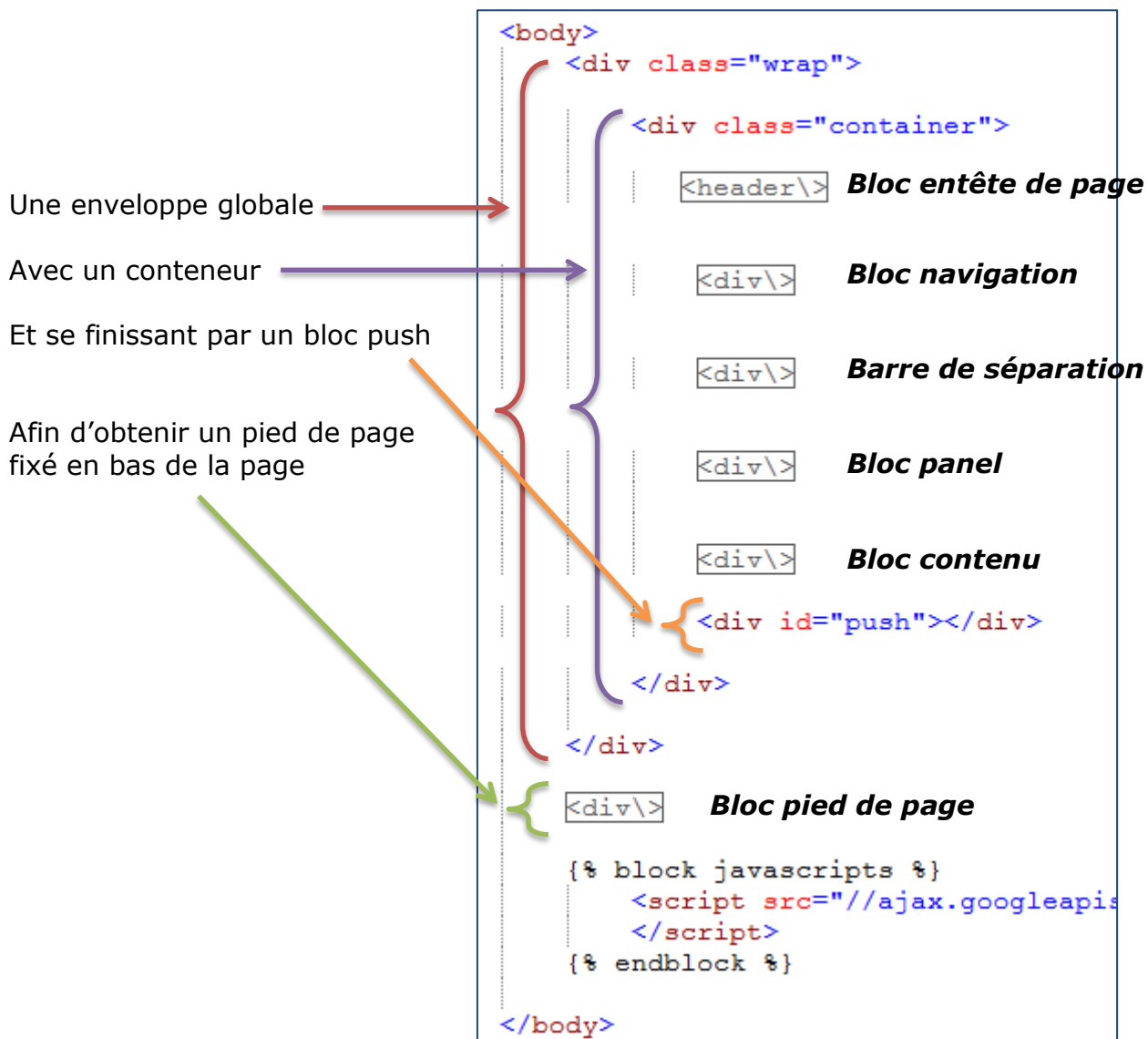
`/app/Resources/views/layout.html.twig`

Il faut donc commencer à s'attaquer à ce layout.

