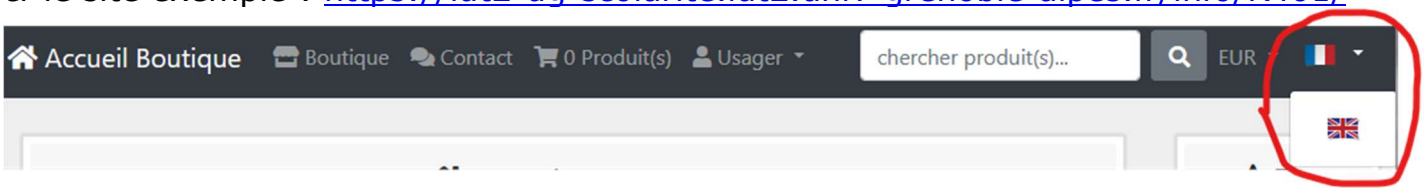
R4.01 - TP 02 : Internationalisation
1 séance

Prérequis : Vous devez impérativement terminer le TP01 avant de débuter celui-ci

2.0 - Objectif du TP

L'objectif de ce TP est de mettre en place l'internationalisation (i18n) sur notre embryon d'application Symfony. Nous allons faire en sorte que les 2 pages statiques, créées lors du TP01, puissent être consultées aussi bien en français qu'en anglais (et/ou toute autre langue qu'il vous plaira d'ajouter). Cela sous-entend bien sûr que le contenu de la barre de navigation devra aussi être traduit. Cette même barre de navigation devra également comporter un petit menu déroulant permettant de choisir la langue affichée sur le site, *cf* le site exemple : https://iut2-dg-scolarite.iut2.univ-grenoble-alpes.fr/info/R401/



⚠ *Lors des prochains TP, nous ne nous « amuserons » plus à traduire les contenus des nouveaux templates que nous créerons.* ***Par contre, nous devrons bien faire en sorte que toutes les routes créées aient toujours des URL préfixées par la locale****.*

2.1 –Internationaliser

Les différentes étapes à réaliser vous ont été décrites dans le cours n°2. Il vous suffit donc de suivre ces étapes (en étant vigiliant·e). ⚠ *Lorsque vous modifiez un fichier de configuration au format YAML, soyez vigilant avec l'indentation. Les informations de même niveau doivent avoir la même indentation, à l'espace près !*

Rappel des étapes à réaliser :

1. Définissez la **locale par défaut** et la **locale de repli** dans le fichier **`config/packages/translation.yaml`** (*cf cours 02, page 4*).

2. Définissez les **locales supportées** par votre application dans le fichier **`config/services.yaml`** (*cf cours 02, page 5*).

3. Modifiez **toutes les routes** de votre application pour que la locale soit transmise dans chaque URL. Pour cela, l'URL doit être préfixée par un paramètre **`{_locale}`** dont on contraindra les valeurs possibles avec un attribut **`requirements`** qui sera égal à l'expression régulière qui vérifie la liste des locales supportées.
   Pour la route de la page d'accueil, on rajoutera aussi un attribut **`defaults`** pour donner une valeur par défaut à la locale lorsque l'on navigue pour la première fois vers la page d'accueil du site. (*cf cours 02, page 4-5*).

4. Modifiez **tous vos *templates*** (y compris le *layout* et la *navbar*) en encadrant les éléments (mot, phrase ou paragraphe) que vous voulez traduire par des balises Twig **`{% trans %}…{% endtrans %}`**. Les éléments à traduire devront être remplacés par un identifiant de la forme :
   `repertoire_du_template.nom_du_template.identifiant_element_a_traduire`

Ces identifiants seront donc ceux qui apparaîtront comme **source** dans les catalogues qui seront produits à l'étape suivante (*cf cours 02, page 8*).

5. Faire produire au CLI de symfony, pour chaque langue **LL** proposée (donc au moins **fr** et **en**), le catalogue qui contiendra toutes les sources définies à l'étape précédente. Pour la langue **LL**, utilisez la commande :
   `php bin/console translation:extract --force LL` (*cf cours 02, page 11*).

6. L'étape précédente a produit, pour chaque locale **LL**, un catalogue (fichier XLIFF) nommé **translations/messages+intl-icu.LL.xlf**. Vous devez maintenant jouer au traducteur ! Dans chacun de ces catalogues, remplissez les balises **<target>**, associées à chaque balise **<source>**, afin d'indiquer par quoi sera remplacé l'identifiant défini dans la source lorsqu'il sera affiché avec la locale **LL** (*cf cours 02, page 10*).

Les étapes 4,5,6 peuvent être répétées plusieurs fois sans perdre le travail qui a été fait précédemment.

Pour vérifier que la traduction est bien en place, naviguez vers les pages qui sont censées être traduites et vérifiez que la locale est prise en compte :

- http://localhost:8000/fr doit afficher votre page d'accueil en français
- http://localhost:8000/en doit afficher cette même page, mais en anglais

… et ceci doit fonctionner pour toutes pages de votre site !

