Le framework Symfony2

Historique et définition

- Framework: cadre de travail
- Objectif: améliorer la productivité des développeurs (se concentrer sur les choses importantes), structurer l'application, accélère le développements

- PHP: 1995
- PHP5 (POO): 2005
- 1ère version de Symfony en 2005, puis 1.4 en 2009
- Basé sur le principe MVC
- Créé par un français (Fabien Potencier) de l'entreprise Sensio Labs
- Gros changements en 2011 avec l'apparition de la version 2

- Actuellement en version 2.7.7

Retour sur le MVC

- Modèle/Vue/Contrôleur

- Contrôleur
  - Génère la réponse à la requête envoyée par l'utilisateur
  - Utilise les autres composants (vue/modèle)

- Modèle
  - Gère les données de notre application
  - Sait comment récupérer ces informations (en général SQL)
  - Permet au contrôleur d'utiliser les données

- Vue
  - Affiche les pages
  - Traite tout le côté design de l'application

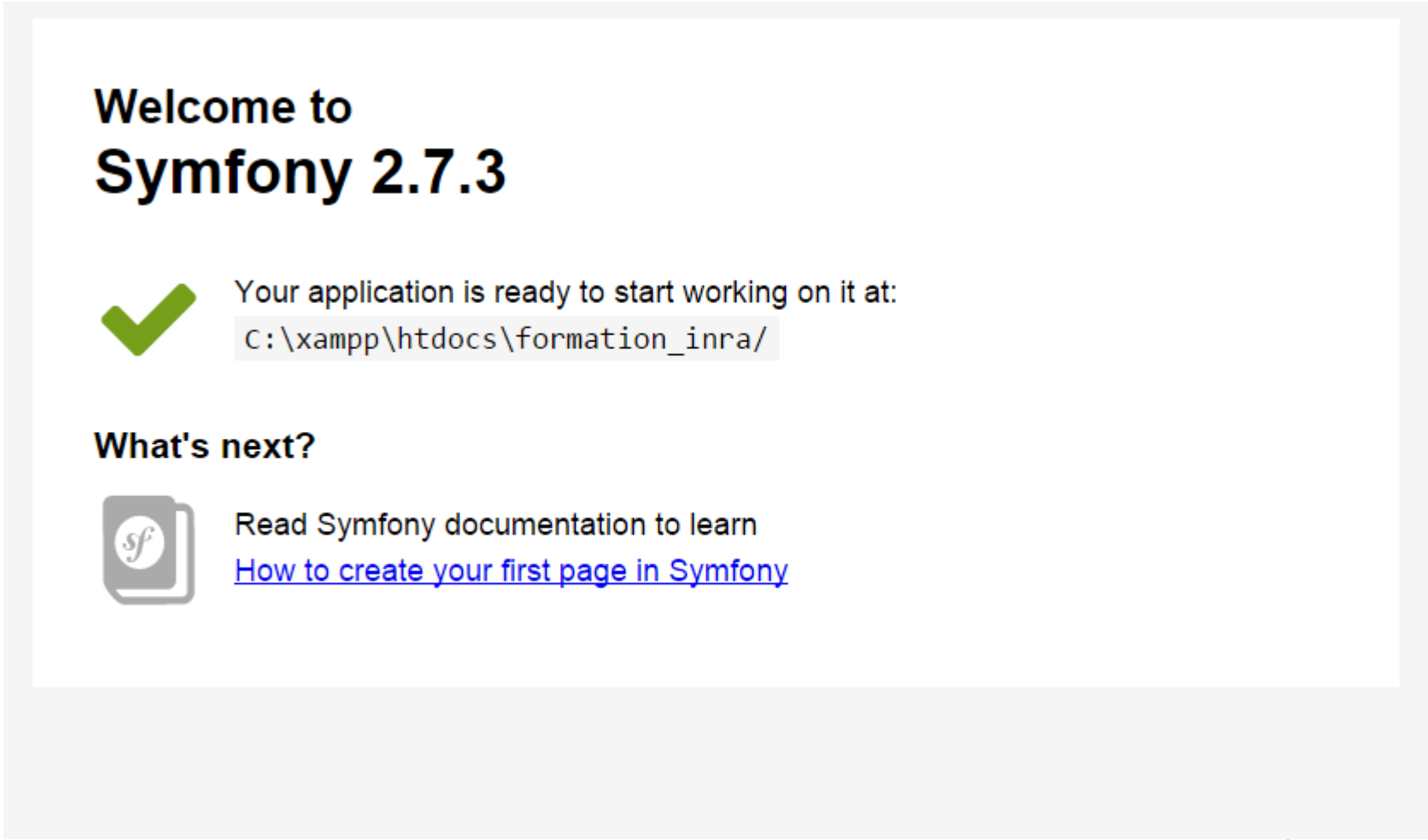
Installation de Symfony2

- En ligne de commande:
  - Se mettre dans le répertoire www (ou htdocs pour xampp)
  - Récupérer le fichier symfony.phar (http://symfony.com/installer)
  - Se placer dans le répertoire et exécuter la commande php symfony.phar new nomApplii → cela va créer un répertoire nomAppli
  - Supprimer le répertoire app/cache
  - Se rendre sur la page internet

  - Télécharger le fichier composer.phar (curl -sS https://getcomposer.org/installer | php ou php -r "readfile('https://getcomposer.org/installer');" | php)

    → ceci créera un fichier composer.phar à la racine de l'espace de travail

Installation de Symfony2



Débuter avec Symfony2

- **L'architecture**
  - 4 répertoires principaux :
  - **App** : contient en gros toute la config de votre site et le cœur de Symfony
    - Notamment app/config/**parameters.yml** pour paramétrer votre base de données
    - Notamment app/config/**routing.yml** qui trace les « routes » (les URL affichées) de notre site et le comportement à avoir derrière