- Bloc `<head>` : on appelle le fichier css de bootstrap.  
On applique également un padding inférieur de 70px sur le corps de la page (pour éviter que le contenu de la page ne chevauche le pied de page)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>{% block title %}TP MVC2{% endblock %}</title>
    {% block stylesheets %}
      <link rel="stylesheet" href="{{ asset('css/bootstrap.css') }}"
        type="text/css" />
    {% endblock %}
    <style type="text/css">
      body { padding-bottom:70px; }
    </style>
  </head>
```

Voici la structure générale du corps de la page : nous compléterons bloc par bloc.



### ➤ Le bloc entête

```

<header class="page-header">
  <div class="row">
    <div class="col-sm-8">
      <div class="row">
        <h1>Gestion des abonnés</h1>
      </div>
      <div class="row">
        <h4>Ce projet est développé par les étudiants SIO SLAM,
        promo 2015</h4>
      </div>
    </div>
    <div class="col-sm-4">
      
    </div>
  </div>
</header>

```

## ➤ Le bloc navigation

```

<div class="row">
  <div class="col-sm-12">
    <ul class="nav nav-pills">
      <li class="active">
        <a href="{{ path('sioinscriptions_accueil') }}">
          Accueil
        </a>
      </li>
      <li>
        <a href="{{ path('sioinscriptions_ajouter') }}">
          Ajouter un abonné
        </a>
      </li>
      <li>
        <a href="http://symfony.com/doc/current/index.html">
          Voir le tutoriel
        </a>
      </li>
    </ul>
  </div>
</div>

```

## ➤ La barre de séparation

```

<div class="row">
  <div class="col-sm-12">
    <hr />
  </div>
</div>

```

## ➤ Le bloc panel

```

<div class="row">
  <div class="col-sm-12">
    <div class="panel panel-info">
      <div class="panel-heading">
        {% block paneltitle %}
        {% endblock %}
      </div>
    </div>
  </div>
</div>

```

## ➤ Le bloc contenu

```

<div class="row">
  <div class="col-sm-12">
    {% block body %}
    {% endblock %}
  </div>
</div>

```

## ➤ Le bloc pied de page

```
<div class="navbar navbar-fixed-bottom">
  <div class="container">
    <hr />
    <div id="footer">
      <p>Les SIO SLAM promo 2015 du lycée Jean Rostand
        de Chantilly ©2014
      </p>
    </div>
  </div>
</div>
```

## Modifier les templates de vues

## ➤ Le template index.html.twig

```
{% extends "::layout.html.twig" %}

{% block title %}
    Gestion abonnés - {{ parent() }}
{% endblock %}

{% block paneltitle %}
    <h3>Liste des abonnés</h3>
{% endblock %}

{% block body %}
    <p>
        {# On affiche tous les messages flash dont le nom est
        {% for message in app.session.flashbag.get('info') %}
            <p>{{ message }}</p>
        {% endfor %}
    </p>

    <table\>

{% endblock %}

{% block javascripts %}
{% endblock %}
```

Bloc stylesheets supprimé (il est commun à toutes les pages et défini dans le layout de base.

Ajout du bloc title

Ajout du bloc panel qui contient le titre de la page

Et sur la balise table, ajout des classes table et table-striped

```
<table class="table table-striped">
    <thead\>
    <tbody\>
</table>
```



## ➤ Le template modifier.html.twig

```
{% extends "::layout.html.twig" %}

{% block title %}
Modification abonné - {{ parent() }}
{% endblock %}

{% block paneltitle %}
<h3>Modification de l'abonné n° {{ abonne.id }}</h3>
{% endblock %}

{% block body %}
    {{ form_start(form) }}
        {{ form_widget(form) }}
    {{ form_end(form) }}

