Rapport de projet

Par Ophélie Cluzel

Partie 3 : Application Web ASP.net modèle MVC

I. Description (cahier des charges)

Le but de cette partie était d'implémenter notre gestionnaire d'équipements du jeu Fortnite sous forme d'une application Web. J'ai choisi de travailler sur les équipements de type pioche. J'ai donc implémenté différentes fonctionnalités de recherche (par nom, par rareté), d'affichage et d'ajout d'équipement dans mon application. Pour faire cela j'ai réutilisé le travail réalisé sur l'agenda de contacts fait en TP.

II. Analyse fonctionnelle

1. Modification de la vue partagée : _Layout.cshtml

Je commence par modifier le texte de la barre de navigation, pour cela il me suffit de modifier le premier champ de la ligne suivante (ici je l'ai modifié par CDAA Application) :

```
@Html.ActionLink("CDAA Application", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
```

Ensuite je vais ajouter mes différentes options (Afficher,Ajouter,Rechercher..) dans la barre de navigation, pour cela il me suffit d'ajouter autant de balises telles que celle-ci-dessous que d'options (ici j'en ai utilisé 7) :

```
<span class="icon-bar"></span>
```

Ensuite je dois également ajouter les actions associées à mes options pour cela je dois ajouter la balise suivante pour chaque option :

```
<li>@Html.ActionLink("Accueil", "Index", "Home")</li>
```

Elle comporte tout d'abord le nom qui sera affiché pour l'option puis le nom de la méthode associée dans le Controller et enfin le nom du Controller.

Comme les Controllers pour mes nouvelles options Afficher, Ajouter, Rechercher par nom et Rechercher par rareté n'existent pas je vais devoir les créer : Pour cela il suffit de faire un clic droit sur le dossier Controllers du projet et de cliquer sur ajouter un nouveau Controller. J'ai donc créé 4 nouveaux Controllers : AfiicherPiochesController.cs, AjouterPiocheController.cs, RechercherPiocheController.cs et RechercherRareteController.cs. On n'oublie pas le post-fixe Controller qui est obligatoire. La création de nouveaux Controllers va générer automatiquement la création de dossiers associés dans le dossier des vues (Views). Nous verrons plus tard en détail le contenu de ces classes Controller.

Ensuite je modifie également le pied de page et le titre de page (qui seront communs à toutes les pages de l'application) pour cela je modifie le texte dans les balises footer et title :

```
<footer>
    <p>&copy; @DateTime.Now.Year - CDAA Application</p>
</footer>
```

```
<title>@ViewBag.Title - CDAA Application </title>
```

2. Modification de la page d'accueil (Index.cshtml)

Je commence par modifier le nom de la page en modifiant le texte de la balise suivante :

```
@{
    ViewBag.Title = "Page d'accueil";
}
```

Puis je modifie la balise h1 (plus grand titre dans la page) mais au lieu de mettre du texte je décide de mettre une image. Pour cela il me suffit de mettre une balise img en lui spécifiant la source de l'image.

```
<h1><img src="~/images/fortnite.jpg" /></h1>
```

Ensuite je modifie les différents champs de texte (sous titres et champs au-dessus des boutons) etc. et pour finir je modifie les actions faites lors du clic sur un bouton pour chacun de mes boutons (dans le Url.Action je mets le nom de la méthode dans le Controller suivi du Controller associé et dans le texte de la balise a je mets le texte qui sera affiché sur le bouton) :

```
<p><a class="btn btn-default" href=@Url.Action("AjouterPioche", "AjouterPioche")>Ajouter</a></p>
```

Pour finir je modifie également l'action générée lors du clic sur le bouton « en savoir plus » en modifiant le href par l'url du site Fortnite.

```
<p><a href="https://www.epicgames.com/fortnite/fr/home" class="btn btn-primary btn-lg">En savoir plus &raquo;</a></p>
```

3. Modèle de données

Je commence par modifier ma classe Equipement en ajoutant des annotations au-dessus de chaque propriété sur ces attributs. Elles seront utiles pour l'affichage des formulaires concernant un équipement (le texte associé à Name sera le message d'invite dans le formulaire):

```
[Display(Name = "Nom :")]
19 références
public string Nom
{
    get { return nom; }
    set { nom = value; }
}
```

Je modifie également l'attribut Photo de ma classe équipement en modifiant son type en string car je n'arrivais pas à gérer le type Image dans l'application. Ainsi l'attribut Photo contiendra maintenant le nom du fichier associé à l'image.

