Tutoriel phase 4

But

- Faire un site web
- Contacter l'API précédemment créé
- Gérer l'asynchronisme

Création du projet asp.net core MVC

ASP.NET MVC permet de faire un site web reposant sur le design pattern MVC. Le Controller récupère les actions utilisateurs. Il fait les traitements métiers puis charge les données dans une classe « Model » puis les renvoie dans la vue. Cette dernière gère toute la partie visuelle de votre site web.
Une méthode d'un controller correspond au minimum à une vue. Les vues sont rassemblées par Controllers. Chaque méthode retourne un IActionResult pour indiquer qu'il affiche une vue.

Pour aller plus loin dans cette partie n'hésitez pas à consulter ce lien : https://dotnet.microsoft.com/apps/aspnet/mvc

Pour appeler une API cliente il faut utiliser un HttpClient comme décrit dans ce lien https://docs.microsoft.com/fr-fr/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client

Tout d'abord, la commande dotnet new mvc --name nomDeVotreProjet va créer un projet ASP.NET Core MVC en utilisant les templates razor (présence du dossier controllers, Models et Views). En faisant la commande dotnet new --help, on a la possibilité de créer plusieurs types de projets :

- React
- Angular

Ainsi vous pouvez utiliser mixer l'utilisation d'ASP.NET Core MVC avec la puissance de ces frameworks. Cependant dans ce workshop on partira sur une application mvc simple.
Le fichier _ViewImports.cshtml permet de partager des namespace entre toutes les vues. Le fichier _ViewStart.cshtml permet de partager du code entre toutes les vues comme définir leur layout par exemple.

En lançant le debug on arrive à une page par défaut se basant sur le Controller HomeController et les vues du sous-dossier Home.

Votre site WEB aura pour charge :

- D'utiliser l'api précédemment développé
- D'afficher une interface graphique web
- De pouvoir :

- Lister tous les livres
- Ajouter un livre
- Supprimer un livre de la bibliothèque

Pour initialiser le projet il suffit donc de faire un simple git clone et d'aller sur la branche InitProject :

- git clone https://github.com/plouiserre/DevoxxBooksFront.git
- le code d'initialisation de l'application se trouve dans la branche master

Correction

Ce projet étant moins long que celui d'avant, il va être réalisé en une seule étape.

Mise en place du Front pour la gestion des livres

Le projet créé précédemment devient le back pour cette application.

Récupérer directement la solution

La solution se trouve dans la brancheApi. Pour récupérer le code il faut taper la commande suivante dans le répertoire de votre solution:

- git checkout branchApi

Développement de cette partie

Création du modèle de donnée

Sur le dossier « Models » créez une nouvelle classe « BookModel ». Son rôle va de récupérer les données issues de votre api reçus en JSON. Pour faciliter la désérialisation, l'attribut « JsonProperty » venant de newtonsoft est utilisé sur chaque propriété.

```
using System.Collections.Generic;
using Newtonsoft.Json;

namespace DevoxxBooksFront.Models
{
    4 references
    public class BookModel
    {
        [JsonProperty("bookId")]
        0 references
        public int BookId { get; set; }

        [JsonProperty("bookTitle")]
        0 references
        public string BookTitle { get; set; }

        [JsonProperty("bookType")]
        0 references
        public string BookType { get; set; }

        [JsonProperty("bookPublication")]
        0 references
        public int BookPublication { get; set; }

        [JsonProperty("autorName")]
        0 references
        public string AuthorName { get; set; }
    }
}
```

Création du controller

Dans le dossier « Controllers », la classe BookController deviendra le controller principal de notre application.

Cette classe va permettre de récupérer les livres à afficher en contactant l'api webapi, d'en ajouter et d'en supprimer.

Avant toute chose, il faut le code pour contacter l'api. Pour cela deux attributs vont être créé :

- une constante stockant l'url de votre api
- un objet HttpClient qui aura la responsabilité d'appeler votre API

```
4 references
private const string URL = "http://localhost:5000";
5 references
private HttpClient client = null;
```

Ensuite une méthode privée doit être créé pour initialiser le HttpClient et être utilisée à chaque fois qu'on contacte le backend.

```
3 references
private void ManageClientHttp(){
    client = new HttpClient()
    {
        BaseAddress = new System.Uri(URL)
    };
    client.DefaultRequestHeaders.Accept.Add(
    new System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"));
}
```

La méthode IActionResult Index() va être appelé quand on tombera sur la page principale de notre application. Dans cette méthode, il faut appeler l'api pour récupérer la liste des livres puis la transmettre à la vue.

Pour la récupérer une requête « Get » est nécessaire. Comme l'asynchronisme est utilisée, la méthode GetAsync réalisera la requête « Get ». Une variable string stocke la réponse récupérée grâce à la méthode ReadAsStringAsync. Pour finir on désérialise la variable string dans une liste de BookModel avec la méthode DeserializeObject de l'objet JsonConvert provenant de Newtonsoft. Pour finir la méthode retourne l'objet View avec la liste des BookModel en paramètre.

```
public async Task<IActionResult> Index(){
    List<BookModel> books = new List<BookModel>();
    ManageClientHttp();
    string uri = URL+ "/api/Books/GetAll/";
    using(var result = await client.GetAsync(uri))
    {
        string response = await result.Content.ReadAsStringAsync();
        books = JsonConvert.DeserializeObject<List<BookModel>>(response) ;
    }
    return View(books);
}
```

Pour aller dans la page de création de livre, une simple méthode retournant un View est nécessaire.

```
public IActionResult Create(){
    return View();
}
```