<p>
    <a href="{{ path('sioinscriptions_accueil') }}">
        <span class="glyphicon glyphicon-chevron-left"></span>
        Retour à la liste
    </a>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
    <a href="{{ path('sioinscriptions_voir', {'id': abonne.id}) }}">
        <span class="glyphicon glyphicon-eye-open"></span>
        Retour à l'abonné
    </a>
</p>
{% endblock %}

{% block javascripts %}
{% endblock %}
```

## ➤ Le template voir.html.twig

Pour le début, les modifications sont identiques :

```
{% extends "::layout.html.twig" %}

{% block title %}
Détail abonné - {{ parent() }}
{% endblock %}

{% block paneltitle %}
    <h3>Détail abonné n° {{ abonne.id }}</h3>
{% endblock %}

{% block body %}

<p>
    {# On affiche tous les messages flash dont le nom est « info » #}
    {% for message in app.session.flashbag.get('info') %}
        <p>{{ message }}</p>
    {% endfor %}
</p>
```

On adapte un peu le contenu pour que le tableau n'occupe pas toute la largeur du conteneur.

Rappel : Bootstrap travaille sur des grilles de 12 colonnes.

Chaque conteneur contient donc 12 colonnes.

Un sous-conteneur contient aussi 12 colonnes, etc.

En fait, la taille d'une colonne n'est pas fixe, elle dépend de la largeur de son conteneur.

On définit donc un conteneur de type « row » qui comporte donc 12 colonnes.

La table est placée dans l'espace occupé par les 6 premières colonnes.

On ajoute également les classes CSS sur la table.

```
<div class="row">
  <div class="col-sm-6">
    <table class="table table-striped">
      <tr></tr>
      <tr></tr>
      <tr></tr>
      <tr></tr>
    </table>
  </div>
  <div class="col-sm-6"></div>
</div>
```

La fin du template est inchangée hormis l'ajout des glyphicons :

[illegible]


Vous pouvez maintenant tester.

Votre application doit commencer à prendre forme.

24/35

## Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015




[Accueil](#) [Ajouter un abonné](#) [Voir le tutoriel](#)

### Liste des abonnés

Nom	Prénom	Date
DUPONT	Pierre	05/12
MAXWELL	Stan	15/03
BONNOT	Jean	08/08
MABOUL	Tony	10/06
DESHAIES	Raoul	01/01

## Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015



[Accueil](#) [Ajouter un abonné](#) [Voir le tutoriel](#)

### Détail abonné n° 1


Nom :	DUPONT
Prénom :	Pierre
Date de naissance :	05/12/1984
Email :	dupont.pierre60@sfr.fr

[Retour à la liste](#)
[Modifier l'abonné](#)
[Supprimer l'abonné](#)

En fait, il n'y a plus que nos formulaires qui ne sont pas « à la sauce bootstrap ».

## Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015



[Accueil](#) [Ajouter un abonné](#) [Voir le tutoriel](#)

### Ajouter un abonné

Nom   
 Prénom   
 Date de Naissance  
     
 Adresse mail   
  
[Retour à la liste](#)

Nous allons retravailler sur le design de nos formulaires.



## Gestion des formulaires

25/35

J'ai choisi le formulaire horizontal.  
Examinons un peu la page dédiée du site de Bootstrap.

getbootstrap.com/css/#forms

### Horizontal form

Use Bootstrap's predefined grid classes to align labels and groups of form controls in a horizontal layout by adding `.form-horizontal` to the form. Doing so changes `.form-group`s to behave as grid rows, so no need for `.row`.

EXAMPLE Copy

Email

Password

☐ Remember me