2.2 – Menu « langue » dans la barre de navigation

On souhaite maintenant disposer dans la barre de navigation (affichée sur chaque page du site) d'un menu déroulant permettant à l'utilisateur de choisir la langue dans laquelle est affichée la page actuelle. Il faut donc coder ce menu déroulant, à l'aide de Twig, dans le *template* **navbar.html.twig**.
Quelques indications pour réaliser cela (lisez-les toutes avant de coder !) :

1. Il faut avoir accès dans le *template* à la liste des locales supportées par l'application. Ce paramètre, défini lors de l'étape 2, doit être « injecté » dans Twig en complétant le fichier **config/packages/twig.yml** de la façon suivante :

```
#config/packages/twig.yml
twig:
  globals:
     supported_locales: '%app.supported_locales%'
```

2. Vous disposerez alors, dans vos *templates* Twig, d'une variable **supported_locales** qui sera une chaîne de caractères (**'fr|en'**) contenant les locales supportées, séparées par le caractère **'|'**. Pour parcourir ces locales dans une itération, vous pourrez utiliser la fonction **split** de Twig qui permet de convertir une chaîne de caractères en tableau, en précisant quel est le séparateur des différents éléments :
{% for uneLocale in supported_locales | split('|') %} … {% endfor %}

3. Le titre du menu déroulant doit être un texte (ou une image) correspondant à la locale actuelle. Pour savoir quelle est la valeur actuelle de la locale, vous pouvez en Twig accéder à la requête HTTP et y récupérer la locale actuelle :
{% set locale = app.request.attributes.get('_locale') %}

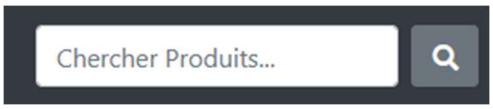
4. Dans l'itération qui va produire le contenu du menu déroulant, il faudra pour chaque locale supportée (sauf la locale actuelle !) produire une URL permettant de naviguer vers la page actuelle, mais en changeant la locale définie dans l'URL de cette page. Rappelez-vous qu'en Symfony+Twig, une URL ne doit pas être « codée » en dur mais doit être forgée, à partir d'un nom de route, à l'aide de la fonction Twig **path**. Si la route en question contient des paramètres, ceux-ci doivent être fournis à la fonction **path** dans un tableau associatif Twig : **{'param1':'val. param1', …}**. Pour pouvoir fabriquer nos liens de rechargement de la page actuelle, il va falloir :

4.a - Disposer du nom de la route actuelle (**route**) :
```
{% set route = app.request.attributes.get('_route') %}
```
4.b - Disposer du tableau des paramètres de la route actuelle (**params**) :
```
{% set params = app.request.attributes.get('_route_params') %}
```
4.c - Modifier la valeur du paramètre **'_locale'** dans **params** en lui donnant la valeur de la locale (**uneLocale**) dans laquelle on veut afficher la page :
```
{% set params = params | merge({'_locale': uneLocale }) %}
```
4.d - Et enfin forger l'URL pour naviguer vers la page actuelle, en utilisant la route actuelle et ses paramètres modifiés :
```
<a href="{{ path(route,params) }}" …>
```

Vous avez bien lu ces indications ? Bravo, vous êtes prêt·e à coder le menu déroulant !

2.3 – Champ de recherche dans la barre de navigation

Il vous reste maintenant à coder le champ de recherche de produits disponible lui aussi dans la barre de navigation :



Quelques indications rapides pour faire cela :

1. Dans le contrôleur **BoutiqueController**, coder une méthode et sa route (**app_boutique_chercher**) qui pourra répondre à une recherche de produits.

- L'URL de la route devra avoir un paramètre **recherche** qui contiendra la chaîne de caractères à rechercher parmi les attributs **libelle** et **texte** de nos produits :
/{_locale}/boutique/chercher/{recherche}
⚠ Il faut, dans la route, donner par défaut la valeur chaîne vide ("") au paramètre **recherche** *(vous comprendrez bientôt pourquoi…)* :

```
#[Route(
  path: '/chercher/{recherche}',
  name: 'app_boutique_chercher',
  requirements: ['recherche' => '.+'], // regexp pour avoir tous les car, / compris
  defaults: ['recherche' => ''])]
```

- La méthode associée à la route aura le prototype suivant :
```
public function chercher(BoutiqueService $boutique,
                         string $recherche) : Response
```
Cette méthode devra utiliser **BoutiqueService** pour trouver la liste des produits dont **libelle** ou **texte** contiennent $**recherche** (voir la méthode proposée dans le service : **findProduitsByLibelleOrTexte**).
Le résultat sera alors transmis à un *template* **boutique/chercher.html.twig** qui se chargera d'afficher le résultat de la recherche.

2. Dans le *template* **`navbar.html.twig`**, il faudra disposer d'un champ **`<input>`** pour pouvoir saisir la recherche et d'un bouton qui, lorsqu'il sera cliqué, devra déclencher une navigation vers l'URL de la page de recherche, URL à laquelle il faudra, grâce à un peu de **`JavaScript`**, concaténer la valeur du champ **`<input>`**.
Cette URL devra en partie être forgée en Twig avec la fonction **`path`** *(et c'est à ce moment que, Eurêka !, vous devez comprendre pourquoi le paramètre recherche de la route doit avoir une valeur par défaut à chaîne vide).*

3. Réfléchissez au meilleur endroit possible pour écrire votre code JS qui est censé se trouver sur toutes les pages de votre site.

4. Pour que tout fonctionne sans problème, même quand la chaîne de recherche contient des caractères spéciaux qui ont un sens particulier dans une URL, il faudra :

- Côté **`JS`** : encoder la chaîne de recherche, avant de forger l'URL, en utilisant la fonction **`encodeURIComponent`**

- Côté fonction **`chercher`** en **`PHP`** : décoder la chaîne de recherche reçue en utilisant la fonction **`urldecode`**