TP2 – Projet de session

(15%)

Notez que pour que le TP2 fonctionne, vous devez avoir complété le TP1 !

Étape 1 - Transformation de votre site statique en application Web (R11-12)

Transformer le TP1 en projet ASP.NET Core. L’objectif est d’avoir 4 vues qui représenteront les 4 pages créées dans le TP1. Les données seront encore « hardcodées » dans les vues pour le moment (on ne va pas créer de modèle « Produit » pour l’instant).

* Créez un projet ASP.NET Core à partir du modèle de projet « Vide » sans HTTPS.
* Recopiez le contenu du fichier « **Program.cs** » disponible dans le matériel de la semaine 6 de manière à écraser tout ce qui était déjà présent dans « **Program.cs** ».
* Mettez vos ressources statiques dans un répertoire "wwwroot" (css/js/images)
* Vous devez créer un contrôleur « **HomeController** ».
  + Il doit avoir l’action « **Index** »
* Vous devez créer un contrôleur « **EnfantController** ».
  + Il doit avoir l’action « **Recherche**»
  + Il doit avoir l’action « **Detail** »
* Vous devez créer un contrôleur « **FavorisController** »
  + Il doit avoir l’action « **Index** » (Ça fait 2 actions avec le même nom, mais pas de soucis, elles ne sont pas dans le même contrôleur !)
* Pour l’instant, les actions des contrôleurs vont avoir comme unique ligne de code l’instruction de retourner la vue portant le même nom que l’action (ex : return View();)
* Créez un fichier « **\_Layout.cshtml** » à partir de la structure HTML que vous avez conçue dans le TP1 et faites-en sorte que les 4 vues utilisent ce layout. (Sachant qu’elles ont toutes le même header et le même footer… il ne reste que le <main> qui variera)

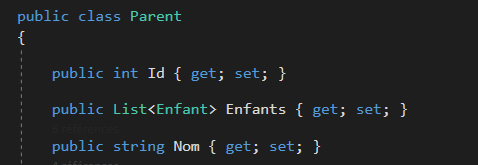
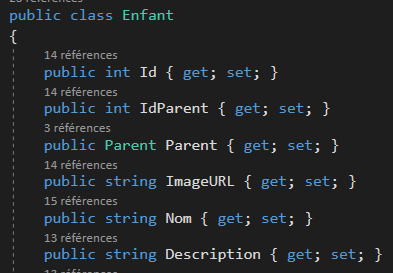
Étape 2 - Lister et Consulter les entités parents et enfants (R15 à R18)

1 - Création de classes pour le Modèle

Dans le namespace « **Models** », créez …

* Une classe qui représente vos entités **Enfant** (Avec les 7 propriétés en attributs)
  + Si vos enfants étaient représentés par des images, vous pourriez créer un attribut de type string qui correspond au nom du fichier image ! Cela sera utile dans la vue.
    - (Ex : private string ImgFile = "pikachu.png";)
* Une classe qui représente vos entités **Parent** (Avec les 3 propriétés en attributs)
* Vos **Parents** et vos **Enfants** doivent avoir un **Id unique** et **numérique**. (Numéros de 1 à 3 pour vos entités parents ... numéros de 1 à 12 pour vos entités enfants !)
* Une classe nommée « **BaseDeDonnees.cs** » ou « **Database.cs** ». Cette classe aura deux **List<>** et contiendra vos entités enfants et parents.
* Attention ! Il doit exister un ou des attributs qui permettent de faire le **lien** entre vos enfants et vos parents. Par exemple, pour les **Dresseurs**, on pouvait accéder à leurs **Pokémons** via un de leurs attributs.
  + La classe **parent** devrait avoir une ***List<>*** d’éléments « **Enfant** ».
  + La classe **enfant** devrait avoir un paramètre de type « **Parent** » et un paramètre de type int qui contient l’**Id** numérique du **parent**.





Les classes ci-dessus sont des exemples pas forcément complets

2 - Données de départ

* Dans le *constructeur* de la classe « **BaseDeDonnees.cs** » / « **Database.cs** », vous devez remplir vos deux listes d’entités en instanciant vos 3 parents et vos 12 enfants avec leurs données. Si certains de vos attributs étaient des « descriptions », oui, ça fait des gros *string*.
* En instanciant vos enfants... n’oubliez pas de leur passer en paramètre l’id numérique de leur parent ! (Pour combler le paramètre IdParent de type int)
* Dans le constructeur de votre base de données, assurez-vous également après avoir rempli la liste des éléments **enfants** et des éléments **parents** de mettre les deux boucles « foreach » comme ci-dessous (adaptez-les au besoin à votre code).