```

<form class="form-horizontal" role="form">
  <div class="form-group">
    <label for="inputEmail3" class="col-sm-2 control-label">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="inputEmail3" placeholder="Email">
    </div>
  </div>
  <div class="form-group">
    <label for="inputPassword3" class="col-sm-2 control-label">Password</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="inputPassword3"
placeholder="Password">
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <div class="checkbox">
        <label>
          <input type="checkbox"> Remember me
        </label>
      </div>
    </div>
  </div>
</form>

```

Chaque champ est encadré par une balise `<div>` avec la classe **form-group**.

Les labels et les champs ont une largeur exprimée en nombre de colonnes.

Le formulaire est associé à la classe **form-horizontal**.

Reprenons un peu le template utilisé pour l'ajout d'un abonné, et plus particulièrement la partie concernant l'affichage du formulaire.

```
{% block body %}
    {{ form_start(form) }}
    {{ form_widget(form) }}
    {{ form_end(form) }}
```

La fonction `form_widget(form)` permet d'afficher tous les champs du formulaire en une seule instruction.

Comment faire pour combiner tout ça ?

En fait, deux possibilités s'offrent à nous :

➤ **Afficher le formulaire champ par champ :**

Le code ci-dessus devient alors :

```
{% block body %}
    {{ form_start(form) }}
    {{ form_errors(form) }} ← Affichage des erreurs générales
    <div>
        {{ form_label(form.nom) }} ← Affichage du label du champ nom
        {{ form_errors(form.nom) }} ← Affichage des erreurs sur champ nom
        {{ form_widget(form.nom) }} ← Affichage champ nom
    </div>
    <div>
        {{ form_label(form.prenom) }}
        {{ form_errors(form.prenom) }}
        {{ form_widget(form.prenom) }}
    </div>
    <div>
        {{ form_label(form.dateNaissance) }}
        {{ form_errors(form.dateNaissance) }}
        {{ form_widget(form.dateNaissance) }}
    </div>
    <div>
        {{ form_label(form.email) }}
        {{ form_errors(form.email) }}
        {{ form_widget(form.email) }}
    </div>
    <input type="submit" />
    {{ form_end(form) }}
```

Un peu plus long à écrire, je vous l'accorde, mais cela laisse la possibilité de personnaliser à souhait et donc pour nous d'ajouter nos classes CSS comme ci-après :

### Ajoute un attribut class à la balise <form>

```
{% block body %}
    {{ form_start(form, { 'attr': {'class': 'form-horizontal'} }) }}

    <span class="alert-danger">
        {{ form_errors(form) }}
    </span>

    <span class="alert-danger">{{ form_errors(form.nom) }}</span>
    <div class="form-group">
        <div class="col-sm-2 control-label">
            {{ form_label(form.nom) }}
        </div>
        <div class="col-sm-3">
            {{ form_widget(form.nom, { 'attr': {'class': 'form-control'} }) }}
        </div>
    </div>

    <span class="alert-danger">{{ form_errors(form.prenom) }}</span>
    <div class="form-group">
        <div class="col-sm-2 control-label">
            {{ form_label(form.prenom) }}
        </div>
        <div class="col-sm-3">
            {{ form_widget(form.prenom, { 'attr': {'class': 'form-control'} }) }}
        </div>
    </div>

    <span class="alert-danger">{{ form_errors(form.dateNaissance) }}</span>
    <div class="form-group">
        <div class="col-sm-2 control-label">
            {{ form_label(form.dateNaissance) }}
        </div>
        <div class="col-sm-2">
            {{ form_widget(form.dateNaissance, { 'attr': {'class': 'form-control'} }) }}
        </div>
    </div>

    <span class="alert-danger">{{ form_errors(form.email) }}</span>
    <div class="form-group">
        <div class="col-sm-2 control-label">
            {{ form_label(form.email) }}
        </div>
        <div class="col-sm-4">
            {{ form_widget(form.email, { 'attr': {'class': 'form-control'} }) }}
        </div>
    </div>

    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-4">
            <input type="submit" class="btn btn-primary"/>
        </div>
    </div>

