TP7 : Traduction

L'objectif principal est de créer un sytème de traduction, avec un switch pour changer de langue. Partons du principe anglais / fr.
Attention : nous ne parlons pas de traiter de la traduction du contenu de la base de données mais de l'affichage du site.

1. Installation & configuration du service

Comme d'habitude, pour utiliser un service nouveau, il faut l'installer avec Composer. Cependant, chance pour nous, il est déjà inclus dans le « website-skeleton » que nous utilisons.

Une fois installé, le service sera configuré par le fichier « config/packages/translation.yml ».

```yaml
# config/packages/translation.yaml
framework:
    default_locale: 'en'
    translator:
        default_path: '%kernel.project_dir%/translations'
```

L'option "**default_locale**" permet de définir la langue par défaut du site. Nous allons configurer le Français en indiquant "**fr**".
L'option "**default_path**" indique le chemin vers le dossier contenant les traductions.

2. Utilisation du service

A/ Dans les contrôleurs

Si vous devez envoyer une chaîne à traduire depuis un contrôleur, comme dans un message flash par exemple, il est possible de le faire en utilisant les instructions suivantes dans le contrôleur.

```
use Symfony\Contracts\Translation\TranslatorInterface;

/**
* Méthode
*/
public function index(TranslatorInterface $translator)
{
   $message = $translator->trans('Message à traduire');
  }
```

B/ Dans les vues avec Twig

Afin de personnaliser l'interface, on peut traduire les différents textes indiqués « en dur » dans notre application.

Par exemple :

```
{# concert/index.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}
    <h1>Bonjour et Bienvenue ! </h1>
    <h2> Nos concerts à venir.... </h2>
    <div class="container">
```

Je souhaite traduire « Bonjour et Bienvenue ! » ainsi que « nos concerts à venir… ».

Deux possibilités:

- soit via la balise « |trans » qui permet d'indiquer qu'une traduction est présente.
Observer bien la façon dont sont indiquées les clés de traduction.

```
{# concert/index.html.twig #}
{% extends 'base.html.twig' %}
{% block body %}
    <h1>{{ 'welcome'|trans }}</h1>
    <h2>{{ 'concert.to_come'|trans }} </h2>
```

```
    <div class="container">
```

- soit via les balises « {% trans %} {% endtrans %} qui encadre le texte :

```
{# concert/index.html.twig #}
{% extends 'base.html.twig' %}
{% block body %}
    <h1>{% trans %}welcome{% endtrans %}</h1>
    <h2>{% trans %}Concerts à venir…{% endtrans %}</h2>
    <div class=« container">
```

Le résultat est le même, j'ai une préférence pour le premier.

3. Création des fichiers de traduction

A/ Création manuelle

Le fichier sera à créer en utilisant le code de la langue sur 2 caractères (à l'emplacement de xx), comme **fr** pour le français.
On créera donc autant de fichiers que de langues, sous le format "**translation/ messages.xx.yaml**"
Pour traduire notre message dans le twig précédent, nous allons indiquer ceci.

```
#translations/messages.fr.yaml
welcome: Bonjour et bienvenue !
concerts:
  to_come: Nos concerts à venir....
```

Observer bien l'indentation. A quoi correspond-elle ?
Que va-t-elle nous permettre de faire à votre avis ?

Je vous invite de même à créer un fichier « messages.en.yaml » et d'y ajouter les traductions correspondantes sur le même modèle.

B/ Création Automatique

Il existe une commande permettant de créer automatiquement le fichier de traduction pour la langue de notre choix.
Cette commande va parcourir tous les fichiers et intégrer toutes les chaines dans le fichier correspondant à la langue choisie.
ATTENTION : cette commande ne traduit pas le contenu

```
php bin/console translation:update --force fr
```

Le fichier **xlf** sera créé automatiquement. Vous pouvez aller voir à quoi ça ressemble, et où il est nécessaire de placer la traduction.
(Je trouve ça personnellement moins lisible que des fichiers yaml, mais c'est selon les goûts de chacun).

