Laboratoire: i18n Traduction des models

# Objectif(s)

* Internationalisation i18n
  + Globalisation
  + Localisation
  + Validation
* Traduction: Models

# Créer la branche de la séance

## Création du Repository et de la branche de la séance

1. Téléchargez les fichiers du laboratoire de la séance **S07\_i18n\_Models**
2. Dans Gitkraken, clonez votre repository GitHub **A21\_3W6\_Labos\_NOM\_PRENOM**

Vous avez le choix entre partir de votre branche : **S05\_MultiPrj\_RepoPatterns** OU de la copie de départ de l’enseignant

NOTE : la version PROF contient des TODOs

1. Cliquez sur la branche de la séance: **S05\_MultiPrj\_RepoPatterns** OU **Main**
2. Créez la branche de la séance: **S07\_i18n\_Models**
3. Modifiez la connexion dans Appsettings.json : le nom du serveur
4. Faitez un Update-database pour générer la BD

# Configuration de l’internationalisation

## Installer les packages et injecter les dépendances

1. TODO 01 : Dans les projets MVC et Models, installez les parckages suivants :

* Microsoft.AspNetCore.Mvc.Localization
* Localization.AspNetCore.TagHelpers

1. TODO 02 : Ajoutez dans StartUp.cs les cultures supportées comme propriété de la classe

|  |
| --- |
| private CultureInfo[] supportedCultures = new[]              {                  new CultureInfo("en-US"),                  new CultureInfo("fr-CA")              }; |

1. Après on configure dans **Startup :**
   1. TODO 03 : Dans **ConfigureServices**, AU DÉBUT, avant le service de connexion à la BD

public void ConfigureServices(IServiceCollection services)

   {

     // TODO 03: Configurer le service AVEC AddControllersWithViews

**#region Localization**

      services.AddLocalization(options => options.ResourcesPath = "Resources");

      services.AddControllersWithViews().AddRazorRuntimeCompilation()

              .AddViewLocalization(LanguageViewLocationExpanderFormat.Suffix)

              .AddDataAnnotationsLocalization();

      /\*.AddDataAnnotationsLocalization(options => {

             options.DataAnnotationLocalizerProvider = (type, factory) =>

                 factory.Create(typeof(SharedResource));

             }); \*/

      services.Configure<RequestLocalizationOptions>(options =>

      {

        options.DefaultRequestCulture = new RequestCulture(culture: "en-US", uiCulture: "en-US");

        options.SupportedCultures = supportedCultures;

        options.SupportedUICultures = supportedCultures;

      });

**#endregion**

* 1. TODO 04 : Dans **Configure**, AU DÉBUT

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

    {

      // TODO 04:Récupération des options de localisation

      var locOptions = app.ApplicationServices.GetService<IOptions<RequestLocalizationOptions>>();

      app.UseRequestLocalization(locOptions.Value);

      if (env.IsDevelopment())

      {

        app.UseDeveloperExceptionPage();

      }

…

1. Dans la View Shared, **\_ViewImports.cshtml**, ajoutez les TagHelpers de localisation et injectez le service qui permet d’utiliser les ressources locales et partagées (Locals/shared Resources) pour afficher la éléments avec la bonne langue

@using ZombieParty

@using ZombieParty\_Models

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers.Localization

@using Microsoft.AspNetCore.Mvc.Localization

@inject IViewLocalizer Locals

@inject IHtmlLocalizer<SharedResource> SharedLocals

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** CONFIG i18n
   * **Description**: Configuration de l’internationalisation
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Ajouter un commutateur de langue

## Créer la PartialView pour le communateur de langue

1. Dans le projet MVC, dans Shared, créez une nouvelle Partial View appelée \_SelectLanguage (le \_ indique que c’est une vue partielle)

@using Microsoft.AspNetCore.Builder

@using Microsoft.AspNetCore.Mvc.Localization

@using Microsoft.Extensions.Options

@inject IOptions<RequestLocalizationOptions> LocOptions

@{

var requestCulture = Context.Features.Get<Microsoft.AspNetCore.Localization.IRequestCultureFeature>();

var cultureItems = LocOptions.Value.SupportedUICultures

.Select(c => new SelectListItem { Value = c.Name, Text = c.NativeName })

.ToList();

}

<div style="display: inline-block; vertical-align: middle;">

<form id="selectLanguage" **asp-controller**="Home"

**asp-action**="SetLanguage" **asp-route-returnUrl**="@Context.Request.Path"

method="post" class="form-horizontal" role="form">

<div class="form-group" style="margin-bottom:0">

<select onchange="this.form.submit()" **name**="culture" class="form-control btn btn-dark"

**asp-for**="@requestCulture.RequestCulture.UICulture.Name" **asp-items**="cultureItems">

</select>

</div>

</form>

</div>

## Insérer la vue dans le menu du Layout

1. Dans le \_Layout, (projet MVC, Shared), ajoutez la Partial View du commutateur de langue dans la NavBar, à la fin pour le positionner à droite de l’écran