  - **Src** : contient tout le code source dans lequel on travaille. Il contient plusieurs namespaces (dans notre cas par exemple on mettra INRA) qui sont des « parties » de notre site, eux-mêmes contenant des bundles
  - **Vendor** : contient toutes les bibliothèques utilisées par Symfony (**Doctrine** pour la base de données, **Twig** pour le design…)
  - **Web** : contient ce que les utilisateurs vont voir. En gros les utilisateurs ne voient que ce que ce répertoire contient et rien d'autres

Débuter avec Symfony2

- Symfony fonctionne avec des bundles. Ceux-ci contiennent quelques répertoires intéressants:
  - **Controller** contient les contrôleurs de notre bundle
  - **Entity** : contient les entités (vulgairement les tables de notre base de données)
  - **Form** : contient les formulaires pour nos classes
  - **Resources** : répertoire très important pouvant contenir:
    - *Config* : Une config plus spécifique si besoin que celle présente dans app/config
    - *Public* : contient tous les codes css et js utilisés dans notre bundle, voire notre site
    - *Views* : contient les vues de notre MVC. Ce répertoire est découpé en sous-répertoires pour les sous-parties de notre site :
      - Default pour l'accueil
      - Animaux pour les animaux
      - …
      - A noter que pour chaque partie (par exemple interventions), on aura 4 fichiers Twig pour écrire notre vue (index, show,edit,add)
- Symfony utilise la bibliothèque Twig pour faire ses vues. En gros Twig est un langage permettant de mixer du code html avec des variables, sans pour autant dénaturer complètement le code.

Les fichiers de config

- Ils sont présents dans le répertoire app/config

- Config.yml
  - Fichier de configuration de symfony
- Routing.yml
  - Fichier définissant les « routes » de votre application, c'est-à-dire les relations entre URL et ce qui est lancé
- Security.yml
  - Fichier de gestion de la sécurité de votre application
- Services.yml
  - Fichier des services génériques de votre application
- Parameter.yml
  - Fichier de gestion de vos paramètres