(Exemple basé sur « The Seven Deathly Sins)

La première boucle permet de remplir la propriété « Enfants » des parents en lui rattachant tous les objets qui devraient être leur enfant. La deuxième permet de remplir la propriété « Parent » des enfants en leur rattachant l’objet représentant le parent.

3 - Injection de dépendance

* Votre classe servant de base de données doit être instanciée comme un Singleton dans **Program.cs** et doit être fournie à vos contrôleurs (Home et Enfant) en paramètre.

4 - Modifier le contrôleur « **EnfantController** » et ses vues

* Les actions et vues « **Recherche** » et « **Detail** » devront être modifiées.
  + Pour **Recherche**, on doit passer à la vue une **List<>** des entités enfants. La vue devra être modifiée légèrement afin d’inclure une boucle qui intègre toutes les informations de chaque enfant.
    - Cela pourrait vous amener à devoir modifier la structure de votre page … soit vous avez une *double boucle* qui pond plusieurs **.row** et plusieurs **.col** en paquets de 3, soit vous avez une *simple boucle* et une seule **.row** dans laquelle vous pondez 12 **.col** et vous utilisez des classes de type **col-lg-4** / **col-12** pour décider combien d’enfants s’affichent par rangée.
  + Pour **Detail**, on doit passer à la vue une seule et unique **entité enfant** avec l’**id** approprié. La **vue** devra être modifiée pour afficher les informations spécifiques à l’**enfant** reçu en paramètre. Votre vue sera donc « généralisée » à tous les **enfants**.
    - Vous pouvez retirer votre *composante Bootstrap* (Collapse, caroussel, etc.) si vous le souhaitez !
    - Nous n’attaquerons pas les **formulaires** maintenant, mais vous pouvez déjà changer la **value** pour le **input type=hidden** pour qu’il correspond à l’**enfant** passé en paramètre.
  + Créez une troisième vue nommée « **NotFound.cshtml**» ou « **NonTrouve.cshtml**» qui utilisera le même **Layout** que vos autres pages, mais qui servira simplement à afficher à l’utilisateur que « *L’entité enfant* » n’a pas pu être trouvé. (Affichez le message comme bon vous semble, la page peut être comique)
    - L’action « **Detail** » retournera cette vue si l’**Id** reçu en paramètre par l’action est invalide !

5 - Modifier le contrôleur « **HomeController** » et sa vue

* L’action « **Index** » du contrôleur « **HomeController** » sert à afficher la liste des entités **parents** (normalement vous en avez 3) ainsi que des statistiques de base sur les entités **enfants** de chaque parent. (Ci-dessous, les stastiques choisies sont « Nombre de personnages » et « Moyenne niveau puissance »)

  
(Exemple basé sur « The Seven Deathly Sins)

* Servez-vous de votre classe **BaseDeDonnees / Database** dans **HomeController** pour envoyer la **List<Parent>** à la vue **Index** qui affichera vos 3 parents avec une boucle.
* Typez (@model) votre vue afin d’indiquer qu’elle puisse recevoir une **List<Parent>**
* Utilisez une **boucle** sur les données reçues afin de créer une colonne par entité parent et afficher les informations du parent (image/nom/description).
* Pour les **statistiques** concernant les enfants **utilisez LINQ** à l’intérieur de la boucle des parents sur la collection (List) de leurs enfants afin d’obtenir les statistiques souhaitées (indice : exercice dresseurs de pokémon!)

Étape 3.1 - Recherche et routage par attribut (R17 à R18)

1. **UTILISATION DES FICHIERS VIEWSTART ET VIEWIMPORT**

Ajoutez au moins un fichier ViewStart et un fichier VIewImports dans votre application de manière à ce qu’ils soient tous les deux pertinents et vous permettent d’économiser une ou plusieurs lignes de code dans certaines vues.