</ul>

<div class="col-md-2 text-right">

@await Html.PartialAsync("\_SelectLanguage")

</div>

</div>

</div>

</nav>

## Insérer le cookie dans le controller Home

1. TODO 05: Dans HomeController, ajoutez l’action/méthode **SetLanguage** suivante :

[HttpPost]

   public IActionResult SetLanguage(string culture, string returnUrl)

   {

     var cookie = CookieRequestCultureProvider.MakeCookieValue(new RequestCulture(culture));

     var name = CookieRequestCultureProvider.DefaultCookieName;

     Response.Cookies.Append(name, cookie, new CookieOptions

     {

       Path = "/",

       Expires = DateTimeOffset.UtcNow.AddYears(1),

     });

     return LocalRedirect(returnUrl);

   }

1. Testez votre commutateur de langue et votre cookie.
   * Démarrer l’application
   * Changer de langue (défaut US)
   * Fermer et redémarrer : la langue est-elle changée?

# Configuration dans le HomeCOntroller

## Configurer la localisation (localizer)

1. TODO 06 : Dans HomeController, Injectez les ressources localizer locals et/ou shared

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

//TODO 06: Injection des resources localizer locals / shared

private readonly IStringLocalizer<HomeController> \_localizer;

//private readonly IStringLocalizer<SharedResource> \_sharedLocalizer;

public HomeController(ILogger<HomeController>

logger, IStringLocalizer<HomeController> localizer)

//,IStringLocalizer<SharedResource> sharedLocalizer)

{

\_logger = logger;

\_localizer = localizer;

//\_sharedLocalizer = sharedLocalizer;

}

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Création Commutateur de langue i18n
   * **Description**: Création de la PartialView pour la sélection de la langue et l’enregistrement du Cookie dans le HomeController
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Traduire les classes du modèles

## Modifier les classes et ajouter les ressources

1. Créez un dossier **Resources** dans le projet Models

NOTE : Des tableaux avec certaines traductions proposées sont disponibles à partir de la page suivante, vous devez ajoutez les manquantes, pour les messages d’erreur entre autres.

### Pour chaque classe du modèle :

1. Ajoutez des annotations **Display** pour chaque propriété affichée (inutile de faire les clés)

**Exemple :**

[Display(Name = "First Name")]

[Required]

[MaxLength(30)]

public string FirstName { get; set; }

1. Ajoutez des annotations **ErrorMessage** avec des clés normées pour les annotations de validation

**Exemple :**

[Display(Name = "Hunting Description")]

[Required(AllowEmptyStrings = false, ErrorMessage = "RequiredValidation")]

public string Description { get; set; }

1. Créez un fichier de type **Resource** dans le dossier **Resources** : nommé **Models.NomClasse.langue.resx,** pour chaque langue sélectionnée.
2. Traduisez les éléments du modèle (propriétés et messages d’erreur)

Utilisez les messages paramétrés :

**Exemple :**

**Annotations**

[Range(1, 10, ErrorMessage = "RangeValidation")]

**Traduction Resource**

RangeValidation = » {0} must be between {1} and {2}.

### Tableaux de traduction

**Category**

|  |  |  |
| --- | --- | --- |
| **Clé** | **En** | **Fr** |
| CategoryName | Category | Catégorie |
| Category | Category | Catégorie |

**ForceLevel**

|  |  |  |
| --- | --- | --- |
| **Clé** | **En** | **Fr** |
| ForceLevelNiv | Force Level | Niveau de force |
| ForceLevel | Force Level | Niveau de force |

**Hunter**

|  |  |  |
| --- | --- | --- |
| **Clé** | **En** | **Fr** |
| Nickname | Nickname | Surnom |
| Biography | Biography | Biographie |
| Hunter | Hunter | Chasseur |

**HuntingLog**

|  |  |  |
| --- | --- | --- |
| **Clé** | **En** | **Fr** |
| Nickname | Force Level | Niveau de force |
| Biography | Biography | Biographie |
| HuntingLog | Hunting Log | Carnet de chasse |

**Weapon**

|  |  |  |
| --- | --- | --- |
| **Clé** | **En** | **Fr** |
| TypeName | Type | Type |
| Description | Description | Description |
| Weapon | Weapon | Arme |

**Zombie**

|  |  |  |
| --- | --- | --- |
| **Clé** | **En** | **Fr** |
| Name | Name | Nom |
| Point | Point | Point |
| ShortDesc | Summary | Résumé |
| ZombieType | Type | Type |

**ZombieType**

|  |  |  |
| --- | --- | --- |
| **Clé** | **En** | **Fr** |
| TypeName | Name | Nom |
| ZombieType | Type | Type |

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT i18n Traduction models
   * **Description**: Création des ressources pour les classes du modèle
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

## Tester dans Index

<th>

@Html.DisplayNameFor(model => model.NomPropriete)

</th>