Gestion de la base de données

- Exemple du fichier parameter.yml pour spécifier la base de données

parameters.yml

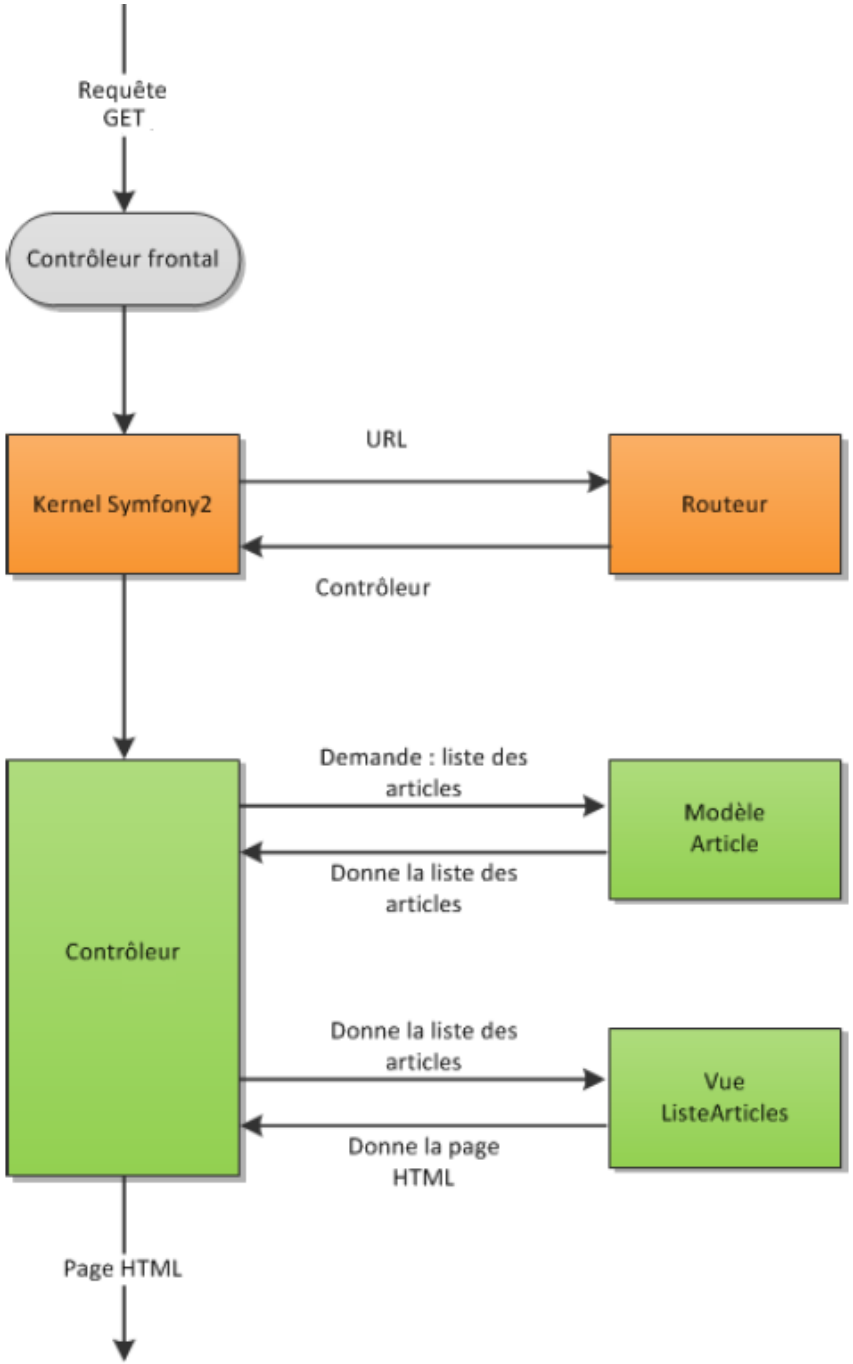
```yaml
# This file is auto-generated during the composer install
parameters:
    database_host: 127.0.0.1
    database_port: null
    database_name: symfony_formation_inra
    database_user: root
    database_password: null
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret: 3ee8d144ea605166a4d6c16adefdcfe047b1f7d4

```

- On crée une base de données en ligne de commande

  php app/console doctrine:database:create

```
c:\xampp\htdocs\formation_inra>php app/console doctrine:database:create
Created database `symfony_formation_inra` for connection named default
```


Requête
GET
Contrôleur frontal
Kernel Symfony2
URL
Routeur
Contrôleur
Contrôleur
Demande : liste des articles
Donne la liste des articles
Modèle Article
Donne la liste des articles
Donne la page HTML
Vue ListeArticles
Page HTML

Les bundles

- Symfony fonctionne en bundles, c'est-à-dire avec une liste de fonctionnalités de notre application, où bien encore des blocs d'une application

- La nomenclature suivie est namespace/nomBundle, le namespace pouvant être par exemple le nom d'une appli