1. **CONSULTER UNE ENTITÉ ENFANT PAR ID**

Dans les dernières étapes, nous avions adapté l’action **Detail** du contrôleur **Enfant** afin d’afficher le détail de l’enfant. Vous avez fait en sorte que l’action Detail reçoit soit un identifiant numérique (ex. : 1, 2, 3, …) ou soit en utilisant le nom de l’entité enfant désiré. Pour cette étape, il est important que les entités enfant possèdes un identifiant numérique (ex. : 1, 2, 3, ...) puisse vous devez faire en sorte que l’entité enfant soit accessible soit à l’aide de son identifiant (qui agira comme une clé primaire) ou soit à l’aide de son nom! Renommez l’action **Detail** en **DetailParID**, et en utilisant, le **routage par attribut** sur l’action assurez-vous que les 3 **patterns de** **requête** ci-dessous soit compatibles avec cette action.

|  |  |
| --- | --- |
| CAS | EXEMPLE D’URL |
| #1 | /enfant/detail/5 |
| #2 | /enfant/5 |
| #3 | /5 |

**CONSULTER UNE ENTITÉ ENFANT PAR SON NOM**

Créez une nouvelle action nommée **DetailParNom** sur le contrôleur Enfant qui permettra de recevoir une chaine de caractères à la place d’un nombre. Le code sera très similaire au code de l’action **Detail** sauf que la recherche se fera sur le nom. Il sera nécessaire d’utiliser le **routage par attribut** afin que les 3 patterns de requête qui suivent soient compatibles avec cette nouvelle action.

|  |  |
| --- | --- |
| CAS | EXEMPLE D’URL |
| #4 | /enfant/detail/nom\_de\_l’enfant |
| #5 | /enfant/nom\_de\_l’enfant |
| #6 | /nom\_de\_l’enfant |

**PETITE REMARQUE #1 – LA CASSE**

Il est important que le code ne soit pas sensible à la casse.  
Si je me base sur l’ensemble des 7 Deathly Sins:

* /enfant/helbram
* /enfant/HelBRam
* /enfant/HELBRAM

Ces exemples devraient tous m’afficher le détail des informations du personnage helbram.   
La fonction ToUpperCase des chaînes de caractères peut être utile!

**PETITE REMARQUE #2 – ENFANT NON-TROUVÉ**

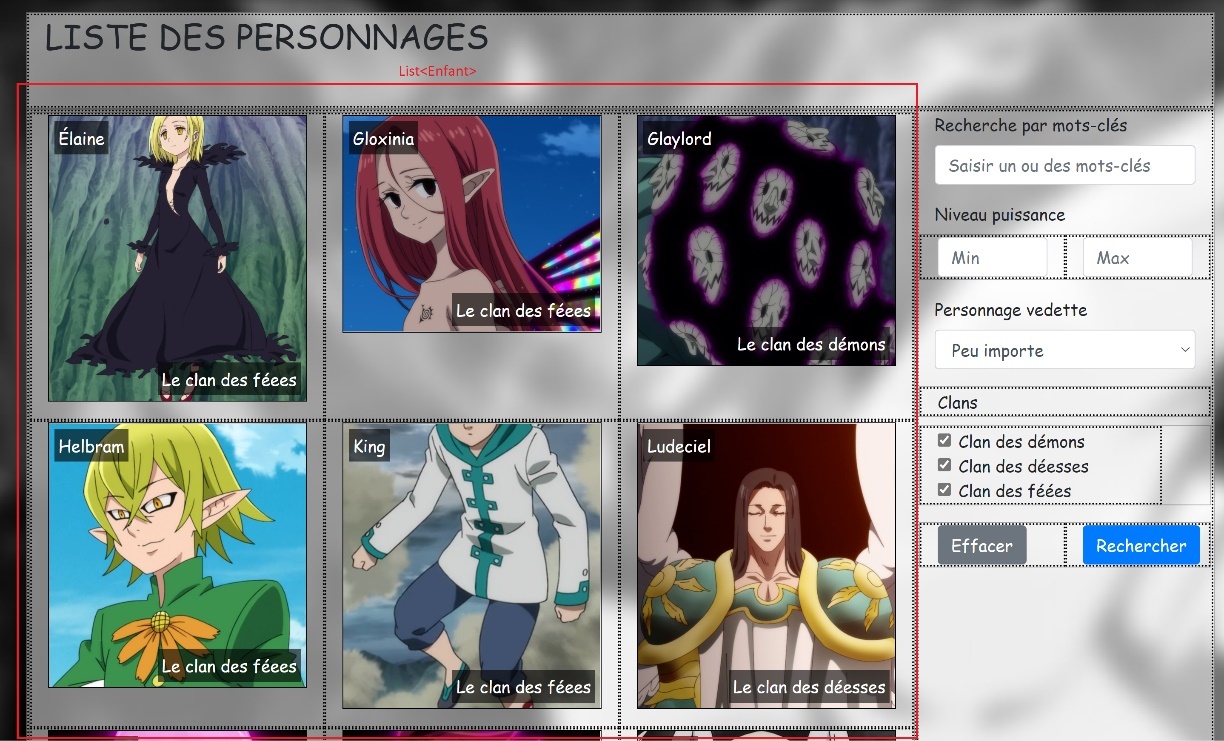
Tout comme pour l’action **Detail** du contrôleur Enfant, l’action **DetailParNom** doit aussi renvoyer la vue « **NotFound** » si l’entité enfant n’a pas été trouvée.