    {{ form_end(form) }}
```

Pour obtenir les messages d'erreur en couleur


Ajoute un attribut class au champ nom

Et voici le résultat :

28/35

## Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015



[Accueil](#) [Ajouter un abonné](#) [Voir le tutoriel](#)

### Ajouter un abonné

Nom

Prénom

Date de Naissance

Adresse mail

[← Retour à la liste](#)

Les SIO SLAM promo 2015 du lycée Jean Rostand de Chantilly ©2014

Un peu plus sympa, n'est-ce pas ?

Il ne reste plus qu'à reproduire la même chose sur le formulaire de modification ...

à moins que ...

## ➤ Habillage de formulaire :

Symfony utilise des templates pour afficher chaque partie d'un formulaire, comme les balises label, les balises input, les messages d'erreur et tout le reste.

Dans Twig, chaque « fragment » de formulaire est représenté par un bloc Twig.

Le fichier `form_div_layout.html.twig` se trouvant dans le répertoire `\vendor\symfony\symfony\src\Symfony\Bridge\Twig\Resources\views\Form\` définit les différents ces différents fragments.

Voici un extrait de ce fichier :

```
{# Widgets #}

{% block form_widget -%}
    {% if compound %}
        {{- block('form_widget_compound') -}}
    {% else %}
        {{- block('form_widget_simple') -}}
    {% endif %}
{%- endblock form_widget %}

{% block form_widget_simple -%}
    {% set type = type|default('text') -%}
    <input type="{{ type }}" {{ block('widget_attributes') }}
        {% if value is not empty %}value="{{ value }}" {% endif %}/>
{%- endblock form_widget_simple %}

{% block form_widget_compound -%}
    <div {{ block('widget_container_attributes') }}>
        {%- if form.parent is empty -%}
            {{ form_errors(form) }}
        {%- endif -%}
        {{- block('form_rows') -}}
        {{- form_rest(form) -}}
    </div>
{%- endblock form_widget_compound %}

[...]
```

```
{% block textarea_widget -%}
    <textarea {{ block('widget_attributes') }}>{{ value }}</textarea>
{%- endblock textarea_widget %}

{% block choice_widget -%}
    {% if expanded %}
        {{- block('choice_widget_expanded') -}}
    {% else %}
        {{- block('choice_widget_collapsed') -}}
    {% endif %}
{%- endblock choice_widget %}

{% block choice_widget_expanded -%}
    <div {{ block('widget_container_attributes') }}>
        {%- for child in form %}
            {{- form_widget(child) -}}
            {{- form_label(child) -}}
        {% endfor -%}
    </div>
{%- endblock choice_widget_expanded %}

[...]
```

Pour personnaliser n'importe quelle partie d'un formulaire, vous avez juste besoin de réécrire le bloc approprié

Pour comprendre comment tout cela fonctionne, voyons comment personnaliser le fragment `form_row` qui constitue une ligne de formulaire, c'est-à-dire un champ accompagné de son label et de son bloc d'affichage d'erreurs

Voici comment est défini ce fragment dans le fichier template de base `form_div_layout.html.twig`

```
{% block form_row -%}
    <div>
        {{- form_label(form) -}}
        {{- form_errors(form) -}}
        {{- form_widget(form) -}}
    </div>
{%- endblock form_row %}
```

On souhaite par exemple ajouter une classe CSS à chaque `form-row`.

Pour cela il suffit de :

- Créer un nouveau fichier template dans lequel on redéfinit le fragment `form-row`

```
{# /app/Resources/views/fields.html.twig #}

{% block form_row %}
{% spaceless %}
    <div class="maClasse">
        {{ form_label(form) }}
        {{ form_errors(form) }}
        {{ form_widget(form) }}
    </div>
{% endspaceless %}
{% endblock form_row %}
```

On ajoute l'attribut « `class` » à l'élément `div` qui entoure chaque ligne.

- Indiquer à notre template de vue d'utiliser ce template