Pour finir il faut modifier la classe ManagerFBR pour la transformer en singleton. Pour cela on commence par lui créer un nouvel attribut statique Instance de type ManagerFBR :

```
private static ManagerFBR instance = null;
```

Ensuite on lui crée une propriété en lecture associée, si l'instance n'a jamais été crée on crée une nouvelle instance de ManagerFBR sinon on réutilise celle qui existe déjà. Cela va nous permettre de travailler toujours sur la même instance.

```
public static ManagerFBR Instance
{ get { if (instance == null)
            instance = new ManagerFBR();
        return instance; } }
```

4. Mise en place de l'option AfficherPioches qui affiche toutes les pioches

Je commence par modifier le code de ma classe AfficherPiochesController crée plus tôt.

Dans celle-ci il y a déjà une méthode générée automatiquement qui est la méthode qui sera appelée par défaut si jamais il y a un problème : celle-ci va appeler la vue correspondante, ici c'est la vue de la page d'accueil.

Je dois maintenant créer la méthode AfficherPioche qui sera appelée lors du clic sur le bouton ou l'option dans la barre de navigation associé. Pour cela on commence par récupérer l'instance du ManagerFBR puis on fait appel à la vue associée en lui passant en paramètre la liste des pioches qui devient le modèle pour cette vue.

```
public ActionResult AfficherPioches()
{
    ManagerFBR mg= ManagerFBR.Instance;
    return View("AfficherPioches", mg.Lpi.Liste);
}
```

Maintenant on doit donc créer la vue associée pour cela il nous suffit de faire un clic droit sur le dossier généré automatiquement dans le dossier Views et d'ajouter une nouvelle vue. Dans cette nouvelle classe on modifie les différents champs de texte (titre de la page etc.) ensuite on parcourt la liste des pioches en faisant appel à Model (paramètre fourni par le Controller) :

```
@foreach (GestionPioches.Pioche pi in Model)
```

Pour chaque attribut de notre objet pioche on affiche dans une cellule de tableau le label associé puis la valeur associée (grâce à l'annotation display que l'on a ajoutée plus tôt) :

```
<td width="200">@Html.LabelFor(_ => pi.Nom)@Html.DisplayFor(_ => pi.Nom) </td>
```

5. Mise en place de l'option RechercherPioche qui permet de rechercher une pioche par son nom

Je commence par modifier ma classe RechercherPiocheController crée plus tôt. Je crée donc la méthode RechercherPioche. Dans celle-ci je commence par récupérer l'instance de ManagerFBR puis on va initialiser un code d'erreur qui sera à 0 si il n'y a pas de problème et à 1 si la pioche recherchée n'existe pas par défaut on lui donne la valeur 0.

```
ViewBag.error = "0";
```

On crée également un nouvel objet pioche que l'on initialise à null. Ensuite on récupère le nom de la pioche recherchée en récupérant la valeur de la textBox intitulée nomRecherche (que l'on verra plus tard dans la vue associée).

```
String nom = Request.QueryString["nomRecherche"];
```

Si la valeur de cette chaine de caractères récupérée n'est pas nulle, on recherche la pioche associée à ce nom dans la liste des pioches grâce à la méthode GetByNom() de la classe ManagerFBR. Si aucune pioche n'est trouvée (si pioche=null) alors on définit le ViewBag.error à 1.

A la fin on appelle la vue associée avec comme Model la pioche que l'on vient de récupérer.

On crée donc la vue associée et on modifie les différents champs de texte de la page.

On commence par indiquer que l'on va faire appel à la méthode RechercherPioche du Controller associé grâce à la ligne suivante :

```
@using (Html.BeginForm("RechercherPioche", "RechercherPioche", FormMethod.Get))
```

Puis on crée une textBox intitulée NomRecherche et un bouton rechercher de type submit qui fera donc appel à la méthode du Controller.

```
@Html.TextBox("nomRecherche")
<input type="submit" value="Rechercher" />
```

Une fois cela fait on vérifie si le Controller a trouvé une pioche (donc si ViewBag=0), on récupère la pioche passée dans le paramètre Model et on l'affiche tout simplement en utilisant labelFor et displayFor pour chacun de ces attributs. Sinon on affiche un message d'erreur comme quoi il n'existe pas de pioche à ce nom.