1. **DÉBUTER LA RECHERCHE**

**PRÉAMBULE – EXPOSITION DE LA PROBLÉMATIQUE**

Dans les dernières étapes, nous avions fait en sorte que l’action **Recherche** du contrôleur **Enfant** récupère la liste des enfants (type List<Enfant>) et la donne à la vue Recherche.cshtml afin de l’afficher.



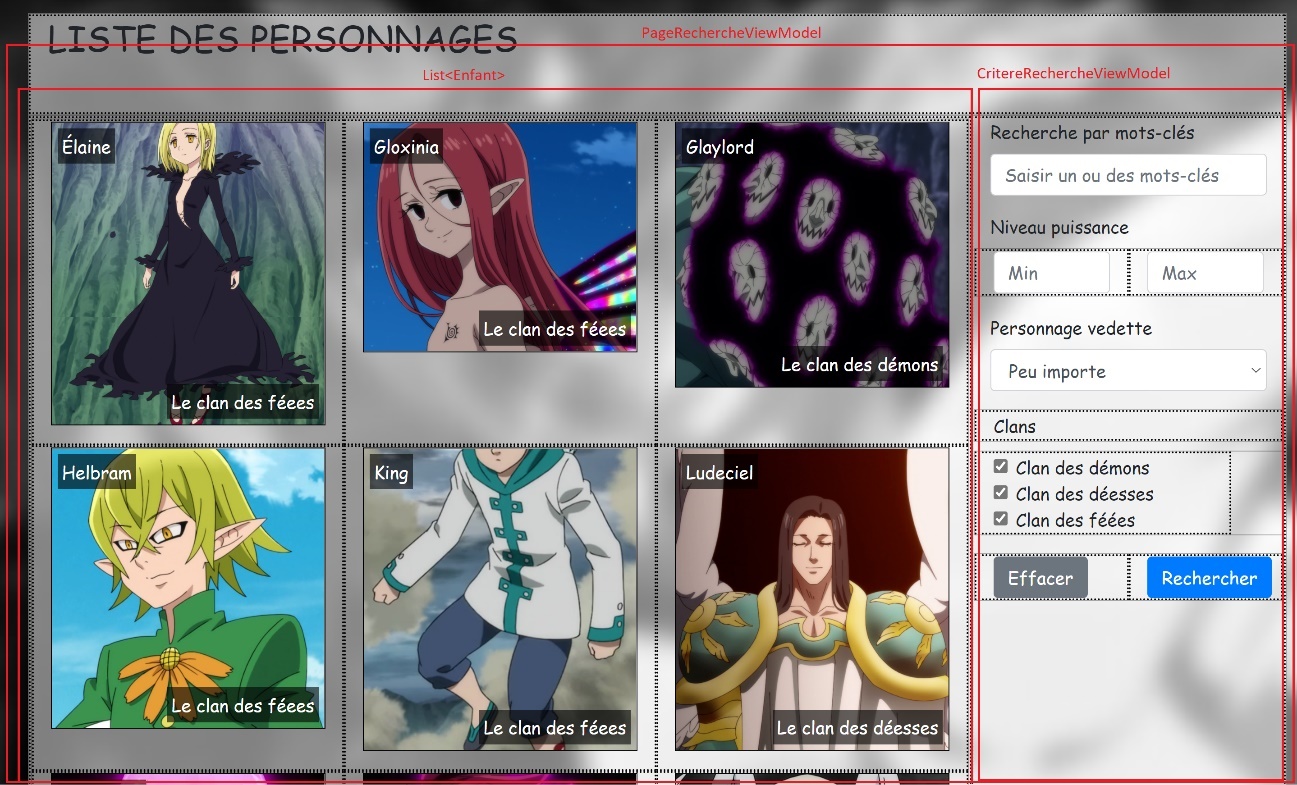
Cette stratégie était bien, cependant, la vue Enfant/Recherche.cshtml devrait recevoir plus d’information qu’uniquement une list<Enfant>, la vue devrait également recevoir les critères de recherche en plus de la list<Enfant> (sans quoi, nous ne serions jamais capables de réafficher les critères de recherche que l’utilisateur a utilisés !).