- Pour créer un bundle:
  - Depuis la console: php app/console generate:bundle
  - Il demande un namespace: INRA/FormationBundle
  - Symfony nous guide dans la suite des épreuves

Générer une entité

- La plupart des applications reposent sur des bases de données. Dans le cas de Symfony, on dit « vulgairement » qu'ici on utilise des entités.
- Une entité c'est quoi? C'est juste une classe représentant une chose de notre monde

- Création par la ligne de commande
  - Depuis la console php app/console doctrine:generate:entity
  - On donne un nom: namespaceBundle:Entite
  - On suit la procédure

Génération des contrôleurs associés aux entités

- Avoir des entités c'est bien, créer des bundles c'est bien, mais ça ne fait pas un MVC
- On a la base, c'est-à-dire une base de données viable
- Il reste à gérer l'affichage et les saisies

- L'outil crud
  - Va permettre de générer tout ce qu'il faut pour débuter avec les contrôleurs pour une entité donnée
    - Le contrôleur
    - Les formulaires

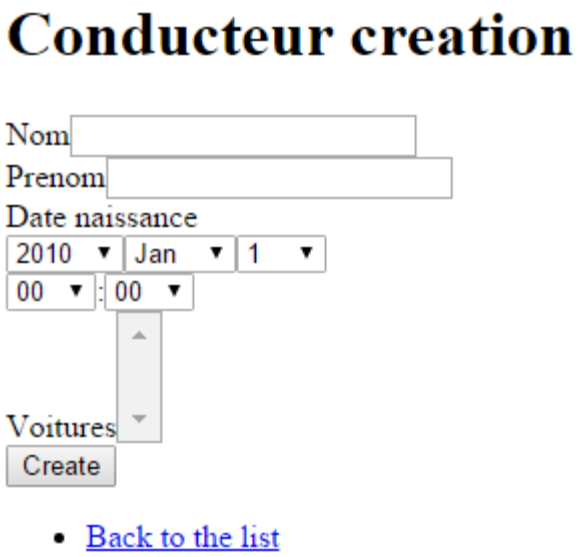
```
php app/console generate:doctrine:crud
```

  - Génère les vues:
    - Création/édition/suppression
    - Index
    - Formulaires associés

Génération des contrôleurs associés aux entités

- Chaque contrôleur a plusieurs méthodes:
  - indexAction
  - createAction
  - createCreateForm
  - newAction
  - showAction
  - editAction
  - createEditForm
  - updateAction
  - deleteAction
  - createDeleteForm

- Chaque contrôleur généré a une route spécifique définie
- Chaque méthode a une route spécifique (/, /new, /{id}, /{id}/edit) avec des méthodes associées
- Par exemple, aller sur /monentite/new revient à appeler la méthode newAction() du contrôleur de l'entité MonEntite



Les vues

- La partie vue est gérée par Twig.
- Se gère depuis le répertoire Resources/views
- Twig est relativement simple, consiste en de l'HTML avec des variables (amenées par le contrôleur)
- Pour changer les vues, on change juste le CSS