6. Mise en place de l'option AjouterPioche qui permet d'ajouter une pioche à la collection des pioches du joueur

Je commence par modifier le code de ma classe AjouterPiocheController crée plus tôt. Cette fois-ci je vais devoir créer deux méthodes une méthode avec l'annotation `[HttpGet]` qui va faire appel à la vue associée qui aura comme modèle une nouvelle pioche qu'il va falloir remplir :

```
[HttpGet]
0 références
public ActionResult AjouterPioche()
{
    return View("AjouterPioche", new Pioche());
}
```

Et une méthode avec l'annotation `[HttpPost]` qui récupère la pioche en paramètre et va l'ajouter au ManagerFBR. Pour finir elle fait appel à la page AfficherPioche.

```
[HttpPost]
0 références
public ActionResult AjouterPioche(Pioche p)
{
    ManagerFBR mg = ManagerFBR.Instance; //singleton
    mg.AjoutPioche(p);
    return RedirectToAction("AfficherPioches", "AfficherPioches", "AfficherPioches");
}
```

Ensuite on va s'intéresser à la vue AjouterPioche.cshtml on indique que son Model est de type pioche et que l'on utilise la méthode AjouterPioche du Controller associé. Pour chaque attribut on affiche son label et une zone d'édition associée.

```
<p>@Html.LabelFor(p => p.Nom)@Html.EditorFor(p => p.Nom)</p>
```

Pour la rareté on crée une liste déroulante dans laquelle on pourra choisir une valeur directement avec les noms contenus dans categorieRarete.

```
@Html.DropDownListFor(p => p.RareteS, new SelectList(Enum.GetNames(typeof(GestionPioches.categorieRarete))))
```

Enfin pour finir on crée un bouton valider de type Submit qui lorsqu'il sera cliqué, lancera un appel à la méthode post du Controller.

7. Mise en place de l'option RechercherRarete qui permet de rechercher la liste des pioches d'une rareté donnée

Je commence par modifier le code de ma classe RechercherRareteController crée plus tôt. Je crée donc la méthode RechercherRarete dans laquelle je récupère l'instance de ManagerFBR. Puis je défini un code d'erreur par défaut à 0 qui sera égal à 1 si aucune pioche de la rareté recherchée n'est trouvée et je crée une nouvelle instance de liste de pioches. On récupère ensuite la valeur de la DropDownList intitulée rareteRecherche, si cette valeur n'est pas nulle on va faire appel à la méthode GetByRarete de la classe ManagerFBR pour récupérer les pioches correspondantes à cette rareté. Si aucune pioche n'est trouvée (donc que le nombre d'éléments dans la liste est égal à 0) on définit le code erreur à 1.

Pour finir on fait appel à la vue associée avec en paramètre la liste de pioche qui sera le Model.

Je dois maintenant créer la vue associée. Je commence par indiquer que l'on fera appel au Controller associé (une fois le bouton validé cliqué).

```
@using (Html.BeginForm("RechercherRarete", "RechercherRarete", FormMethod.Get))
```

Je crée ensuite une dropdownlist intitulée rareteRecherche contenant les valeurs de categorieRarete et un bouton valider de type submit.

```
@Html.DropDownList("rareteRecherche",new SelectList(Enum.GetNames(typeof(GestionPioches.categorieRarete))))
<input type="submit" value="Rechercher" />
```

Puis une fois cela fait on vérifie le code erreur, s'il est égal à 0 (donc si au moins une pioche de cette rareté a été trouvée) on va afficher chaque pioche de la liste de pioche qui est dans Model. Pour cela on utilise labelFor et displayFor sur chacun des attributs de la pioche. Sinon on affiche un message d'erreur comme quoi il n'existe aucune pioche de cette rareté.

8. Mise en place des options Contact et A Propos de

Pour ces deux options j'ai simplement modifié des champs de texte dans les vues associées et les viewBag.message dans le HomeController.