**PRÉAMBULE – EXPOSITION DE LA SOLUTION**

Étant donné que la vue a besoin de plusieurs informations de natures différentes :

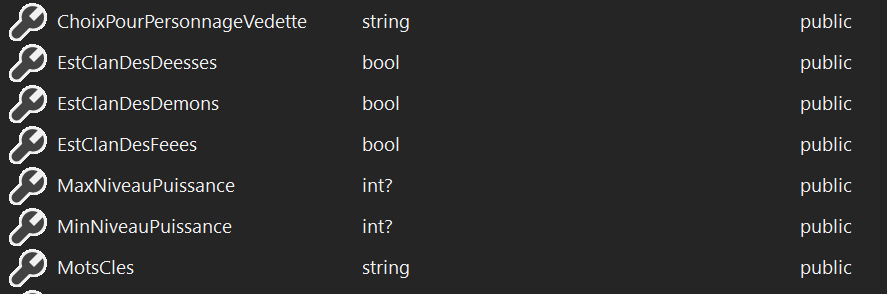
* La liste des enfants correspondant à la recherche
* Les critères de recherche

Nous pourrions créer un ViewModel nommé « PageRechercheViewModel » qui servira à représenter les données nécessaires pour la page. Cette entité aura une liste des enfants correspondant à la recherche ainsi qu’un objet représentant les critères de recherche qui avait été utilisé qu’on nommera « CritereRechercheViewModel ».

****

Dans les prochaines pages, nous allons faire le code pour mettre cette stratégie en place et rendre la recherche fonctionnelle!

**ÉTAPE #1 – CRÉER UNE STRUCTURE REPRÉSENTANT LES CRITÈRES DE RECHERCHE**

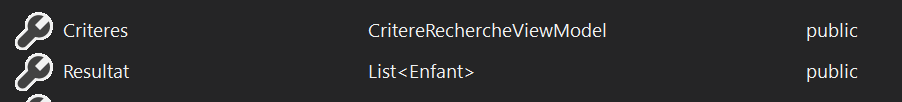
Créez une classe nommée « CritereRechercheViewModel » dans le dossier « ViewModels » qui représentera une structure de données permettant de stocker les différentes informations relatives à la section correspondant à vos critères. La structure de la classe sera différente en fonction de votre sujet et des critères que vous avez utilisés.[[1]](#footnote-2) Vous devez laisser le constructeur vide. La classe devrait uniquement définir des propriétés publiques.

Si dans votre formulaire vous avez des champs de type « Input » qui peuvent recevoir un nombre et que le nombre n’est pas obligatoire, vous devez indiquer que le type est nullable avec le symbole « ? » (Ex : int?, double?). **Vous devez également vous assurer que les attributs « name » de vos éléments HTML dans le formulaire de recherche sont identiques aux noms des propriétés de votre classe!** C’est de cette manière que ASP.NET Core va associer les informations du formulaire de recherche aux propriétés de la classe automatiquement!

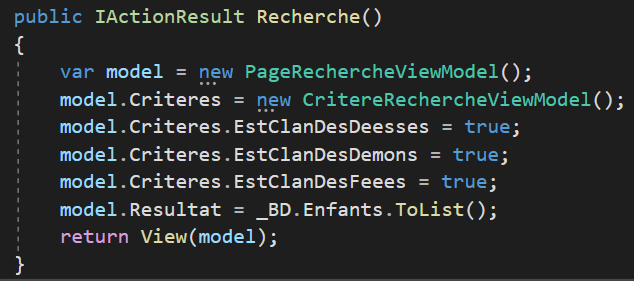

**ÉTAPE #2 – CRÉER UNE STRUCTURE REPRÉSENTANT LES INFORMATIONS DE LA VUE**

Créer la classe nommée « PageRechercheViewModel » dans le dossier « ViewModels » qui représentera les données nécessaires pour l’affichage de la vue. Vous devez laisser le constructeur vide. La classe devrait uniquement définir des propriétés publiques.