Pour créer un livre, l'application web doit :

- Faites une requête POST grâce à l'attribut [HttpPost] en transmettant le contenu du formulaire récupéré (gérer automatiquement par asp.net mvc) dans un objet BookModel passé en paramètre de la méthode Create
- Sérialisez l'objet BookModel avec la méthode SerializeObject de la classe JsonConvert.
- Faites un post en asynchrone à notre backend grâce à la méthode PostAsync de HttpClient.
- Si tout se passe bien, on se redirige vers la page listant tous les livres grâce à la méthode RedirectToAction
- S'il y a une erreur on retourne sur la même page.

```
[HttpPost]
0 references
public async Task<IActionResult> Create(BookModel book){
    ManageClientHttp();
    string contentJson = JsonConvert.SerializeObject(book);
    string uri = URL + "/api/Books/Create";
    HttpResponseMessage response = await client.PostAsync(uri, new StringContent(contentJson, Encoding.UTF8,"application/json"));
     if (response.IsSuccessStatusCode)
        return RedirectToAction("Index");
    else
        return View();
}
```

Pour supprimer un livre, une astuce pas forcément super sexy a été choisi. Quand on arrive sur la page de suppression, on contacte l'api pour supprimer un livre puis on refait une redirection dans la page listant les livres. **Ne pas le faire en production.**

```
public async Task<IActionResult> Delete(int bookId){
    ManageClientHttp();
    string uri = URL+"/api/Books/Delete?bookId="+bookId;
    HttpResponseMessage response = await client.DeleteAsync(uri);
    return RedirectToAction("Index");
}
```

Mise en place des vues

Dans le dossier Books on va créer deux vues en .cshtml :

- Index listant tous les livres
- Create étant un formulaire pour créer un livre

La partie code doit être toujours utilisé avec un @ pour indiquer à votre compilateur qu'il doit gérer ce code C#.

Vue Index

Pour indiquer ce qu'attend la vue comme modèle de donnée il faut utiliser **@model** suivie du type de votre modèle. Dans le cas présent on veut une liste de **BookModel**.

Pour lister la liste des livres on va utiliser @foreach pour faire une boucle et afficher un par un la liste de livres:
@foreach(var cm in @Model)
{

}
Dans la boucle pour faire référence à une propriété de votre objet, cela doit être fait sous cette forme @cm.NomDeLaPropriété.

Pour faire un lien qui va contacter une méthode de votre controller précédemment développée il faut utiliser la méthode ActionLink de @Html sous cette forme : @Html.ActionLink.

Pour styliser un minimum la page, bootstrap est utilisé. Voici le code de la page

```
@{
    ViewData["Title"] = "Home Page";
}
@model List<BookModel>

<h1>Listes des livres en stock</h1>
<div>
    @foreach(var cm in @Model) {
        <div class="row">
            <div class="col-md-12">
                <h2> @cm.BookTitle </h2>
            </div>
        </div>
        <div class="row">
            <div class="col-md-6">
                Id du livre @cm.BookId
            </div>
            <div class="col-md-6">
                Type du livre @cm.BookType
            </div>
        </div>

        <div class="row">
            <div class="col-md-6">
                Date de parution @cm.BookPublication
            </div>
            <div class="col-md-6">
            Nom de l'auteur @cm.AuthorName
            </div>
        </div>

        @Html.ActionLink("Suppression du livre", "Delete","Book", new { bookId = cm.BookId })
    }
</div>
@Html.ActionLink("Ajout d'un nouveau livre", "Create", "Book")
```

Create

Le model de cette vue est BookModel.
Pour faire un formulaire, BeginForm de Html doit être utilisé sous cette forme :

```
@using(Html.BeginForm())
{
        Votre code de votre formulaire
}
```

Pour un libellé de votre formulaire, on utilise la méthode LabelFor sous cette forme : @Html.LabelFor(model=> model.PropertyOfModel).

Pour afficher une textbox, et permettre au framework de faire le lien entre la donnée saisie et l'objet bookModel il faut utiliser TextBoxFor de Html sous cette forme : @Html.TextBoxForm(model=>model.PropertyOfModel).

A la fin pour faire un submit de votre formulaire et ainsi permettre à la méthode Create en HttpPost de votre controller, il faut mettre un submit en Html comme celui-ci

```
<input type="submit" value="Ajouter un livre"/>
```

Voilà à quoi ressemble votre vue

```
@model BookModel
@using(Html.BeginForm())
{
    <fieldset>
        <legend>Livre</legend>

    <div class="editor-label">
        @Html.LabelFor(model => model.BookTitle)
    </div>

    <div class="editor-field">
        @Html.TextBoxFor(model => model.BookTitle)
    </div>

    <div class="editor-label">
        @Html.LabelFor(model => model.BookType)
    </div>

    <div class="editor-field">
        @Html.TextBoxFor(model => model.BookType)
    </div>

    <div class="editor-label">
        @Html.LabelFor(model => model.AuthorName)
    </div>

    <div class="editor-field">
        @Html.TextBoxFor(model => model.AuthorName)
    </div>

    <div class="editor-label">
        @Html.LabelFor(model => model.BookPublication)
    </div>

    <div class="editor-field">
        @Html.TextBoxFor(model => model.BookPublication)
    </div>

    <input type="submit" value="Ajouter un livre"/>


    </fieldset>

}
```

Dans la page Startup.cs il faut modifier la méthode Configure. Dans la partie UseMvc remplacer {controller=Home} par {controller=Book}. Ainsi la page Index de la partie Book sera chargée au chargement de l'application.