```
{% form_theme form '::fields.html.twig' %}
{% extends "::layout.html.twig" %}

[...]
{% block body %}
    {{ form_start(form) }}
        {{ form_widget(form) }}
    {{ form_end(form) }}
[...]
```

Et voilà, le tour est joué.

Evidemment, il ne faut pas hésiter à examiner le contenu de template de base des formulaires (`form_div_layout.html.twig`) pour voir comment sont constitués les différents fragments.

Pour plus d'informations sur le sujet :

<http://symfony.com/fr/doc/current/book/forms.html#habillage-de-formulaire-theming>

Revenons à notre application.

### ➤ **Création du fichier template de redéfinition des fragments de formulaire**

Créer sous `/app/Resources/views` un nouveau fichier nommé `fields.html.twig`.

Ecrire le contenu de ce fichier comme décrit ci-après :

#### ▪ **Tout d'abord, nous devons étendre le template de formulaire de base**

Il s'agit du fichier « `form_div_layout.html.twig` » qui se trouve dans le dossier `/vendor/symfony/symfony/src/Symfony/Bridge/Twig/Resources/views/Form`

```
{% extends 'form_div_layout.html.twig' %}
```

#### ▪ **On redéfinit ensuite le bloc *form-row***

Ouvrez le template de formulaire de base `form_div_layout.html.twig` afin de copier tout le bloc `{% block form_row %} ... {% endblock form_row %}` puis collez-le à la suite de la ligne précédente dans votre nouveau template.

On ajoute les `div` qui vont bien afin de gérer les classes css bootstrap.

On ajoute la classe `form-contrôle` au champ de formulaire.

```
{# On commence par simplement ajouter le form-group au row de nos formulaires #}
{% block form_row %}
{% spaceless %}
    {{ form_errors(form) }}
    <div class="form-group">
        <div class="col-sm-2">
            {{ form_label(form) }}
        </div>
        <div class="col-sm-3">
            {% set attr = attr|merge({'class': attr.class|default('') ~ ' form-control'}) %}
            {{ form_widget(form) }}
        </div>
    </div>
{% endspaceless %}
{% endblock form_row %}
```

La commande `spaceless` permet de supprimer les espaces inutiles dans le code HTML qui sera généré, afin d'obtenir des tailles de pages optimisées.

#### ▪ **On redéfinit ensuite le bloc *form-start***

On doit ici ajouter la classe `form-horizontal` à la balise `form`.

Il n'est pas nécessaire de réécrire tout le fragment : on fait appel à `parent()` qui est l'équivalent d'un copier-coller depuis le template hérité.

L'ajout de la classe se réalise à l'aide de la commande `set attr`.

```

{# Puis on redéfinit le formulaire #}
{% block form_start -%}
{% spaceless %}
    {% set attr = attr|merge({'class': attr.class|default('') ~ ' form-horizontal'}) %}
    {{ parent() }}
{% endspaceless %}
{%- endblock form_start %}

```

#### ▪ On redéfinit ensuite le bloc *form-errors*

On ne peut pas ici procéder comme pour le formulaire car la classe css à intégrer se fait dans une balise `<div>` à ajouter après le condition du bloc d'origine.

Vous allez donc récupérer le fragment `{% block form_errors %} ... {% endblock form_errors %}` depuis le template de formulaire de base et l'ajoutez à votre fichier.

Apportez les modifications suivantes :

```

{# Puis on redéfinit les blocs erreurs #}
{% block form_errors -%}
{% spaceless %}
    {% if errors|length > 0 -%}
        <span class="alert-danger">
            <ul>
                {%- for error in errors -%}
                    <li>{{ error.message }}</li>
                {%- endfor -%}
            </ul>
        </span>
    {%- endif %}
{% endspaceless %}
{%- endblock form_errors %}

```

#### ▪ Enfin, on redéfinit ensuite le bloc *form-submit*

Ici, on fait de nouveau appel à `parent()`. Donc il n'est pas nécessaire de récupérer le code du fragment de base.

On ajoute les classes `btn` et `btn-primary` au bouton et on le place dans deux div imbriquées avec les classes CSS adéquates.