**ÉTAPE #3 – MODIFIER L’ACTION « RECHERCHE » DU CONTRÔLEUR « ENFANT »**

Nous allons maintenant modifier le code de l’action **Recherche** du contrôleur **Enfant** afin qu’il fournisse à la vue un objet de type « PageRechercheViewModel ». Modifiez le code de l’action afin qu’il ressemble au code ci-dessous :



Suite à ce changement, l’action « Recherche » ne donne plus à la vue une simple List<Enfant> représentant les résultats à la vue, mais donne désormais un objet « PageRechercheViewModel » avec à l’intérieur une liste de résultats (dans la propriété « Resultat ») et les critères de recherche dans la propriété « Critères ». Si vous tentez de compiler maintenant, vous allez obtenir des erreurs lorsque vous tentez d’accéder à la vue Recherche.cshtml étant donné que vous n’avez pas encore apporté les changements nécessaires au niveau de la vue!

**ÉTAPE #4 – AFFICHER LES DONNÉES ET LES CRITÈRES DE RECHERCHE**

Concernant la vue « Recherche.cshtml », vous devez apporter les changements suivants :

* Modifiez la vue en conséquence du fait qu’elle doit maintenant recevoir un objet de type PageRechercheViewModel (en utilisant la directive @model)
* La propriété « Model » avec un « M » majuscule représente donc maintenant votre objet de type PageRechercheViewModel. Vous pouvez donc accéder aux résultats de la recherche en écrivant @Model.Resultat et vous pouvez accéder aux critères de recherche en écrivant @Model.Criteres
* Faites-en sorte que la liste des résultats se réaffiche en utilisant @Model.Resultat pour votre boucle d’affichage des résultats.
* Faites-en sorte que les différents champs du formulaire de recherche affichent bien les valeurs contenues dans l’objet @Model.Criteres.

**CHAMPS DE TYPE INPUT**

* Comme vue avant l’intra, les valeurs pour les champs **input** se modifient avec l’attribut HTML « value ». Il suffit donc de simplement affecter la valeur d’une des propriétés des critères de recherche à l’attribut « value » afin que le champ affiche cette valeur :



**CHAMPS DE TYPE CHECKBOX**

* Comme vue avant l’intra, un champ checkbox est coché si l’attribut checked est présent sur l’élément checkbox. Une manière simple de faire est d’utiliser un « if » et dans un certain cas faire afficher un checkbox sans l’attribut checked et dans l’autre cas afficher un checkbox avec l’attribut checked. Je vous suggère également de donner à l’attribut « value » la valeur « true », ainsi lorsque la valeur sera récupérée côté serveur elle sera vue comme une valeur booléenne « true ». Dans le cas où que la case n’est pas cochée, la valeur ne sera pas envoyée et le serveur va construire l’objet CritereRechercheViewModel en laissant la valeur par défaut des booléens (qui est false).

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**CHAMPS DE TYPE SELECT**

* Appliquez le même principe qu’utiliser pour les *checkbox* à la différence que c’est l’attribut « selected » qui doit être présent sur l’option retenue. Si votre select possède 4 options différentes, vous aurez donc 4 IF/ELSE à coder! C’est pénible, mais vous allez survivre!

**TESTEZ, TESTEZ, TESTEZ!**  
Pensez à tester que ça fonctionne bien!   
**Une manière simple de tester est de modifier temporairement l’action Recherche** en modifier les valeurs des critères afin de s’assurer que ça s’affiche bien dans la vue :  