Les vues

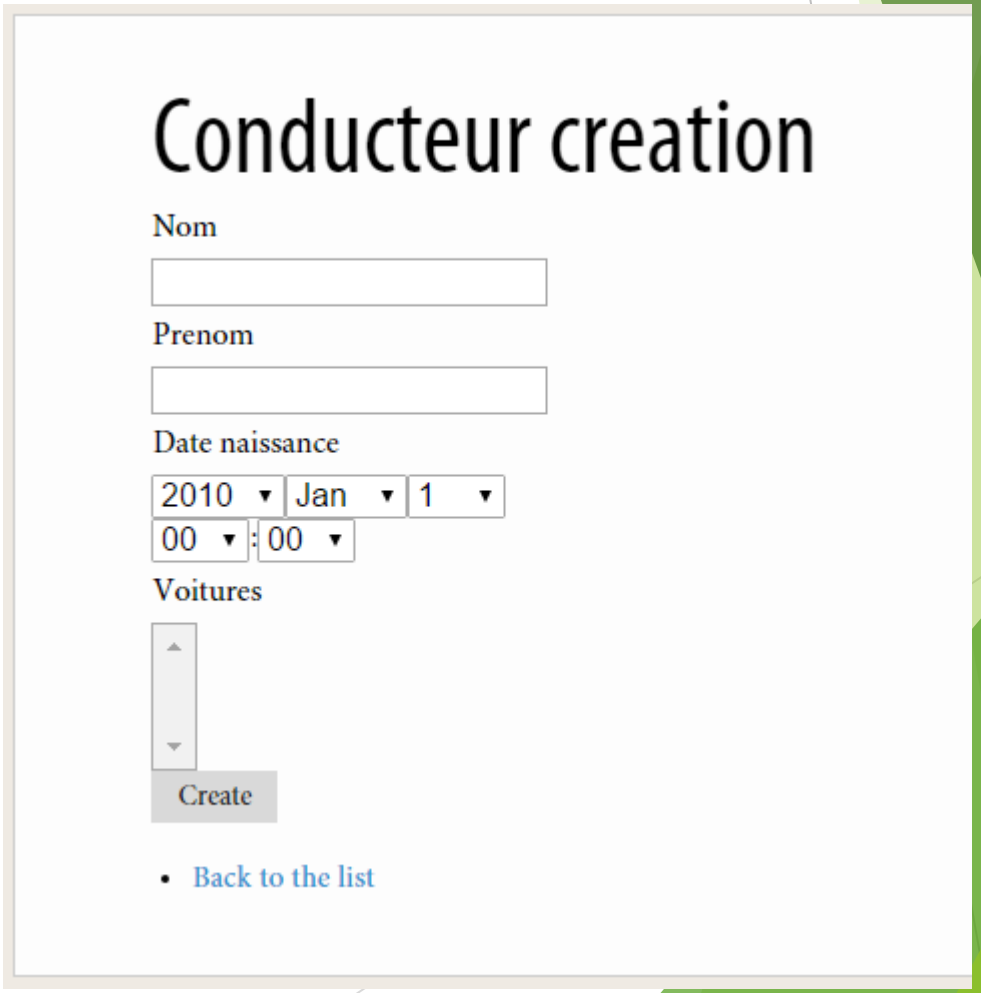
- Les formulaires

```twig
{% extends '::base.html.twig' %}

{% block body -%}
    <h1>Conducteur creation</h1>

    {{ form(form) }}

        <ul class="record_actions">
    <li>
        <a href="{{ path('conducteur') }}">
            Back to the list
        </a>
    </li>
</ul>
{% endblock %}
```