```

{# On redéfinit le bouton submit #}
{%- block submit_widget %}
{% spaceless %}
    {% set attr = attr|merge({'class': attr.class|default('') ~ ' btn btn-primary'}) %}
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-4">
            {{ parent() }}
        </div>
    </div>
{% endspaceless %}
{%- endblock submit_widget %}

```



Vous obtenez :

33/35

```
{% extends 'form_div_layout.html.twig' %}

{# On commence par simplement ajouter le form-group au row de nos formulaires #}
{% block form_row %}
{% spaceless %}
    {{ form_errors(form) }}
    <div class="form-group">
        <div class="col-sm-2 ">
            {{ form_label(form) }}
        </div>
        <div class="col-sm-3">
            {% set attr = attr|merge({'class': attr.class|default("") ~ ' form-control'}) %}
            {{ form_widget(form) }}
        </div>
    </div>
{% endspaceless %}
{% endblock form_row %}

{# Puis on redéfinit le formulaire #}
{% block form_start -%}
{% spaceless %}
    {% set attr = attr|merge({'class': attr.class|default("") ~ ' form-horizontal'}) %}
    {{ parent() }}
{% endspaceless %}
{%- endblock form_start %}

{# Puis on redéfinit les blocs erreurs #}
{% block form_errors -%}
{% spaceless %}
    {% if errors|length > 0 -%}
        <div class="alert-danger">
            <ul>
                {%- for error in errors -%}
                    <li>{{ error.message }}</li>
                {%- endfor -%}
            </ul>
        </div>
    {%- endif %}
{% endspaceless %}
{%- endblock form_errors %}

{# On redéfinit le bouton submit #}
{%- block submit_widget %}
{% spaceless %}
    {% set attr = attr|merge({'class': attr.class|default("") ~ ' btn btn-primary'}) %}
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-4">
            {{ parent() }}
        </div>
    </div>
{% endspaceless %}
{%- endblock submit_widget %}
```

## ➤ Etendre ce template dans nos vues

Vous allez me dire : « *c'est presque aussi long que la solution précédente !!!* ».

Oui, sauf que l'on écrit ce fichier une seule fois et qu'il peut ensuite être utilisé par tous nos formulaires.

Il suffit de faire référence à notre « **thème** » dans nos templates de vues.

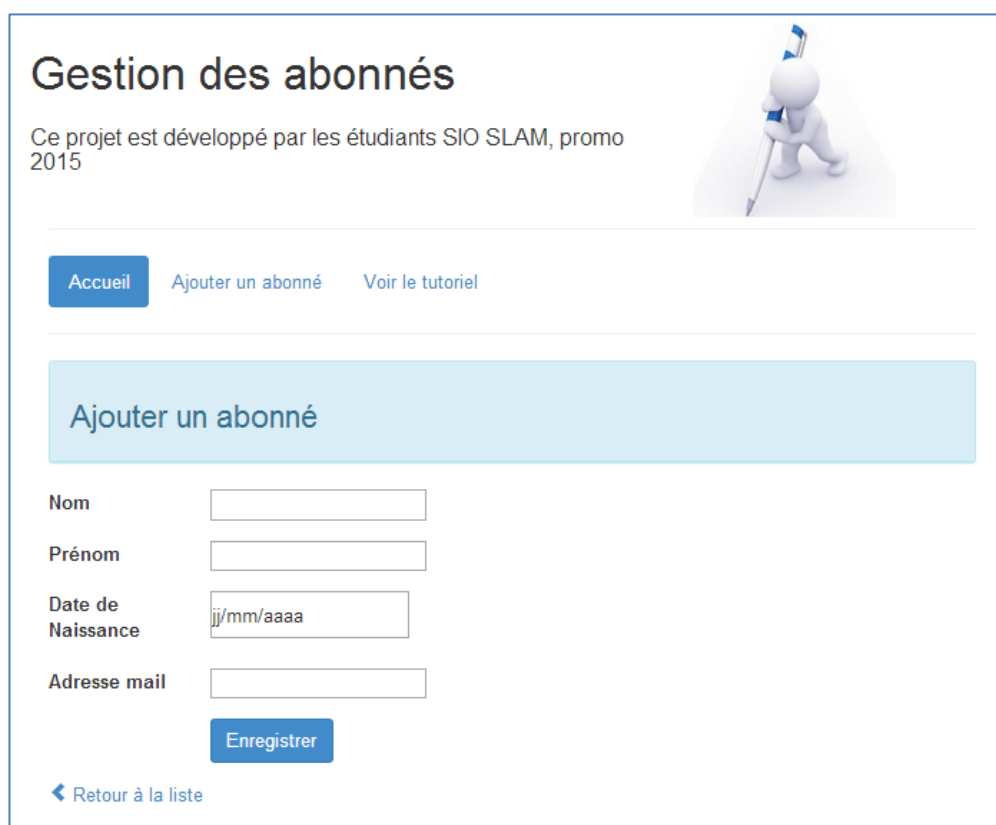
Une seule petite instruction à ajouter au début de vos templates et c'est tout.

Complétez votre template `ajouter.html.twig` comme indiqué ci-dessous :