 Spécifications (non exhaustives) en lien avec les éléments de contenu évalués

|  |  |
| --- | --- |
| **ÉTAPE #1 - TRANSFORMER PARTIELLEMENT VOTRE TP1 EN PROJET ASP.NET CORE** | **Éléments de contenu évalués** |
| * Création d'un projet ASP.NET Core vide et configuration du fichier "Program.cs" | ASP.NET Configuration de base |
| * Accès aux ressources statiques (wwwroot) et s'assurer que vous êtes capable d'y accéder | ASP.NET Configuration de base |
| * Création de la page de disposition (Shared/\_Layout) afin de pouvoir l'utiliser dans vos vues | Vues Razor |
| * Création des contrôleurs (EnfantController et HomeController) et de leur vues (à cette étape-ci, les données des vues sont encore hardcodées) | Contrôleur et Vues Razor |
| **ÉTAPE #2 - LISTER ET CONSULTER LES ENTITÉS ENFANTS** |  |
| * Création des classes pour représenter le modèle (BaseDeDonnees, Parent, Enfant) | Modèles |
| * Ajuster la structure des classes "Parent" et "Enfant" afin que créer une relation entre les deux classes | Modèles |
| * Définir les données de départ pour la liste des parents et des enfants (en fonction de votre projet) | Modèles |
| * Relier les enfants au parent et inversement (avec la boucle *foreach* à placer à la fin du constructure de votre BaseDeDonnees) | Modèles |
| * Configuration de l'injection de dépendance afin que la base de données puisse être passée contrôleur (singleton) | ASP.NET Configuration de base |
| * Modifier le contrôleur "EnfantController" (afin d'afficher la liste des enfants dans la vue "Recherche") | Contrôleur |
| * Modifier le contrôleur "EnfantController" (afin d'afficher l'enfant correspondant à un certain ID) | Contrôleur |
| * Modifier la vue "Enfant/Recherche" afin d'utiliser les données transmises par le contrôleur | Vues Razor |
| * Modifier la vue "Enfant/Detail" afin d'utiliser les données qui ont été transmises par le contrôleur | Vues Razor |
| * Créer une vue pour indiquer une entité enfant inexistante (et l'utiliser si l’entité enfant qu'on désire afficher n'existe pas) | Vues Razor |
| * Modifier le contrôleur "HomeController" (afin d’afficher la liste des parents dans la vue "Index") | Contrôleur |
| * Modifier la vue "Home/Index" afin d’utiliser les données transmises par le contrôleur. | Vues Razor |
| **ÉTAPE #3 - RECHERCHE D’ENTITÉS ENFANTS ET ROUTAGE PAR ATTRIBUT** |  |
| * Création des fichiers \_ViewStart et \_ViewImports. (Layout et directives) | Vues |
| * Routage par attribut pour les ids et les noms des entités enfants avec l’action Enfant/Detail. | Contrôleurs |
| * Création d’un ViewModel pour encapsuler les critères de recherche. | Modèles |
| * Création d’un ViewModel pour encapsuler les données nécessaires à l’affichage de la vue Enfant/Recherche. | Modèles |
| * Modification de la vue Enfant/Recherche pour afficher les données du ViewModel. | Vues |