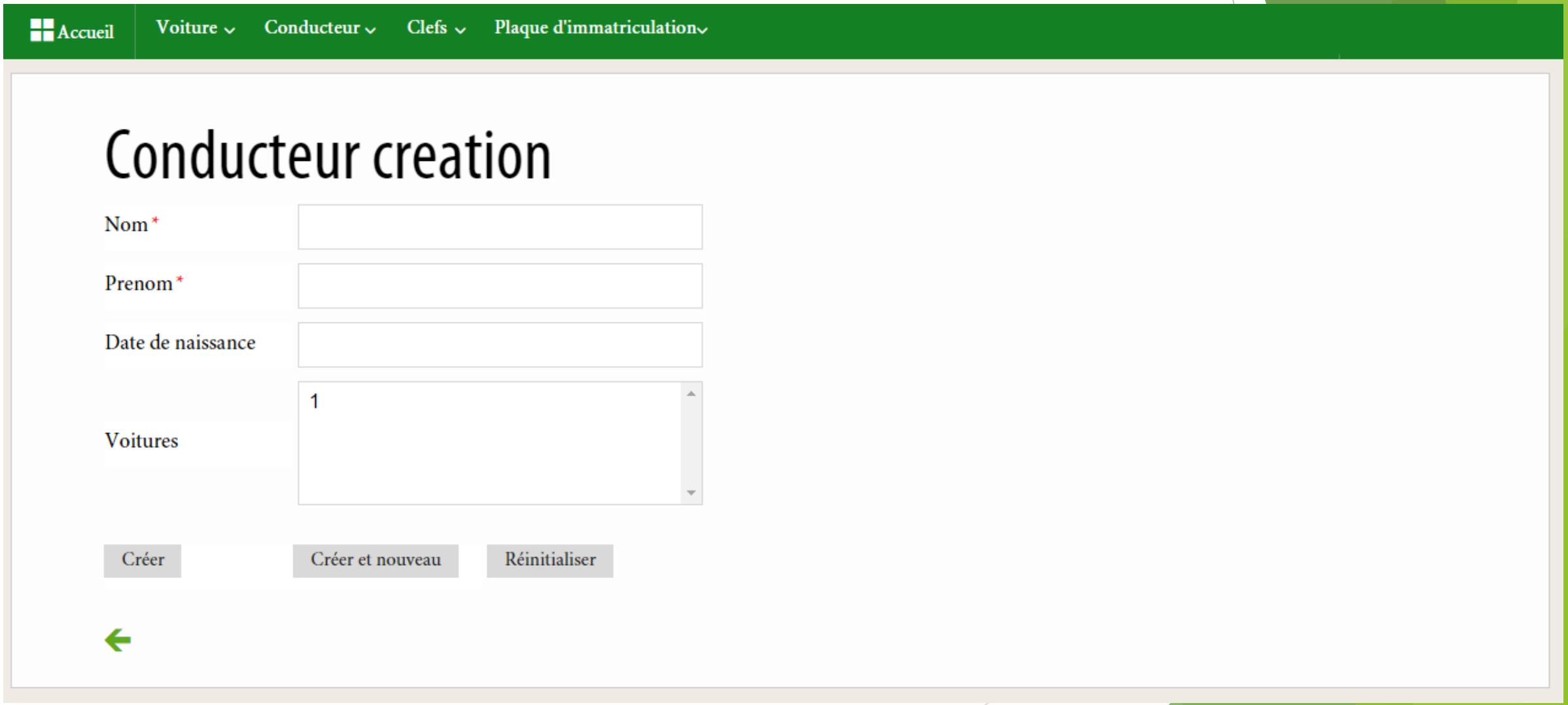


Les vues

```
{{ form_start(form) }}
    <div class="input-control text size2">{{ form_label(form.nom, 'Nom') }}</div>
    <div class="input-control text size4">{{ form_widget(form.nom) }}</div><br />
    <div class="input-control text size2">{{ form_label(form.prenom, 'Prenom') }}</div>
    <div class="input-control text size4">{{ form_widget(form.prenom) }}</div><br />
    <div class="input-control text size2">{{ form_label(form.datenaissance, 'Date de naissance') }}</div>
    <div class="input-control text size4">{{ form_widget(form.datenaissance) }}</div><br />
    <div class="input-control text size2">{{ form_label(form.voitures, 'Voitures') }}</div>
    <div class="input-control select size4">{{ form_widget(form.voitures) }}</div><br />
    <br/>
    <div class="input-control text size2 float-left">{{ form_widget(form.submit) }}</div>
    <div class="input-control text size2">{{ form_widget(form.ajouter) }}</div>
    <a href="{{ path(app.request.attributes.get('_route')) }}" >{{ form_widget(form.reinit) }}</a>
    <br/>
    {{ form_rest(form) }}
    {{ form_errors(form) }}
            {{ form_end(form) }}

    <a href="{{ path('conducteur') }}">
        <span class="icon-arrow-left fg-green large" title="return"></span>
    </a>
```

Les vues


Accueil
Voiture
Conducteur
Clefs
Plaque d'immatriculation
Conducteur creation
Nom *
Prenom *
Date de naissance
Voitures
1
Créer
Créer et nouveau
Réinitialiser

Les bundles associés

- Un des gros avantages de Symfony
- Il en existe tout un tas très simples d'installation et d'utilisation
  - Gestion des utilisateurs (avec ou sans gestion du LDAP)
  - Formulaires de contact
  - Filtres
  - Partie admin
  - HighCharts aussi par exemple

- Installation
  - En 3 étapes
    - Dans le fichier composer.json
    - Dans le fichier app/AppKernel.php
    - Régénérer le répertoire web

Avantages / inconvénients

- Avantages
  - Gain de productivité
  - Code propre (bien organisé)
  - Architecture MVC intégrée
  - Grosse communauté et nombreux bundles, donc nombreuses mises à jour
  - Possibilité de générer sa base avant ou après les entités, facilité de maintenance
  - Très rapide à installer et configurer

- Inconvénients
  - Courbe d'apprentissage
    - POO
    - MVC
  - Complexité de la configuration (XML, YAML, annotation)

Une petite démo

https://wiki.inra.fr/wiki/wisards/Symfony/