```
{% extends "::layout.html.twig" %}
{% form_theme form '::fields.html.twig' %}

{% block title %}
Ajout abonné - {{ parent() }}
{% endblock %}
```

Testez. Vous devez obtenir :



The screenshot shows a web application titled "Gestion des abonnés". Below the title, it says "Ce projet est développé par les étudiants SIO SLAM, promo 2015". There is a navigation bar with links: "Accueil", "Ajouter un abonné", and "Voir le tutoriel". The main content area has a light blue header with the text "Ajouter un abonné". Below this, there are four input fields: "Nom", "Prénom", "Date de Naissance" (with a date picker showing "jj/mm/aaaa"), and "Adresse mail". A blue "Enregistrer" button is at the bottom right of the form. A link "Retour à la liste" is at the bottom left.

C'est un peu plus beau, n'est-ce pas ?

Et, en cas d'erreur :

35/35

## Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015



[Accueil](#)
[Ajouter un abonné](#)
[Voir le tutoriel](#)

Ajouter un abonné

- Le nom doit comporter entre 3 et 5 caractères
- Le prénom doit comporter entre 3 et 5 caractères
- La date de naissance doit être postérieure à 1940

Nom

Prénom

Date de Naissance

Adresse mail

[Enregistrer](#)

[Retour à la liste](#)

Il ne reste plus qu'à ajouter la référence au template `ajouter.html.twig` dans les vues `modifier.html.twig` et `supprimer.html.twig` et le tour est joué !

## Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015



[Accueil](#)
[Ajouter un abonné](#)
[Voir le tutoriel](#)

Modification de l'abonné n° 2

Nom

Prénom

Date de Naissance

Adresse mail

[Valider](#)

[Retour à la liste](#) [Retour à l'abonné](#)

## Gestion des abonnés

Ce projet est développé par les étudiants SIO SLAM, promo 2015



[Accueil](#)
[Ajouter un abonné](#)
[Voir le tutoriel](#)

Supprimer un abonné

Etes-vous certain de vouloir supprimer l'abonné "DUPONT Pierre" ?

[Supprimer](#)

[Retour à l'abonné](#)

Alors, trop fort ces Frameworks, non ?

Mais attendez, ce n'est pas fini : vous allez voir encore plus de puissance !!!!

**Suite au prochain TP ...**