Grille d’évaluation du TP2 (100 points). Le TP2 complet comptera pour 15% de la note finale du cours.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Niveau de performance | | | | |
|  |  | Insuffisant | Faible | Suffisant | Satisfaisant | Supérieur |
| Éléments de contenu évalués | **ASP.NET Configuration de base** (15pts) :  -Création d'un projet ASP.NET Core  -Structurer le répertoire de travail adéquatement  -Configurer correctement l’accès aux ressources statiques  -Configurer correctement le fichier « Program.cs »  -Configurer un singleton de type « BaseDeDonnée » par l’injection de dépendance  -Modifier la route par défaut en conséquence de votre projet. | 0 à 5 pts  L'utilisation d'ASP.NET est **absente** ou plutôt **superficielle** et **ne démontre pas la maitrise** par l'étudiant de ces possibilités.   **Peu d'éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 7 pts  L'utilisation d'ASP.NET **démontre peu la maitrise** par l'étudiant de ces possibilités.   **La plupart des éléments principaux**  en lien avec les spécifications attendues ont été **réalisés** correctement. | 9 pts  L'utilisation d'ASP.NET **démontre une maitrise minimale** par l'étudiant de ces possibilités.   **Les principaux éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 12 pts  L'utilisation d'ASP.NET **démontre une grande maitrise** par l'étudiant de ces possibilités.   **Les principaux éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 15 pts  L'utilisation d'ASP.NET **démontre une parfaite** **maitrise** par l'étudiant de ces possibilités.    **L'ensemble** des éléments en lien avec les spécifications attendues ont été **réalisés** correctement. |
| **Contrôleur** (30pts)**:**  -Créer les contrôleurs (HomeController, EnfantController, FavorisController)  -Créer les actions demandées pour chaque contrôleur  -Modifier le constructeur des contrôleurs afin de recevoir le singleton en paramètre  -Validation des paramètres reçue par les actions du contrôleur et réaliser les traitements appropriés  -Sélectionner au niveau des actions du contrôleur la bonne vue à retourner selon le contexte  -Modifier les actions afin d’obtenir les données nécessaires à partir de la BD et les transmettre aux vues | 0 à 12 pts  L'utilisation de contrôleurest **absente** ou plutôt **superficielle** et **ne démontre pas la maitrise** par l'étudiant de ces possibilités.   **Peu d'éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 15 pts  L'utilisation de contrôleur **démontre peu la maitrise** par l'étudiant de ces possibilités.   **La plupart des éléments principaux**  en lien avec les spécifications attendues ont été **réalisés** correctement. | 18 pts  L'utilisation de contrôleur **démontre une maitrise minimale** par l'étudiant de ces possibilités.   **Les principaux éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 24 pts  L'utilisation de contrôleur **démontre une grande maitrise** par l'étudiant de ces possibilités.   **Les principaux éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 30 pts  L'utilisation de contrôleur **démontre une parfaite** **maitrise** par l'étudiant de ces possibilités.    **L'ensemble** des éléments en lien avec les spécifications attendues ont été **réalisés** correctement. |
| **Vues** (35pts) :  -Définir un fichier "ViewStart" et y déposer les blocs de code répétitifs  -Définir un fichier "ViewImport"& y déposer toutes les directives répétitives  -Afficher les informations d'un ViewModel correctement à l'utilisateur  -Afficher les informations d'un Model correctement à l'utilisateur  -Générer une vue non-typée (sans modèle) et l’adapter  -Créer une page de disposition et l'appliquer aux vues  -Modifier les vues non-typée afin qu’elles deviennent typée (@model)  -Écrire des expressions implicites et explicite Razor dans une vue  -Appliquer des concepts de programmation de base  -Utiliser de manière appropriée la directive @using  -Utiliser dans une vue de manière limitée des requêtes LINQ sur des attributs du modèle | 0 à 14 pts  L'utilisation de vues est **absente** ou plutôt **superficielle** et **ne démontre pas la maitrise** par l'étudiant de ces possibilités.   **Peu d'éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 18 pts L'utilisation de vues **démontre peu la maitrise** par l'étudiant de ces possibilités.   **La plupart des éléments principaux**  en lien avec les spécifications attendues ont été **réalisés** correctement. | 21 pts  L'utilisation de vues **démontre une maitrise minimale** par l'étudiant de ces possibilités.   **Les principaux éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 28 pts  L'utilisation de vues **démontre une grande maitrise** par l'étudiant de ces possibilités.   **Les principaux éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 35 pts  L'utilisation de vues **démontre une parfaite** **maitrise** par l'étudiant de ces possibilités.    **L'ensemble** des éléments en lien avec les spécifications attendues ont été **réalisés** correctement. |
| **Modèles** (20pts) :  -Concevoir la structure de la classe “BaseDeDonnees” en respectant les consignes données  -Concevoir les classes qui représenteront le modèle (classe “Parent” et “Enfant”) en respectant les consignes données  -Concevoir le jeu de données nécessaires pour donner des valeurs initialise à la base de données  -Les liaisons entre les entités parent “Parent” et “Enfant” ont bien été déterminé (boucle foreach)  -Concevoir un ViewModel afin de représenter des données attendues par une vue | 0 à 7 pts  L'utilisation de modèles est **absente** ou plutôt **superficielle** et **ne démontre pas la maitrise** par l'étudiant de ces possibilités.   **Peu d'éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 10 pts  L'utilisation de modèles **démontre peu la maitrise** par l'étudiant de ces possibilités.   **La plupart des éléments principaux**  en lien avec les spécifications attendues ont été **réalisés** correctement. | 12 pts  L'utilisation de modèles **démontre une maitrise minimale** par l'étudiant de ces possibilités.   **Les principaux éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 16 pts  L'utilisation de modèles **démontre une grande maitrise** par l'étudiant de ces possibilités.   **Les principaux éléments** en lien avec les spécifications attendues ont été **réalisés** correctement. | 20 pts  L'utilisation de modèles **démontre une parfaite** **maitrise** par l'étudiant de ces possibilités.    **L'ensemble** des éléments en lien avec les spécifications attendues ont été **réalisés** correctement. |

1. Certains critères de recherche étaient obligatoires dans le TP1 et on s’attend à ce qu’ils soient présents dans le TP2. Vous devriez avoir 6 paramètres : 3 pour vos trois checkbox associées aux parents, 1 pour filtrer selon une propriété de type texte, 1 pour filtrer selon vedette ou non (liste déroulante) et 1 libre. [↑](#footnote-ref-2)