4. Mettre en place le choix de la langue

Pour permettre aux utilisateurs de **choisir la langue** dans laquelle le site doit s'afficher, nous allons devoir stocker la langue choisie dans la session.
Pour commencer, nous allons créer une **variable d'environnement** qui contiendra la liste des langues souhaitée.
Cette variable sera déclarée dans "**config/services.yaml**" de cette façon

```
parameters:
    # ici vos autres variables
    app.locales: [fr, en]
```

Nous allons également déclarer cette variable globalement dans les fichiers Twig afin d'y accéder depuis tous les fichiers. Nous l'appellerons "**locales**" et l'intégrons dans « **config/packages/twig.yaml**"

```
twig:
    default_path: '%kernel.project_dir%/templates'
    debug: '%kernel.debug%'
    strict_variables: '%kernel.debug%'
    # On déclare ci-dessous le nom de la variable globale
    globals:
        locales: '%app.locales%'
```

On crée maintenant une **méthode** dans un **contrôleur** qui permettra de **changer la langue**.
Cette méthode contiendra le code suivant

```
/**
 * @Route("/change_locale/{locale}", name="change_locale")
 */
public function changeLocale($locale, Request $request)
{
    // On stocke la langue dans la session
    $request->getSession()->set('_locale', $locale);

    // On revient sur la page précédente
    return $this->redirect($request->headers->get('referer'));
}
```

Nous appellerons cette méthode depuis notre menu ou tout fichier Twig dans lequel nous donnerons le choix de la langue aux utilisateurs.
Je vous laisse ici réfléchir à où est-ce qu'il serait le plus opportun de placer ce bout de code pour qu'il soit accessible depuis le site tout entier…

```
{% for locale in locales %}
    {% if locale != app.request.locale %}
        <a href="{{ path('change_locale', {'locale': locale}) }}">{{ locale }}</a>
    {% endif %}
{% endfor %}
```

Et voilà, la traduction est en place !

5. Intercepter les requêtes

Afin de gérer les changements de langues dans le coeur de Symfony (Kernel), nous allons nous attacher à chacune des requêtes en créant un souscripteur d'évènements (EventSubscriber). Cela permet d'écouter toutes les évènements et d'intervenir sur l'un d'enter eux.
Pour ce faire, nous allons entrer la commande suivante :

```
php bin/console make:subscriber LocaleSubscriber
```

A la question de l'évènement à utiliser, entrer :
Symfony\Component\HttpKernel\Event\RequestEvent

Le fichier "*LocaleSubscriber.php*" est créé dans le dossier "src/EventSubscriber"

Nous allons remplacer son code par celui-ci

```php
<?php
namespace App\EventSubscriber;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpKernel\Event\RequestEvent;
use Symfony\Component\HttpKernel\KernelEvents;

class LocaleSubscriber implements EventSubscriberInterface
{
    // Langue par défaut
    private $defaultLocale;

    public function __construct($defaultLocale = 'en')
    {
        $this->defaultLocale = $defaultLocale;
    }

    public function onKernelRequest(RequestEvent $event)
```

```
    {
        $request = $event->getRequest();
        if (!$request->hasPreviousSession()) {
            return;
        }

        // On vérifie si la langue est passée en paramètre de l'URL
        if ($locale = $request->query->get('_locale')) {
            $request->setLocale($locale);
        } else {
            // Sinon on utilise celle de la session
            $request->setLocale($request->getSession()->get('_locale',
$this->defaultLocale));
        }
    }

    public static function getSubscribedEvents()
    {
        return [
            // On doit définir une priorité élevée
            KernelEvents::REQUEST => [['onKernelRequest', 20]],
        ];
    }
}
```

Et Go plus qu'à tester :)

Vous pouvez déclarer de nouvelles langues en créant les fichiers de traductions liés, ainsi que traduire les autres pages de votre site.